

Tecnologie web for dummies

Uniform Resource Identifier (URI)

Le tre caratteristiche cardine del web sono:

- **Identificazione:** Il WWW è uno spazio informativo (per umani e applicazioni) in cui ogni elemento di interesse è chiamato risorsa ed è identificato da un **identificatore globale** chiamato **URI**;
- **Interazione:** Un protocollo di comunicazione chiamato HTTP permette di scambiare messaggi su una rete informatica (nel nostro caso TCP/IP);
- **Formato:** la disponibilità di apposite applicazioni sul computer ricevente e la corretta identificazione del formato dati con cui la risorsa è stata comunicata permette di accedere al suo contenuto in maniera chiara e non ambigua. I formati sono molti, tra cui XHTML, PNG, RDF, ecc.).

In questo paragrafo si tratterà degli URI, ovvero dell'identificazione delle risorse.

RISORSA:

Una **risorsa** è qualunque cosa a cui sia **assegnato un nome**.

Quindi, siccome con gli URI si individua univocamente una risorsa, con essi si può identificare qualunque cosa, anche oggetti non informatici (di fatto è soltanto un nome univoco).

In pratica sul web si definisce risorsa qualunque struttura che sia oggetto di scambio tra applicazioni all'interno del World Wide Web stesso.

Quindi oltre ai file memorizzati in un file system gerarchico, una risorsa può essere:

- un dato in un database, e l'URI essere la chiave di ricerca;
- il risultato dell'elaborazione di un'applicazione, e l'URI essere i parametri di elaborazione;
- una risorsa non elettronica (per esempio un libro, una persona, un pezzo di produzione industriale), e l'URI essere il suo nome Uniforme;
- un concetto astratto (per esempio la grammatica di un linguaggio);

Per questo motivo si usa il termine Risorsa al posto di File e si fornisce per costruire gli URI una sintassi indipendente dal sistema effettivo di memorizzazione.

GLI URI:

Gli URI (Uniform Resource Identifier) sono un concetto inventato con la nascita del web e ne sono stati probabilmente il principale fattore di successo, infatti attraverso essi il web è in grado di identificare risorse accessibili tramite il proprio protocollo (HTTP) e tramite tutti gli altri protocolli esistenti (FTP, Telnet, ecc.).

Berners Lee inoltre li ha ideati in modo che siano stringhe human readable, ovvero semplici, con una loro logica e facilmente memorizzabili.

Sostanzialmente gli **URI** sono una **sintassi universale**, indipendente dal protocollo, facilmente memorizzabile o scambiabile **con cui identificare le risorse di rete**.

SIR TIMOTHY JOHN BERNERS-LEE:
Sir Timothy John Berners-Lee
(Londra, 8 giugno 1955) è un informatico britannico, insignito del premio Turing 2016, coinventore insieme a Robert Cailliau del World Wide Web.
Vedi:
https://it.wikipedia.org/wiki/Tim_Berners-Lee

Gli URI codificano come una stringa di indirizzo secondo un dato protocollo tutte le istruzioni di accesso alle varie specifiche risorse disponibili.

Il web utilizza tali identificatori in un'ampia varietà di modi, per esempio:

- Immagini ed altri oggetti inclusi nei documenti HTML;
- Link ipertestuali;
- Identificatori di namespace (un vocabolario di termini a cui fare riferimento) per documenti XML;
- Identificatori di risorsa su cui esprimere affermazioni;
- identificatori di risorse di cui fornire firme crittografiche o valori hash.

il formato dei dati non è specificato nell'URI, ma può anche variare nel tempo.

Gli Uniform Resource Identifier (**URI**) si dividono in:

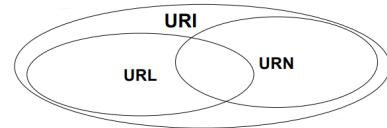
- Uniform Resource Locator (**URL**): contiene tutte le informazioni immediatamente utilizzabili per accedere alla risorsa (ad esempio, il suo indirizzo di rete), ma può con buone probabilità cambiare nel tempo;
- Uniform Resource Names (**URN**): identifica in maniera univoca e permanente una risorsa, ma generalmente deve essere trasformato da un apposito servizio nell'URL attualmente associato alla risorsa.

Nella visione classica, i due insiemi di identificatori erano disgiunti: un URI era o un URL, che riportava informazioni sul protocollo, sul nome dell'host e sul nome locale della risorsa, oppure era un URN, che non riportava queste informazioni perché non persistenti.

Invece nella visione moderna la distinzione tra URL e URN

è secondaria rispetto al concetto di schemi: ogni URI appartiene ad uno schema (la parte della stringa che precede i due punti, vedi il paragrafo SINTASSI DEGLI URI).

Alcuni schemi sono persistenti e possono essere usati per identificare in maniera non ripudiabile delle risorse, altri per natura contengono informazioni dirette per l'accesso e per questo sono più soggetti a modifiche.



Esempio:

il nome di una persona è un name, identifica in maniera univoca (si suppone non ci siamo omonimi) e per tutta la vita una persona.

il numero di telefono è un locator: permette di raggiungere il proprietario chiamandolo, ma può cambiare se il proprietario cambia sim.

Sugli URI si possono fare due diverse operazioni:

- **URI resolution** (risoluzione): è la generazione dell'URL assoluto corrispondente all'URI indicato; si esegue quando l'URI è un URI reference (vedi paragrafo URI REFERENCE) oppure un URI a cui non corrisponde una risorsa fisica (un URI che non è un URL)
 - Input: un URI
 - Output: un URI
- **URI dereferencing** (dereferenziazione): è la fornitura della risorsa identificata dall'URI (ad esempio, il documento cercato)
 - Input: un URL

- Output: una risorsa

SINTASSI DEGLI URI:

gli URI sono stringhe di caratteri prevalentemente ASCII con qualche estensione.

Sono progettati per essere:

- Trascrivibili: sono sequenze di caratteri tratti da un insieme molto limitato (lettere, numeri, pochi caratteri speciali) per permettere la trascrizione degli URI anche su canali non elettronici (un pezzo di carta, un cartellone, ecc.);
- Fornire identificazione, non interazione: le operazioni eseguibili sulle risorse ed i protocolli per eseguirli non sono indicati nell'URI, ma nell'interpretazione che degli URI si fa all'interno dello specifico schema;
- Fornire spazi di nomi organizzati gerarchicamente: i caratteri ":" , "/" , "?" e "#" separano ambiti diversi all'interno degli URI; il primo, onnipresente ed obbligatorio, è lo schema di interpretazione del resto. Schemi gerarchici usano il carattere "/" per separare livelli di specificità diversi del nome locale. "?" separa la parte di identificazione della risorsa da qualunque parametro necessario per accedervi, e "#" separa l'identificativo della risorsa da quello del frammento interno a cui si fa riferimento.

I caratteri ammessi negli URI sono:

curi : unreserved | reserved | escaped

- **unreserved**: i caratteri non riservati sono gli alfanumerici ed alcuni caratteri di punteggiatura privi di ambiguità:
unreserved: uppercase | lowercase | digit | punctuation
dove: (il simbolo separatore "|" è stato omesso per chiarezza)
punctuation: - _ ! ~ * ' ()
- **reserved**: i caratteri riservati sono caratteri che hanno delle funzioni particolari in uno o più schemi di URI; in questo caso vanno usati direttamente quando assolvono alle loro funzioni ed indicati con caratteri di escape quando sono invece parte della stringa identificativa naturale:
reserved: ; / ? : @ & = + \$,
- **escaped**: i caratteri escaped fanno riferimento alle seguenti tipologie di caratteri:
 - I caratteri non US-ASCII (cioè ISO Latin-1 > 127);
 - I caratteri di controllo: US-ASCII < 32;
 - I caratteri unwise: caratteri che possono funzionare correttamente o non funzionare in base alle diverse implementazioni;
unwise: { } | \ ^ []
 - I delimitatori: spazio < > # % "
 - I caratteri riservati quando usati in contesto diverso dal loro uso riservato

In questi casi i caratteri devono essere indicati in maniera escaped, secondo la seguente sintassi:

escaped: %XX, dove XX è il codice esadecimale del carattere

Esempio:

http://www.alpha.edu/uno/due/tre

http://www.alpha.edu/uno/due%2Ftre

non sono uguali, perché, benché il codice esadecimale 2F corrisponda al carattere “/”, nel primo caso esso ha significato gerarchico, e nel secondo fa parte del nome dell’ultima sottoparte della gerarchia.

Un URI è diviso sempre nelle 5 parti seguenti [di cui 3 facoltative]:

URI = schema : [/ authority] path [? query] [# fragment]

- **schema**: è una stringa registrata presso IANA che indica il tipo di connessione che deve essere creato e consente l’accesso ad uno spazio di nomi organizzati gerarchicamente;
- **autorità primaria** (o authority): nome di dominio o indirizzo ip, consente l’accesso ad un ulteriore spazio di nomi organizzati gerarchicamente; l’autorità è a sua volta divisa in:
authority = [userinfo @] host [: port]
 - **userinfo** non deve essere presente se lo schema non prevede identificazione personale;
 - **host** è o un nome di dominio o un indirizzo IP;
 - **port** può essere omessa se ci si riferisce ad una well-known port (per http è la porta 80).

IANA:
Internet Assigned Numbers
Authority
Vedi <https://www.iana.org/>

- identificarsi tramite URI è una pratica attualmente poco sfruttata in quanto significherebbe passare in chiaro sulla rete la propria password (userinfo può contenere anche la password);
- **path**: identificativo della risorsa all’interno dello spazio di nomi dello schema e (se esiste) della authority; generalmente è diviso in blocchi separati da slash “/”;
 - **query**: informazioni aggiuntive passate alla risorsa utilizzate per personalizzare la risposta; tipicamente ha una forma del tipo nome1=valore1&nome2=valore+lungo+molte+parole e generalmente questo è il modo con cui dei valori vengono trasferiti dal form html al server;
 - **fragment**: identificatore di una risorsa secondaria della risorsa primaria (una risorsa associata, dipendente o in molti casi una porzione della risorsa stessa, in quest’ultimo caso il server restituisce tutta la risorsa e la selezione viene fatta dal browser).

Esempi di URI:

http://www.ietf.org/rfc/rfc2396.txt

ftp://ftp.is.co.za/rfc/rfc1808.txt

https://purl.oclc.org/OCLC/PURL/FAQ

file:///Documenti/corsi/tw04/slides/l1.html

telnet://melvyl.ucop.edu/

mailto:John.Doe@example.com

data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUAAAAFCAYAAACNbyblAAA
AHEIEQVQI12P4//8/w38GI AXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJgg==

ROUTE:

Una route è un'associazione della parte path di un URI ad una risorsa gestita o restituita da un server web.

Può essere:

- **Managed route**: il server associa ogni URI ad una risorsa o attraverso il file system locale (risorse statiche) oppure generate attraverso una computazione (risorse dinamiche); è molto di moda oggigiorno con node.js ed express.js;
- **File-system route**: il server associa la radice della parte path ad una directory del file system locale e ogni filename valido all'interno di quella directory genera un URI corretto e funzionante; questo è il vecchio approccio via web server come Apache e le applicazioni server-side di tipo LAMP.

Nel file system route è necessario specificare il formato del file puntato per permettere al sistema di capire come restituire il suo contenuto.

Attualmente si tende ad utilizzare più il managed route del file system.

Esempi:

Una managed route

```
• var router = require("express").Router();  
•  
• function getName(req, res) {  
•   res.send("<p>Fabio</p>");  
• }  
•  
• function getEmail(req, res) {  
•   res.send("fabio.vitali@unibo.it");  
• }  
•  
• router.get("/name", getName);  
• router.get("/email", getEmail);  
• app.use("/css", express.static('css'))
```

URI della richiesta	Risorsa restituita
http://www.example.com/name	<p>Fabio</p>
http://www.example.com/email	fabio.vitali@unibo.it
http://www.example.com/css/style.css	contenuto del file css/style.css

Una file-system route

Organizzazione del file system	URI disponibili
/	-
var/	-
www/	-
index.html	http://www.example.com/
alice/	-
index.html	http://www.example.com/alice/
bruce/	-
index.html	http://www.example.com/bruce/
...	...
fabio/	-
index.html	http://www.example.com/fabio/
pages/	-
doc1.html	http://www.example.com/fabio/pages/doc1.html
doc2.html	http://www.example.com/fabio/pages/doc2.html
index.html	http://www.example.com/fabio/pages/
img/	-
img1.gif	http://www.example.com/fabio/img/img1.gif
img2.jpg	http://www.example.com/fabio/img/img2.jpg
css/	-
style.css	http://www.example.com/fabio/css/style.css

^-- javascript

URI REFERENCE (URI REF):

Un URI assoluto contiene tutte le parti predefinite dal suo schema, esplicitamente precise.

Un URI gerarchico può però anche essere relativo (detto tecnicamente un **URI reference**) ed in questo caso riportare solo una parte dell'URI assoluto corrispondente, tagliando progressivamente cose da sinistra.

Un URI ref quindi è un frammento di URI che permette in maniera algoritmica di risalire all'URI completo, questo processo è detto **risoluzione**.

Un URI ref non ha tutte le informazioni necessarie per raggiungere la risorsa puntata, però dato un **URI di base** (ad esempio, l'URI assoluto del documento ospitante l'URI reference) si può risalire all'URI completo.

Esempio:

l'URL reference pippo.html posto dentro al documento di URI

http://www.sito.com/uno/due/pluto.html fa riferimento al documento il cui URI assoluto è http://www.sito.com/uno/due/pippo.html

Risolvere un URI relativo significa identificare l'URI assoluto cercato sulla base dell'URI relativo stesso e, di solito, dell'URI di base. Questo avviene come segue:

	Dato il base URI http://www.site.com/dir1/doc1.html
Se inizia con "#", è un frammento interno allo stesso documento di base	#anchor1 si risolve come http://www.site.com/dir1/doc1.html#anchor1
Se inizia con uno schema, è un URI assoluto	http://www.site.com/dir2/doc2.html si risolve come http://www.site.com/dir2/doc2.html
Se inizia con "/", allora è un path assoluto all'interno della stessa autorità del documento di base, e gli va applicata la stessa parte autorità.	/dir3/doc3.html si risolve come http://www.site.com/dir3/doc3.html

Altrimenti:	Dato il base URI http://www.site.com/dir1/doc1.html
Altrimenti, si estrae il path assoluto dell'URI di base, meno l'ultimo elemento, e si aggiunge in fondo l'URI relativo.	doc4.html si risolve come http://www.site.com/dir1/doc4.html
Si procede infine a semplificazioni:	dir5/doc5.html si risolve come http://www.site.com/dir1/dir5/doc5.html
"/" (stesso livello di gerarchia): viene cancellata	./doc6.html si risolve come http://www.site.com/dir1./doc6.html che è equivalente a http://www.site.com/dir1/doc6.html
".." (livello superiore di gerarchia): viene eliminato insieme all'elemento precedente.	../doc7.html si risolve come http://www.site.com/dir1/..doc7.html che è equivalente a http://www.site.com/doc7.html

Per approfondimenti vedere <https://www.ietf.org/rfc/rfc2396.txt> sezione 5.

Esempio:

Dato l'URI base: <http://www.sito.com/uno/due/tre>, i seguenti URI relativi diventano:

URI relativo	URI assoluto
piippo:pluto	piippo:pluto
piippo	http://www.sito.com/uno/due/piippo
./piippo	http://www.sito.com/uno/due/piippo
#pluto	(risorsa attuale)#pluto
piippo#pluto	http://www.sito.com/uno/due/piippo#pluto
.	http://www.sito.com/uno/due/
/	http://www.sito.com/uno/due/
..	http://www.sito.com/uno/

.. /	http://www.sito.com/uno/
.. / pippo	http://www.sito.com/uno/pippo
... / ..	http://www.sito.com/
... / .. /	http://www.sito.com/
... / .. / pippo	http://www.sito.com/pippo

HTTP E HTTPS:

Sono i due protocolli più usati nel web.

Sono due schemi quasi identici, tra i quali cambia il tipo di connessione e la well known port, inoltre **HTTPS** prevede una la presenza di **crittografia** in entrambi i sensi della comunicazione.

La loro sintassi è:

- http://host[:port]/path[?query][#fragment]
- https://host[:port]/path[?query][#fragment]

dove:

- **host** è l'indirizzo TCP-IP o DNS dell'host in cui si trova la risorsa;
- **port** è la porta a cui il server è in ascolto per le connessioni; di default la porta è 80 per HTTP e 443 per HTTPS;
- **path** è un pathname gerarchico per l'identificazione della risorsa;
- **query** è una frase che è l'oggetto di una ricerca sulla risorsa specificata;
- **fragment** è un identificativo di una sottoparte dell'oggetto; la definizione e il ritrovamento di queste sottoparti è a carico del client, quindi la parte di fragment viene ignorata dal server che restituisce l'intero oggetto.

LO SCHEMA FILE (RFC 8089):

un browser può utilizzare gli URI per accedere ai **file salvati in locale** nel computer utilizzando lo schema "file".

È equivalente a scegliere "Apri file" nel menu del browser.

La sintassi è:

file://host/path [#fragment]

La parte host può essere eliminata assumendo che sia localhost; ciò porta alla sintassi più frequente:

file:///path

ATTENZIONE:

MS Windows accetta come separatore anche "\", che però non è nello standard approvato!

Esempio: file:///c:/Users/mario/Pictures/img1.jpg

LO SCHEMA DATA (RFC 2397):

È uno schema non gerarchico che non fa riferimento ad una risorsa ma la CONTIENE: tutti i dati della risorsa sono inseriti nell'URL vero e proprio.

È usato per immagini (e non solo) inline su cui non si vuole attivare una connessione HTTP separata.

La sintassi è:

- **data:[<media type>][;base64],<data>**
- **media type** è un media type MIME registrato presso IANA;
- In più si può codificare il dato, di solito usando **base64**;
- Poi ci sono i **data** della risorsa, ovvero il contenuto vero e proprio.

Esempio:

```
data:text/plain;charset=UTF-8;some%20text%20for%20you
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUAAAAFCAYAAACNbyblAAA
AHEIEQVQI12P4//w38GIAXDIBKE0DHxgljN BAAO9TXL0Y4OHwAAAABJRU 5ErkJgg==
```

FTP:

è un protocollo standard di trasferimento file che si può usare sia in maniera autenticata che anonima.

La sintassi della parte specifica è:

```
ftp://[user[:password]@]host[:port]/path
```

dove:

- **User e password** sono utente e password per l'accesso ad un server FTP; la mancanza di user fa partire automaticamente una connessione anonima;
- **Host, port e path** sono l'indirizzo del server, la porta di connessione (la porta di default è 21) ed il nome del file dell'oggetto ricercato come per HTTP.

Attualmente si tende a scoraggiare l'uso della password nell'URI, in quanto è un'evidente situazione di scarsa sicurezza (la password viene passata in maniera non crittografata nel web), tuttavia lo schema lo prevede come parte facoltativa.

URI CHE INIZIANO CON // :

È un URI ref che omette soltanto lo schema.

per problemi di sicurezza i browser moderni ostacolano l'accesso http ad una pagina https e viceversa, quindi per mantenere lo stesso schema della pagina precedente lo si omette e si lascia la scelta del protocollo al processo di risoluzione.

Un URI reference può quindi iniziare con un nome dominio, del tipo:

```
//www.sito.com/dir1/dir2/fig1.gif
```

Esso deve essere risolta in un URI assoluto utilizzando le parti mancanti dell'URI base, che in questo caso sarebbe la pagina HTML ospitante il tag.

Quindi questo significa: "carica l'immagine fig1.gif utilizzando lo stesso protocollo della pagina HTML: HTTP se siamo fuori dall'area protetta e HTTPS se siamo dentro l'area protetta".

È prassi utilizzare questo tipo di URI nelle chiamate CDN (vedi prossimo paragrafo):

```
//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js
```

```
//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.j
```

CONTENT DELIVERY NETWORK (CDN):

È una rete fortemente distribuita di server commerciali che collaborano tra loro per distribuire in maniera omogenea contenuti di grande successo, senza inutili duplicazioni di occupazione e trasmissione di file.

Un uso corretto dei CDN permette di sfruttare al meglio anche le capacità di caching delle macchine degli utenti finali o dei nodi intermedi della rete.

Ad esempio ogni utente che naviga su molti siti web, che usano le stesse librerie importanti (ad esempio, Bootstrap, JQuery etc.) ma non le prendono da un CDN, deve scaricare tutte le volte gli stessi identici file; se questi siti invece fanno riferimento a librerie sullo stesso CDN, esse verranno scaricate una volta sola e rimarranno in cache anche per gli altri siti. Questo sistema però permette ai proprietari dei server di tracciare gli utenti nei siti che navigano e di utilizzare queste informazioni anche a scopo commerciale.

URL PERMANENTI:

Esistono molti schemi di URL che forniscono (o quantomeno promettono) le garanzie richieste ad un URL per essere un URN, in particolare la permanenza.
In tutti i casi, un sistema server-side prende un URL a cui non corrisponde una risorsa fisica e identifica quale sia l'URL fisico corretto per quella risorsa.

L'accesso può avvenire per:

- **Dereferenziazione:** il sistema converte l'URI virtuale nell'URI fisico, accede alla risorsa e la restituisce al richiedente;
- **Redirezione:** il sistema fornisce al richiedente una risposta speciale (302 temporary redirect) e l'URI che oggi (ma è detto lo sia per sempre) permette di accedere alla risorsa corretta.

Esempio:

PURL (permanent URL) o i permalink offerti da molte piattaforme di blog ecc...

URI REWRITING:

Anche un routing di tipo file system può essere trasformato in un routing gestito, attraverso degli helper di routing.

Ad esempio Apache, come anche molti altri server web, fornisce un modulo di riscrittura degli URL, chiamato mod_rewrite.

Il rewriting trasforma un URI visibile in un URL fisico sulla base di specifiche regole.

Per esempio:

<http://example.com/wiki/index.php?title=Argomento>
può essere esposto all'esterno semplicemente come
<http://example.com/Argomento>

Vantaggi:

- Permette di nascondere dettagli dell'implementazione;
- Permette di realizzare sistemi di nomi perduranti oltre la vita utile del software utilizzato;
- Permette di esprimere informazioni semantiche sulla struttura del sito invece che sulle tecnologie usate.

Una variante è l'URI aliasing, che permette di accedere alla risorsa con entrambi gli URI (quello originale e quello riscritto).

URI SHORTENER:

Con la nascita di twitter ed il suo limite dei 140 caratteri, sorge il problema di inserire nei tweet link ad URL anche piuttosto lunghi.

Servizi come bit.ly, tr.im o goo.gl permettono di creare URL molto brevi corrispondenti a URL molto lunghi.

Il servizio è un semplice rewrite (redirect, per essere più precisi) che automaticamente crea un nome opaco molto breve.

Esempio:

goo.gl/T655I reindirizza al sito

http://www.ted.com/talks/tim_berners_lee_the_year_open_data_went_worldwide.html

application/x-www-form-urlencoded :

E' un'estensione della codifica degli URI applicata anche a risorse trasmesse su un canale HTTP (ad esempio il contenuto di un POST).

I codici non alfanumerici sono sostituiti da '%HH' (HH: codice esadecimale del carattere), gli spazi sono sostituiti da '+', i nomi dei controlli sono separati da '&' ed il valore è separato dal nome da '='.

Generalmente la parte query di un URI usa il formato urlencoded.

Esempio:

<http://mysite.com/serversidescript?user=Fabio+Vitali&pwd=%40%40%40>

Anche i form che trasmettono dati al server (metodo POST) possono usare questo tipo di formato dati nel body della richiesta.

Esempio:

[user=Fabio+Vitali&pwd=%40%40%40](http://mysite.com/serversidescript?user=Fabio+Vitali&pwd=%40%40%40)

INTERNATIONALIZED RESOURCE IDENTIFIER (IRI):

Gli URI possono usare solo caratteri ASCII 7 bit (e neanche tutti), ma attraverso escaping è possibile accedere a tutti i caratteri ISO Latin-1.

Ovviamente questo è sgradevole per tutte le culture che usano caratteri non latini o per cui le estensioni di caratteri latini siano fondamentali.

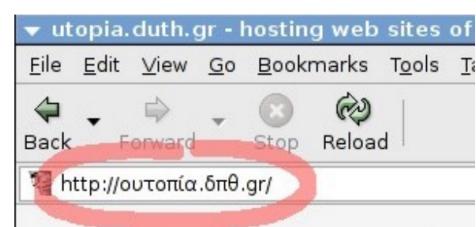
IRI (Internationalized Resource Identifier), RFC 3987 di Gennaio 2005, fornisce una sintassi per inserire caratteri di UCS-4 in un URI e per risolverli.

In realtà non tutti i caratteri sono ammessi: non sono accettati tutti i codici di controllo e tutti i codici non corrispondenti a caratteri usati nelle lingue.

Per poter far funzionare gli IRI è però necessario discutere dei nomi di dominio internazionalizzati (IDN).

Internationalized Domain Name (IDN):

Internationalized Domain Name (IDN), o anche Internationalized Domain Name for Application (IDNA), è uno standard IETF per estendere i nomi di dominio a tutti i caratteri non ASCII di Unicode (RFC 3490). Questi nuovi nomi di dominio vengono detti IDN.



RISCHI DI IDN (ATTACCO VIA OMOGRAFI):

Uno dei rischi più importanti di IDNA è il fatto che molti caratteri di vari script si assomigliano molto.

Ad esempio, i seguenti caratteri sono in cirillico: а с е и ј о п с љ

Questo può permettere a utenti maliziosi di registrare nomi di dominio sospettosamente simili a nomi noti e fare spoofing (cioè impersonificazione).

Esempio:

il dominio "http://paypal.com" contiene 2 a in cirillico, ma è ovviamente indistinguibile da "http://paypal.com"

Per ovviare a questi inconvenienti alcuni registri di nomi di dominio non accettano domini contenenti mix di caratteri di script diversi, ma ad ogni modo questa non è una risoluzione definitiva e può capitare che utenti malevoli riescano comunque nel loro intento.

COMPACT URI (CURIE):

In molti casi, la lunghezza degli URI può diventare considerevole, e magari URI molto simili possono essere presenti in molte parti diverse di una stessa risorsa, richiedendo uno spreco inutile di spazio e un rischio frequente di errori di compilazione.

I CURIE (Compact URI) sono un modo per esprimere in maniera compatta famiglie di URI che condividono lo stesso prefisso.

Hanno una sintassi del tipo:

[prefix:curie]

(le parentesi quadre e i due punti sono obbligatori)

Il **prefisso** è associato (in uno di molti modi, ad esempio con XML namespaces) ad un URI assoluto che deve essere sostituito nel CURIE per ottenere l'URI cercato.

Esempio:

```
<html xmlns:dom="http://www.dominio.com/">
    <p>Si veda <a href="[dom:doc1]">doc 1</a></p>
</html>
```

In questo caso, il CURIE [dom:doc1] viene risolto nell'URI <http://www.dominio.com/doc1>

LINKED DATA:

Oltre alle pagine web ed alle applicazioni web, esistono in rete quantità incredibili di informazioni strutturate che sarebbe utile mettere a disposizione non sotto forma di pagine HTML pensate per la lettura da parte di umani, ma come dati accessibili e manipolabili equivalentemente da umani e software.

Il nome Linked Data è stato proposto da Tim Berners-Lee nel 2009 come modello concettuale per la caratterizzazione di tutte le risorse di dati che possono essere messi a disposizione ed incrociati da applicazioni ed umani.

Per essere Linked Data, una collezione di dati deve:

- Usare URI per identificare oggetti;
- Usare HTTP URI in modo che questi oggetti possano essere referenziati e cercati da persone e user agent;
- Fornire informazioni utili sull'oggetto quando la sua URI è dereferenziata, usando formati standard come per esempio RDF;
- Includere link ad altre URI relative ai dati esposti per migliorare la ricerca di altre informazioni relative nel Web.

LINKED OPEN DATA:

I LOD sono Linked Data di tipo aperto: "Linked Open Data (LOD) is Linked Data which is released under an open license, which does not impede its reuse for free." (Tim Berners-Lee)

Sono diventati un argomento politico di grande interesse da quando alcune organizzazioni politiche (di tutto lo spettro politico) ne hanno fatto il centro di campagne per favorire la trasparenza delle pubbliche amministrazioni e l'accesso della cittadinanza alle informazioni e ai processi decisionali delle pubbliche amministrazioni.

HyperText Transfer Protocol (HTTP)

HTTP permette lo scambio di risorse identificate da URI.

HTTP è un protocollo client-server, generico e stateless utilizzato non solo per lo scambio di documenti Web ma anche in molte applicazioni distribuite.

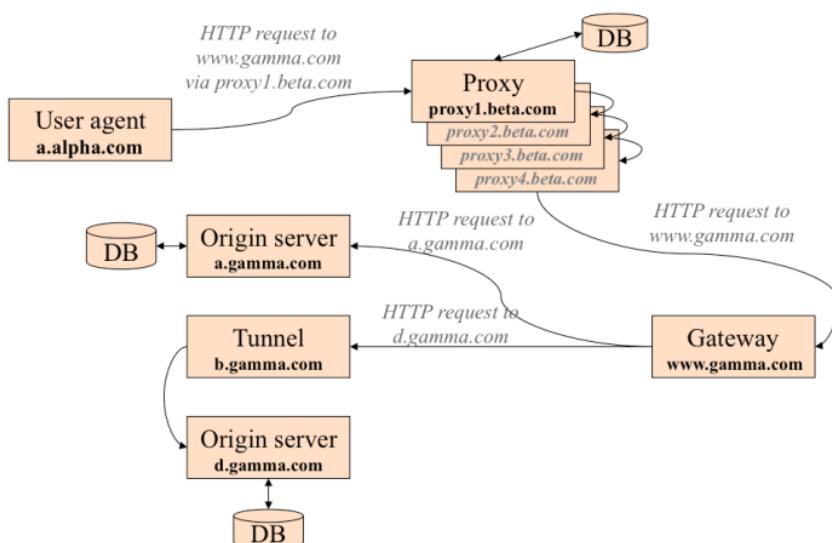
Nello specifico è:

- **Client-server**: il client attiva la connessione e richiede dei servizi; il server accetta la connessione, nel caso identifica il richiedente, e risponde alla richiesta; alla fine chiude la connessione;
- **Generico**: HTTP è indipendente dal formato dati con cui vengono trasmesse le risorse, può funzionare per documenti HTML come per binari, eseguibili, oggetti distribuiti o altre strutture dati più o meno complicate (esiste un meccanismo detto **negoziazione** che permette di scegliere come visualizzare risorsa);
- **Stateless**: Il server non è tenuto a mantenere informazioni che persistano tra una connessione e la successiva sulla natura, identità e precedenti richieste di un client (cosa che in realtà comunque accade spesso con i cookie, che però non sono http); il client è quindi tenuto a ricreare da zero il contesto necessario al server per rispondere.

HTTP implementa inoltre sofisticate politiche di **caching** che permettono di memorizzare copie delle risorse sui server (proxy, gateway, ecc...) coinvolti nella trasmissione e controllare in modo accurato la validità di queste copie.

RUOLI DELLE APPLICAZIONI HTTP:

- **Client**: un'applicazione che stabilisce una connessione HTTP, con lo scopo di mandare richieste;
- **Server**: un'applicazione che accetta connessioni HTTP e genera risposte;
- **User agent**: il client che inizia una richiesta HTTP (tipicamente un browser, ma può anche essere un bot);
- **Origin server**: il server che possiede fisicamente la risorsa richiesta.



CONNESSIONE HTTP:

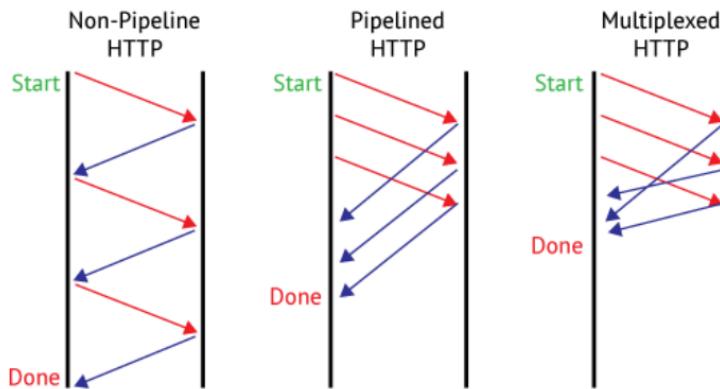
Una connessione HTTP è composta da una serie di **richieste** ed una serie corrispondente di **risposte**.

Queste connessioni sono **persistenti**, ovvero consentono di trasferire più file con un'unica connessione TCP (la connessione TCP non viene chiusa immediatamente).

In particolare vengono usate alternativamente due tecniche per rendere più veloce la comunicazione:

- **Pipelining**: trasmissione di più richieste senza attendere l'arrivo della risposta alle richieste precedenti, ma le risposte sono restituite nello stesso ordine delle richieste;
- **Multiplexing**: nella stessa connessione è possibile avere richieste e risposte multiple, restituite anche in ordine diverso rispetto alle richieste e successivamente "ricostruite" nel client.

Nella costruzione di una pagina ci sono molte richieste HTTP perché generalmente tale pagina è composta da più risorse e si deve fare una richiesta per ognuna di esse (html, css, immagini, ecc...).



HTTP/2:

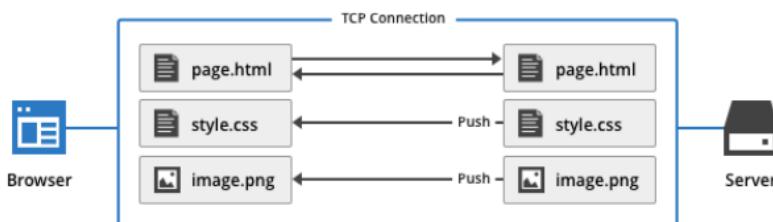
Inizialmente chiamato HTTP 2.0 è la seconda major revision di HTTP ed è stato standardizzato da IETF nel RFC 7540 nel maggio 2015 (HTTP1.1 è del 1997!).

È basato su SPDY, il protocollo proposto da Google per ridurre i tempi di latenza di HTTP, ma è stato poi generalizzato e migliorato sulla base del feedback di altri players.

Non è una riscrittura del protocollo, i concetti principali e la compatibilità con HTTP 1.x restano gli stessi; infatti con http1.1, http/2 e poi http/3 si possono fare le stesse cose, ma ci sono molte ottimizzazioni che migliorano le prestazioni.

HTTP/2 ha infatti introdotto molte novità per migliorare le performance:

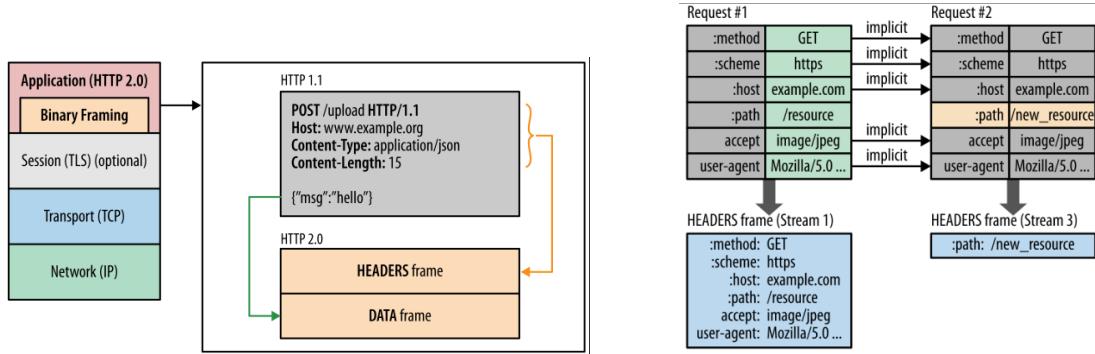
- il già citato **multiplexing**;
- supporto per operazioni di **PUSH**: il server anticipa le richieste successive del client (spedisce in anticipo le risorse prima di ricevere le richieste), ciò si fa quando è quasi sicuro quale sarà la richiesta successiva;



- **compressione** degli header: HTTP/2 ha gli stessi ruoli, verbi e headers di HTTP/1.1, tuttavia è un protocollo binario in cui i messaggi non sono più in chiaro, ma sono codificati in maniera compressa e separando il blocco degli header dal payload, in questo modo i pacchetti spediti sono più leggeri; inoltre ogni flusso di dati (stream) è

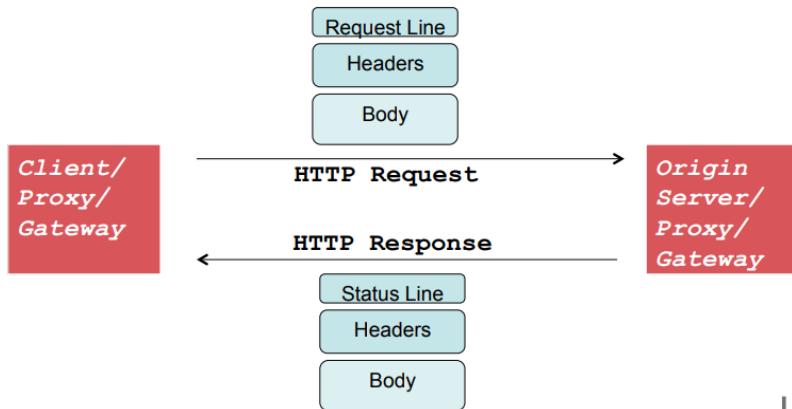
suddiviso in frame separati e identificati che viaggiano anche sovrapposti (multiplexing);

- **Binary framing:** solo gli header diversi da quelli delle richieste precedenti vengono inseriti nel frame e spediti.



Struttura di HTTP:

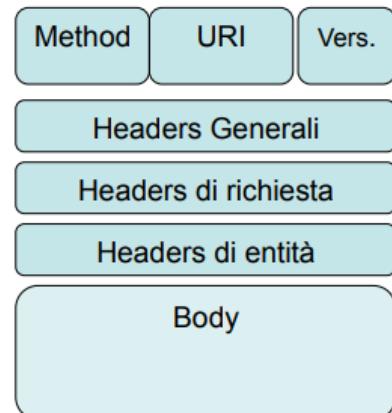
Richieste e risposte sono file di testo con struttura molto simile



STRUTTURA DELLA RICHIESTA:

La richiesta HTTP si compone di:

- **Method**: azione del server richiesta dal client;
- **URI**: identificativo della risorsa locale al server;
- **Version**: numero di versione di HTTP utilizzata;
- **Header**: sono linee RFC822 divise in
 - header generali;
 - header di entità;
 - header di richiesta.
- **Body**: è un messaggio MIME, contiene la vera e propria risorsa.



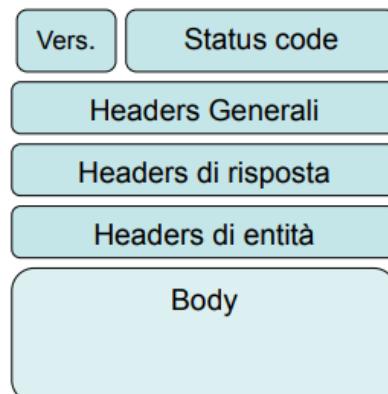
Esempio di richiesta:

```
GET /beta.html HTTP/1.1
Referer: http://www.alpha.com/alpha.html
Connection: Keep-Alive
User-Agent: Mozilla/4.61 (Macintosh; I; PPC)
Host: www.alpha.com:80
Accept: image/gif, image/jpeg, image/png, */
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

STRUTTURA DELLA RISPOSTA:

La risposta HTTP si compone di:

- **Status code**: indica se la richiesta è andata a buon fine o meno;
- **Version**: numero di versione di HTTP utilizzata;
- **Header**: sono linee RFC822 divise in
 - header generali;
 - header di entità;
 - header di risposta.



- **Body:** è un messaggio MIME, contiene la vera e propria risorsa.

Esempio di risposta:

RICHIESTA:

```
GET /index.html HTTP/1.1
```

```
Host: www.cs.unibo.it:80
```

RISPOSTA:

```
HTTP/1.1 200 OK
```

```
Date: Fri, 26 Nov 2007 11:46:53 GMT
```

```
Server: Apache/1.3.3 (Unix)
```

```
Last-Modified: Mon, 12 Jul 2007 12:55:37 GMT
```

```
Accept-Ranges: bytes
```

```
Content-Length: 3357
```

```
Content-Type: text/html
```

```
<HTML> ... </HTML>
```

STATUS CODE:

Lo status code più famoso è il codice di errore 404.

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

Le classi sono:

- **Informational** (1xx): una risposta temporanea alla richiesta, durante il suo svolgimento;
- **Successful** (2xx): il server ha ricevuto, capito e accettato la richiesta;
- **Redirection** (3xx): la richiesta è corretta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta;
- **Client error** (4xx): la richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata);
- **Server error** (5xx): la richiesta può anche essere corretta, ma il server non riesce a soddisfarla per un problema interno.

L'uso corretto degli status code aiuta a costruire API chiare e più semplici da usare:

- Il client non ha necessità di leggere il body della risposta ma già dallo status code può capire se la richiesta è andata a buon fine, poi può comunque trovare dettagli nel body ma non è essenziale.
- Permette a tutte le entità coinvolte nella comunicazione di capire meglio cosa succede, e sfruttare al meglio i meccanismi di caching e redirezione di HTTP.
- Inoltre migliora uniformità e interoperabilità.

Esempi:

100	Continue (se il client non ha ancora mandato il body)
200	Ok (GET con successo)
201	Created (PUT con successo)
301	Moved permanently (URL non valida, il server conosce la nuova posizione)
400	Bad request (errore sintattico nella richiesta)

401	Unauthorized (manca l'autorizzazione)
403	Forbidden (richiesta non autorizzabile)
404	Not found (URL errato)
500	Internal server error (tipicamente un errore nel codice server-side)

HEADER:

Gli header sono righe di testo (RFC822) che specificano informazioni aggiuntive, sono presenti sia nelle richieste che nelle risposte e ne descrivono diversi aspetti.

Si dividono in 4 categorie, la cui struttura è però sempre la stessa:

- **header generali:**

Gli header generali si applicano al messaggio trasmesso (che sia una richiesta o una risposta), ma non necessariamente alla risorsa trasmessa.

Esempi:

- Date: data ed ora della trasmissione;
- Transfer-Encoding: il tipo di formato di codifica usato per la trasmissione;
- Cache-Control: il tipo di meccanismo di caching richiesto o suggerito per la risorsa;
- Connection: il tipo di connessione da usare (tenere attiva, chiudere dopo la risposta, ecc.).

- **header di entità:**

Gli header dell'entità danno informazioni sul body del messaggio o, se non vi è body, sulla risorsa specificata.

Esempi:

- Content-Type: il tipo MIME dell'entità nel body. Questo header è obbligatorio in ogni messaggio che abbia un body;
- Content-Length: la lunghezza in byte del body. Obbligatorio, soprattutto se la connessione è persistente;
- Content-Encoding, Content-Language, Content-Location, Content-MD5, Content-Range: la codifica, il linguaggio, l'URL della risorsa specifica, il valore di digest MD5 e il range richiesto della risorsa.

- **header di richieste:**

Gli header della richiesta sono posti dal client per mandare specifiche informazioni sulla richiesta e su se stesso al server.

Esempi:

- User-Agent: una stringa che descrive il client che origina la richiesta, tipicamente tipo, versione e sistema operativo del client;
- Referer: l'URL della pagina mostrata all'utente mentre richiede il nuovo URL;
- Host: il nome di dominio e la porta a cui viene fatta la connessione.

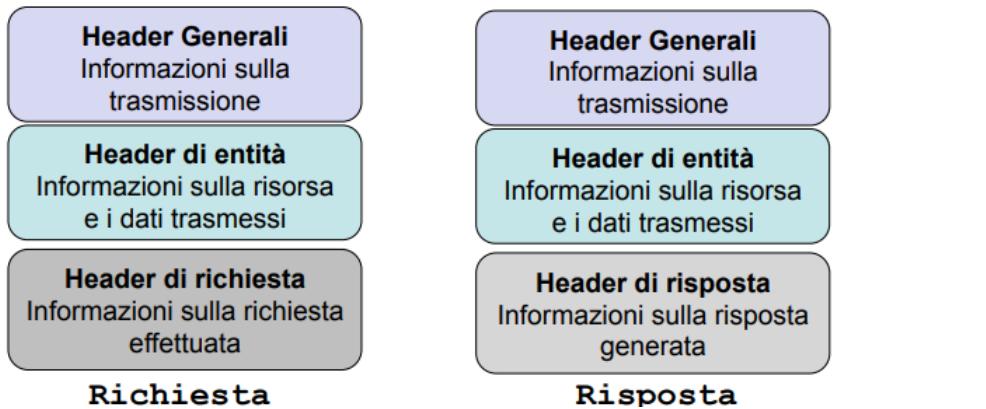
Altri header della richiesta sono usati per gestire la cache ed i meccanismi di autenticazione.

- **header di risposta:**

Gli header della risposta sono posti dal server per mandare specifiche informazioni sulla risposta e su se stesso al client.

Esempi :

- Server: una stringa che descrive il server: tipo, sistema operativo e versione;
- WWW-Authenticate: challenge (serie di informazioni) utilizzata per i meccanismi di autenticazione.



CONTENT-TYPE:

Tra gli header della risposta sono particolarmente utili quelli che definiscono il tipo di dato contenuto nella risposta; infatti queste informazioni permettono al client di processare correttamente la risorsa.

Quindi se viene fornita un'entità in risposta gli header **Content-type** e **Content-length** sono obbligatori: è solo grazie al content type che lo user agent sa come visualizzare l'oggetto ricevuto, mentre è solo grazie al content length che lo user agent sa che ha ricevuto tutto l'oggetto richiesto.

Metodi di HTTP:

I **metodi** HTTP (anche detti verbi HTTP) indicano l'**azione** che il client richiede al server sulla **risorsa** (o meglio, sulla rappresentazione della risorsa, o meglio ancora sulla copia della rappresentazione della risorsa).

Un uso corretto di tali metodi aiuta a creare applicazioni interoperabili e in grado di sfruttare al meglio i meccanismi di caching di HTTP.

In generale sono valutati in base a due proprietà principali:

- **Sicurezza:** un metodo è sicuro se non genera cambiamenti allo stato interno del server (a parte ovviamente nei log); un metodo sicuro può essere eseguito da un nodo intermedio (es. una cache) senza effetti negativi;
- **Idempotenza:** un metodo è idempotente se l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta (sempre tralasciando i log); un metodo idempotente può essere ri-eseguito da più agenti o in più tempi diversi senza effetti negativi.

INTEROPERABILITÀ:

Capacità di due o più sistemi, reti, mezzi, applicazioni o componenti, di scambiare informazioni tra loro e di essere poi in grado di utilizzarle.

vedi

[https://www.treccani.it/enciclopedia/interoperabilita_\(Encyclopaedia-della-Scienza-e-della-Tecnica\)](https://www.treccani.it/enciclopedia/interoperabilita_(Encyclopaedia-della-Scienza-e-della-Tecnica))

I metodi principali sono: GET, HEAD, POST, PUT, DELETE, OPTIONS, PATCH (ce ne sono altri ma noi non li trattiamo);

IL METODO GET:

Il più importante (ed unico fino alla versione 0.9) metodo di HTTP è GET, che richiede una risorsa ad un server.

Questo è il metodo più frequente ed è quello che viene attivato facendo click su un link ipertestuale di un documento HTML o specificando un URL nell'apposito campo del browser. GET è sicuro ed idempotente, infatti i dati che restituisce possono cambiare unicamente se vengono cambiati a livello server.

Esempio:

```
GET /courses/tw.html  
GET /students/123456  
GET /students/123456/exams/
```

IL METODO HEAD:

Il metodo HEAD è simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi senza il corpo.

Viene usato per verificare validità, accessibilità e coerenza in cache di un URI.

HEAD è sicuro e idempotente.

Esempio:

```
HEAD /courses/tw.html
```

IL METODO POST:

Il metodo POST serve per trasmettere informazioni dal client al server relative alla risorsa identificata nell'URI, ma può essere usato anche per creare nuove risorse.

Viene usato per esempio per spedire i dati di un form HTML ad un'applicazione server-side.

POST non è sicuro né idempotente.

Esempio:

```
POST /courses/1678
{
  "titolo": "Tecnologie Web",
  "descrizione": "Il corso..bla..bla.."
}
```

IL METODO PUT:

Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata.

In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome.

Attenzione: non offre garanzie di controllo degli accessi o locking.

PUT è idempotente ma non sicuro.

Esempio:

```
PUT /courses/1678
{
  "id": 1678,
  "titolo": "Tecnologie Web",
}
```

IL METODO DELETE:

Il metodo DELETE serve a rimuovere le informazioni connesse ad una risorsa.

Dopo l'esecuzione di un DELETE, la risorsa non esiste più ed ogni successiva richiesta di GET risulterà in un errore 404 Not Found.

Invocare questo metodo su una risorsa non esistente è lecito e non genera un errore.

DELETE è idempotente e non sicuro.

Esempio:

```
DELETE /courses/1678
```

IL METODO PATCH:

Il metodo PATCH è usato per aggiornare parzialmente la risorsa identificata dall'URI. Indica quali modifiche effettuare su una risorsa esistente; è usato quindi per indicare modifiche incrementali e non sovrascrivere intere risorse nel server, come invece fa PUT. PATCH non è né sicuro né idempotente.

Esempio:

```
PATCH /courses/1678
{
  "op" : "update"
  "cfu": "6"
}
```

IL METODO OPTIONS:

OPTIONS viene usato per verificare opzioni, requisiti e servizi di un server, senza implicare che seguirà poi un'altra richiesta.

È usato per il problema del cross-site scripting: CORS.

Riassumendo le proprietà principali:

HTTP Methods	IDEMPOTENT	SAFE METHOD
GET	YES	YES
HEAD	YES	YES
OPTION	YES	YES
DELETE	YES	NO
PUT	YES	NO
PATCH	NO	NO
POST	NO	NO

REpresentational State Transfer (REST)

REST è l'acronimo di REpresentational State Transfer, ed è il modello architetturale che sta dietro al World Wide Web e in generale dietro alle applicazioni web “**ben fatte**” secondo i progettisti di HTTP.

Un'applicazione REST si basa fondamentalmente sull'uso dei protocolli di trasporto (HTTP) e di naming (URI), per generare interfacce **generiche** di interazione con l'applicazione e **fortemente connesse** con l'ambiente d'uso.

Viceversa, applicazioni non REST si basano sulla generazione di un'API che specifica le funzioni messe a disposizione dell'applicazione, nonché sulla creazione di un'interfaccia indipendente dal protocollo di trasporto e ad esso completamente nascosta.

API WEB:

Un'API Web descrive un'**interfaccia HTTP** che permette ad applicazioni terze di utilizzare i servizi dell'applicazione che la implementa.

Queste applicazioni terze possono essere:

- Applicazioni automatiche che utilizzano i dati della mia applicazione;
- Applicazioni Web che mostrano all'utente un menù di opzioni, magari anche un form, e gli permettono di eseguire un'azione sui dati della mia applicazione.

Un'API web deve soddisfare alcune specifiche caratteristiche per essere considerata REST. Idealmente, se avesse tutte queste caratteristiche, il programmatore dell'applicazione serverside non dovrebbe nemmeno curarsi di sapere se la richiesta gli arriva da un modulo interno dell'applicazione o da una richiesta esterna via HTTP, perché vengono trattate in maniera analoga e nascoste agli occhi del programmatore.

IL MODELLO CRUD:

È un pattern tipico delle applicazioni di trattamento dei dati utilizzato anche da REST.

Assume che tutte le operazioni possibili sui dati siano:

- **Create:** inserimento nel database di un nuovo record;
Crea un cliente il cui nome è "Rossi Spa", il telefono "051 654321", la città "Bologna" e restituisce il codice identificatore che è 4123.
- **Read:** accesso in lettura al database, può essere:
 - individuale;dammi la scheda del cliente con id=4123;
 - contenitore;dammi la lista dei clienti la cui proprietà città è uguale al valore "Bologna";
- **Update;**Cambia il numero di telefono del cliente il cui id=4123 in "051 123456";
- **Delete;**Rimuovi dal database il cliente con id=4123;

STRUTTURA DI REST:

L'architettura REST si basa su quattro punti:

- Definire come **risorsa** ogni concetto rilevante dell'applicazione Web;
- Associargli un **URI** come identificatore e selettore primario;

- Usare i verbi HTTP per esprimere ogni operazione dell'applicazione secondo il **modello CRUD**:
 - creazione di un nuovo oggetto (metodo **PUT**);
 - visualizzazione dello stato della risorsa (metodo **GET**);
 - cambio di stato della risorsa (metodo **POST**);
 - cancellazione di una risorsa (metodo **DELETE**);
- Esprimere in maniera **parametrica** ogni rappresentazione dello **stato interno** della risorsa, personalizzabile dal richiedente attraverso un **Content Type** preciso.

Tale architettura si poggia anche sui tre presupposti del Web, sui quali il sistema è stato progettato sin dall'inizio:

- Che venga definita come risorsa ogni entità, ed associato ad essa un URI come identificatore (**il nome**);
- Che ogni interazione con un'applicazione Web sia esprimibile con un metodo HTTP (**il verbo**);
- Che ciò che viene scambiato tra gli attori dell'interazione sia una rappresentazione di uno stato della risorsa specificato attraverso un Content-Type (**il formato**).

REST infatti non è un nuovo protocollo, ma solo un modo di strutturare le applicazioni per sfruttare a pieno le caratteristiche di HTTP.

Invece nelle applicazioni non REST, progettate secondo il modello tradizionale SOAP (Simple Object Access Protocol), esiste un intermediario di messaggi che:

- raccoglie una richiesta via HTTP;
- ne interpreta i parametri;
- chiama la funzione interna corretta con i parametri corretti;
- riceve il valore di ritorno;
- lo ritrasmette al cliente originale.

La richiesta è indirizzata a questo intermediario e contiene nel body tutte le informazioni necessarie per soddisfarla ed il protocollo HTTP è usato solo per trasferire le informazioni in modo trasparente.

Esempio REST:

crea cliente:

```
PUT clients/1234 HTTP/1.1
Host: http://www.sito.com:80
Content-Type: text/xml; charset=utf-8
Content-length: 474
<client xmlns:m="http://www.myApp.com/">
  <nome>Rossi S.p.A.</nome>
  <tel>051 654321</tel>
  <citta>Bologna</citta>
</client>
```

aggiorna cliente:

```
PUT clients/1234 HTTP/1.1
Host: http://www.sito.com:80
Content-Type:application/json; charset=utf-8
Content-length: 474
{
  "nome": "Rossi S.p.A.",
  "tel" : "051 654321",
  "citta" : "Bologna"
}
```

INDIVIDUI E COLLEZIONI:

REST identifica due concetti fondamentali: **individui e collezioni**.

Esempio:

- un cliente vs. l'insieme di tutti i clienti;
- un esame vs. l'insieme di tutti gli esami superati;

Fornisce un URI ad entrambi ed ogni operazione (di tipo CRUD) avviene su uno e uno solo di questi concetti.

A seconda della combinazione di verbi e risorse si ottiene l'insieme delle operazioni possibili. Ciò che passa come corpo di una richiesta o di una risposta non è la risorsa, ma una sua rappresentazione, di cui gli attori si servono per portare a termine l'operazione.

Le collezioni possono contenere individui o altre collezioni ed è consigliabile strutturare gli URI in modo gerarchico per esplicitare queste relazioni.

In questo modo si ottiene un'API più leggibile e routing semplificato in molti framework di sviluppo:

Esempio:

- Tutti i clienti
/clients/
- Il cliente 1234
/clients/1234/
- Tutti gli ordini del cliente 1234
/clients/1234/orders/

Le collezioni sono intrinsecamente plurali, mentre gli individui sono intrinsecamente singolari, per questo motivo (e siccome è una buona pratica rendere le collezioni

visivamente distinguibili dagli individui) per i nomi delle collezioni si utilizza un termine plurale e uno slash in fondo all'URI, mentre per i nomi degli individui si utilizza un termine singolare.

Esempio:

URI	Rappresentazione
/customers/	collezione dei clienti
/customers/abc123	cliente con id=abc123
/customers/abc123/	collezione delle sotto-risorse del cliente con id=abc123 (es. indirizzi, telefoni,ecc.)
/customers/abc123/addresses/1	primo indirizzo del cliente con id=abc123

FILTRI E SEARCH NEGLI URI REST:

Un filtro genera un **sottoinsieme** specificato attraverso una regola di qualche tipo.

La gerarchia permette di specificare i tipi più frequenti e rilevanti di filtro.

Altrimenti si può usare la parte query dell'URI di una collezione.

Esempio di entrambe le tecniche (filtrati e ricerca con query):

Filtrati: prime due righe

Ricerca: ultime tre righe

URI	Rappresentazione
/regions/ER/customers/	collezione dei clienti dell'Emilia Romagna
/status/active/customers/	collezione dei clienti attivi
/customers/?tel=0511234567 oppure /customers/? search=tel&value=0511234567	collezione dei clienti che hanno telefono = 051 1234567
/customers/? search=sales&value=100000&op=gt	collezione dei clienti che hanno comprato più di 100.000€

USO DEI VERBI HTTP IN REST:

- **GET**: Elencare tutti i clienti
GET /customers/
- **GET**: Accedere ai dati del cliente id=abc123
GET /customers/abc123
- **POST**: Creare un nuovo cliente (il client non decide l'identificatore)
POST /customers/
- **PUT**: Creare un nuovo cliente (il client decide l'identificatore)
PUT /customers/abc123
- **PUT**: Modificare (tutti) i dati del cliente id=abc123
PUT /customers/abc123
- **POST**: Modificare alcuni dati del cliente id=abc123
POST /customers/abc123/telephones/
- **DELETE**: Cancellare il cliente id=abc123
DELETE /customers/abc123

Attenzione:

semantica di POST:

Nelle vecchie versioni di HTTP (ad es. RFC2616) si diceva:

"The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. It essentially means that POST request-URI should be of a collection URI."

Nel 2014 è stata approvata una modifica e chiarificazione ad alcuni testi del documento di HTTP (RFC7231), in particolare:

"The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics."

In pratica quindi, il POST può essere usato in una moltitudine di situazioni secondo una **semantica decisa localmente**, purché non sovrapposta a quella degli altri verbi.

CONTROLLO DELLE VERSIONI DI UN'API:

È naturale che le API siano modificate durante il loro ciclo di vita, di conseguenza è fondamentale dividerle in versioni.

REST non fornisce linee guida stringenti ma esistono due approcci principalmente adottati:

- Il più usato consiste nel memorizzare il numero di versione dell'API all'inizio degli URI; questa tecnica però viola l'idea di identificare solo una risorsa nell'URI;

Esempio:

`http://api.miosito.com/v1/clients/`
`http://api.miosito.com/v1/clients/1234`

- Alternativamente si possono usare gli header HTTP Accept ed i meccanismi di content negotiation per specificare la versione dell'API supportata;

Esempio:

`Accept: application/vnd.example.v1+json`
`Accept: application/vnd.example+json;version=1.0`

In questo caso gli URI sono più puliti ma la complessità lato client aumenta.

MODELLI DI MATURITÀ DELLE API:

È molto comune trovare API che dichiarano di essere RESTful ma che in realtà non rispettano completamente le linee guida del paradigma.

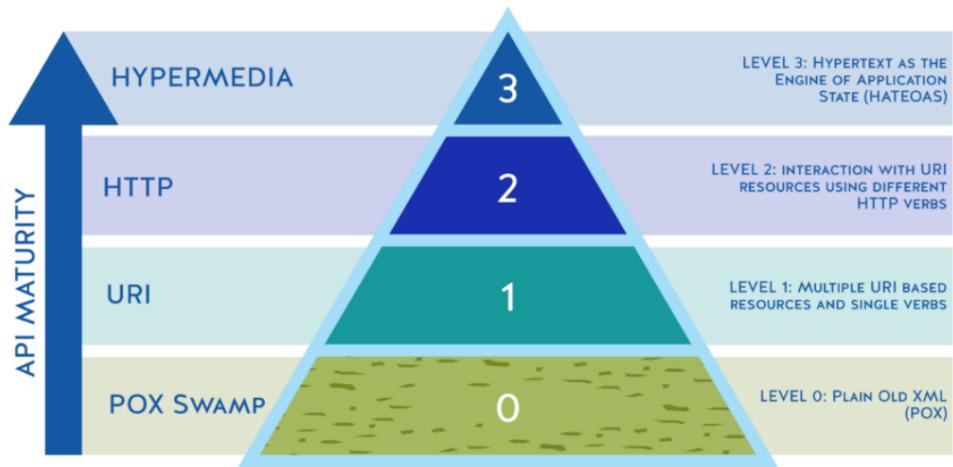
Per fare chiarezza diversi studiosi hanno proposto modelli per verificare "quanto un'API è davvero REST" e misurarne il livello di maturità.

Tra i più rilevanti c'è il modello di Richardson, presentato per la prima volta alla conferenza QCon (Enterprise Software Development Conference) nel 2008

(<https://www.crummy.com/writing/speaking/2008-QCon/act3.html>, un post di Martin Fowler che descrive il modello di Richardson

<https://martinfowler.com/articles/richardsonMaturityModel.html>)

Leonard Richardson individua 4 livelli:



ALTRI CONSIGLI E LINEE GUIDA:

- Adottare una convenzione di denominazione coerente e chiara negli URI;
- Usare gerarchie ma valutare i livelli necessari (chiarezza vs. carico sul server);
- Evitare di creare API che rispecchiano semplicemente la struttura interna di un database;
- Fornire meccanismi (generalmente i parametri nelle query) per filtrare e paginare le risposte;
- Supportare richieste asincrone e in questo caso restituire codice HTTP 202 (Accettato ma non completato) e informazioni (URL) per accedere allo stato della risorsa.

OpenAPI

Una API è RESTful se utilizza i principi REST nel fornire accesso ai servizi che offre.

Per documentare un'API è necessario definire:

- end-point (URI / route) che supporta, ovvero gli URI a cui manda le richieste e da cui riceve le risposte, separando collezioni e elementi singoli;
- metodi HTTP di accesso, ovvero cosa succede con un GET, un PUT, un POST, un DELETE, ecc;
- rappresentazioni in Input e Output, di solito per fare ciò non si usa un linguaggio di schema, ma un esempio sufficientemente complesso;
- condizioni di errore e i messaggi che restituisce in questi casi.

Swagger ed Open API:

Swagger è un ecosistema di strumenti per creazione, costruzione, documentazione ed accesso ad API utilizzato soprattutto in ambito REST.

In particolare con esso sono stati creati un linguaggio per la documentazione di API REST (ormai standard) e strumenti per la loro modifica, documentazione e test.

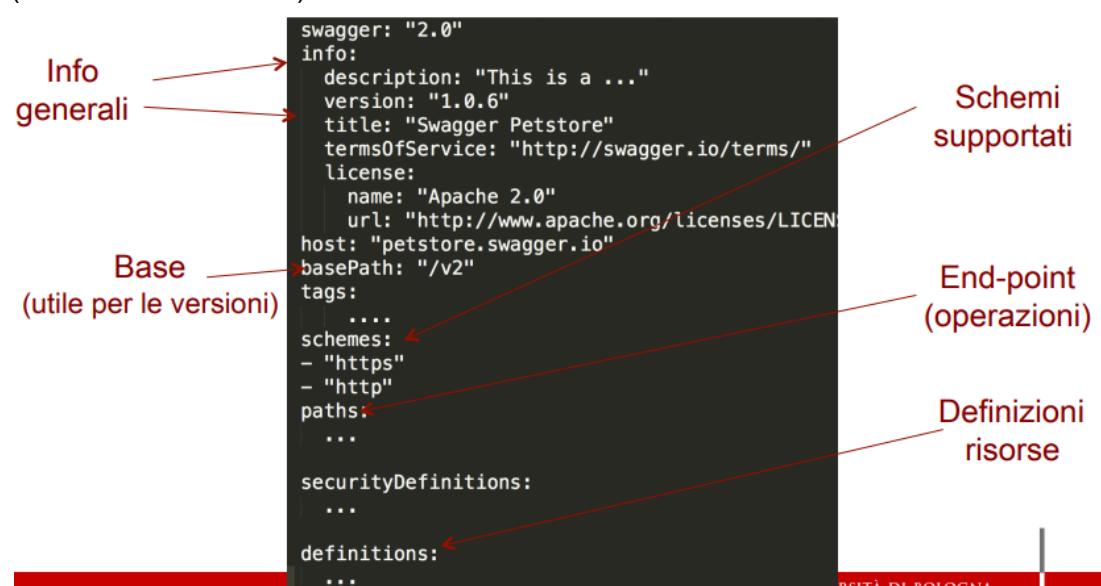
Nel 2016 il linguaggio è stato reso di pubblico dominio ed ha preso il nome di **Open API**.

Open API può essere espresso sia in JSON che in YAML (trattati in seguito) e permette la generazione automatica di documentazione, modelli e codice.

In questi appunti si tratta la versione "2.0" dell'API.

STRUTTURA BASE IN OPENAPI:

(versione 2.0 in YAML)



SEZIONE PATHS:

La sezione **paths** di un'API descrive i percorsi (URL) corrispondenti alle operazioni possibili sull'API secondo la struttura: <host>/<basePath>/<path>.

Per ogni percorso (path o endpoint) si definiscono tutte le possibili operazioni che, secondo i principi REST, sono identificate dal metodo HTTP corrispondente.

Per ogni path quindi ci sono tante sottosezioni quante sono le operazioni e per ognuna di esse si hanno:

- **Informazioni generali;**
- **Parametri** di input e di output (trattati nei paragrafi successivi).

Struttura di un path:



PARAMETRI IN INPUT:

I parametri in input sono descritti nella sezione **parameters** che definisce una lista di parametri (il simbolo "-" individua un elemento di una lista), i quali hanno al loro interno:

- **tipo** del parametro: keyword in che può assumere valori path, query o body;
- **nome** (keyword name) e descrizione (description);
- se è opzionale o obbligatorio (**required**);
- **formato** dei valori che il dato può assumere: keyword type o schema, più altre proprietà dipendenti dal tipo di dato.

PARAMETRI DI OUTPUT:

I possibili output (dati, codici HTTP e messaggi di errore) sono definiti attraverso la **keyword responses**, ovvero si specifica il tipo di output atteso nel body delle risposte.

Ogni risposta ha inoltre un **id numerico** univoco associato al codice HTTP corrispondente:

- 200 viene usato per indicare che non c'è stato alcun errore;
- da 400 in poi vengono in genere usati per indicare messaggi di errore.

TIPI DI DATO:

I dati in input ed output di un path possono essere di vario tipo:

- **Primitivi**: interi, stringhe, date, booleani;

In questo caso nella sezione formato si inserisce il valore **type** ed i suoi dettagli.

Nel caso in cui il parametro indichi un input e sia una stringa, si può utilizzare al posto di type la keyword **enum** seguita da una lista per indicare un insieme di possibili valori.

- Oggetti (dichiarabili soltanto nel body);

In questo caso invece si usa la keyword **schema** seguita dalla definizione delle proprietà o, molto più di frequente, dal riferimento alla definizione tramite **\$ref**.

- Array di oggetti o dati primitivi;

Quando si dichiara il tipo di dato si usa **type:array** seguito da **items** per indicare il tipo degli elementi nel vettore.

Solitamente si dichiarano nel body ma si può fare anche nella parte query.

Esempi di parametri path e query:

```

/pet/{petId}:
  get:
    summary: Find pet by ID
    description: Returns a single pet
    operationId: getPetById
    parameters:
      - name: petId
        in: path
        description: ID of pet to return
        required: true
        type: integer
        format: int64

/pet/:
  get:
    summary: Finds Pets by status
    operationId: findPetsByStatus
    parameters:
      - name: status
        in: query
        description: Status values that need to be considered for filter
        required: true
        type: array
        items:
          type: string

```

Parametro <petId> nell'URI

Parametro <status> nella parte query dell'URI /pet/?status=ready

Array di stringhe

Esempio di parametri in input
(notare i parametri nel body)

```

/user/{username}:
  put:
    tags:
      - user
    summary: Updated user
    description: This can only be done by the logged in user.
    operationId: updateUser
    parameters:
      - name: username
        in: path
        description: name that need to be updated
        required: true
        type: string
      - in: body
        name: body
        description: Updated user object
        required: true
        schema:
          $ref: '#/definitions/User'

```

Parametro nel path

Oggetto <User> nel body

OGGETTI E DEFINIZIONI:

Nell'esempio precedente il body contiene un oggetto di tipo User, viene infatti passata un'intera risorsa (più propriamente la sua rappresentazione) come parametro.

La sezione **definitions** permette di definire i tipi degli oggetti, le loro proprietà ed i valori che possono assumere.

Questi tipi possono essere referenziati (tramite schema e poi \$ref, come nell'esempio) sia nelle richieste che nelle risposte.

Esempi di modelli e tipi di dato:

```
User:  
  type: object  
  properties:  
    id:  
      type: integer  
      format: int64  
    username:  
      type: string  
    firstName:  
      type: string  
    lastName:  
      type: string  
    email:  
      type: string  
    password:  
      type: string  
    phone:  
      type: string  
    userStatus:  
      type: integer  
      format: int32  
    description: User Status  
  
Order:  
  type: object  
  properties:  
    id:  
      type: integer  
      format: int64  
    petId:  
      type: integer  
      format: int64  
    quantity:  
      type: integer  
      format: int32  
    shipDate:  
      type: string  
      format: date-time  
    status:  
      type: string  
      description: Order Status  
    enum:  
      - placed  
      - approved  
      - delivered  
    complete:  
      type: boolean
```

Esempio di risposta:

```
/pet/:  
  get:  
    summary: Finds Pets by status  
    operationId: findPetsByStatus  
    parameters:  
      - name: status  
        in: query  
        description: Status values that need to be applied to the operation  
        required: false  
        type: array  
        items:  
          type: string  
    responses:  
      '200':  
        description: successful operation  
        schema:  
          type: array  
          items:  
            $ref: '#/definitions/Pet'  
      '400':  
        description: Invalid status value
```

Tipo della risposta.
Vettore di
oggetti <Pet>

Codici
HTTP

SWAGGER EDITOR:

Al seguente indirizzo si trova un editor online per fare e testare modelli in OpenAPI:
<https://editor.swagger.io/>

Markup

Si definisce markup ogni mezzo per rendere esplicita una particolare interpretazione di un testo.

Oltre a rendere il testo più leggibile, il markup permette anche di specificare ulteriori usi del testo.

Con il markup per sistemi informatici, si specificano le modalità esatte di utilizzo del testo nel sistema stesso.

Esempio:

In italiano la punteggiatura e lo spazio sono simboli di markup, non hanno significato di per sé ma servono a rendere comprensibili il testo al lettore.

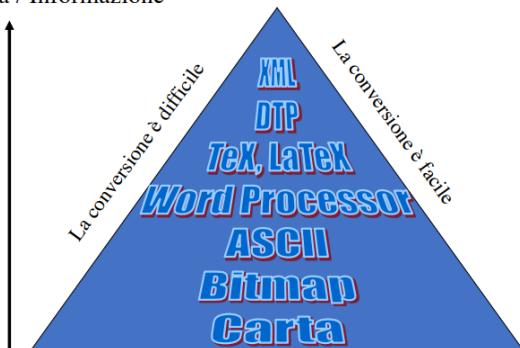
Quando un autore scrive specifica anche i delimitatori di parola (chiamati spazi), i delimitatori di frase (chiamati virgolette) e i delimitatori di periodo (chiamati punti).

La numerazione delle pagine o l'uso dei margini per creare effetti sul contenuto sono noti da centinaia di anni.

Eppure questo a stretto rigore non fa parte del testo, ma del markup: nessuno dirà ad alta voce ‘virgola’ o ‘punto’ nel leggere un testo, ma creerà adeguati comportamenti paralinguistici (espressioni, toni, pause) per migliorare in chi ascolta la comprensione del testo.

La piramide dei linguaggi di markup

Energia / Informazione



Man mano che si sale nella piramide il linguaggio ed il formato in cui è espressa la risorsa diventano più difficili, ma più ricchi di informazioni esplicite (a livello informatico).

Per esempio sulla carta noi possiamo leggere cosa c'è scritto, ma un computer dovrebbe fare un'operazione non banale di riconoscimento del testo da una foto, poi dovrebbe cercare di interpretare la formattazione, ecc...; in ascii invece il computer ha già il testo esplicito, in xml anche la formattazione, ecc...

MODI DEL MARKUP:

Proprietario o pubblico

Un formato proprietario è creato da una specifica azienda con uno specifico scopo commerciale.

L'azienda ne detiene i diritti e dunque è in grado di modificarlo, aggiornarlo o rivoluzionarlo in qualunque momento e per qualunque motivo.

Invece un formato pubblico è stato creato da un gruppo di interesse (individui, aziende, enti non commerciali, ecc.) come modello di armonizzazione tra esigenze di ciascun partecipante.

Il gruppo tipicamente pubblica le specifiche del formato, permettendo a chiunque di realizzare strumenti software per quel formato.

A volte questo si concretizza in uno standard ufficiale avente valore normativo.

Binario o leggibile

Un formato binario è la memorizzazione esatta delle strutture in memoria dell'applicazione, che non hanno niente a che vedere con le esigenze di comprensione di esseri umani.

Esempio:

Nei formati word, pdf, photoshop, ecc... non si riesce a leggere come testo il file di salvataggio, ma è necessario aprirlo con l'applicazione con cui è stato creato.

Un formato leggibile invece è fatto per essere, in casi speciali, letto anche da esseri umani che possono intervenire per fare modifiche.

In questo caso l'applicazione deve trasformare quanto legge in una struttura interna utile per le operazioni di modifica o presentazione, questa fase si chiama parsing.

Interno o esterno

Il markup interno inserisce istruzioni di presentazione all'interno del testo, in mezzo alle parole.

Il markup esterno invece prevede due blocchi di informazioni: il contenuto ed il markup separati e collegati tra di loro da indirezione.

Il markup interno richiede sintassi particolari per distinguere il markup dal contenuto; tipicamente si adottano segnalatori particolari che cambiano il tipo di interpretazione del documento.

La presenza di caratteri segnalatori nel testo richiede l'adozione di tecniche di escaping.

Il markup esterno richiede un meccanismo di indirezione, basato su indirizzi, offset o identificatori per associare con correttezza il markup al contenuto.

In informatica l'indirezione (detta anche riferimento indiretto) è la tecnica che consente di indicare un oggetto o un valore mediante un suo riferimento invece che direttamente. Il termine viene utilizzato frequentemente nei linguaggi di programmazione per indicare l'uso dei puntatori.

Da <https://it.wikipedia.org/wiki/Indirezione>

Procedurale o descrittivo

Il markup assolve a diversi ruoli a seconda del sistema di elaborazione, dell'applicazione, dello scopo a cui il documento è soggetto (i più importanti sono procedurale e descrittivo):

- Puntuazionale:

Il markup puntuazionale consiste nell'usare un insieme prefissato di segni per fornire informazioni perlopiù sintattiche sul testo.

Tali regole di punteggiatura sono sostanzialmente stabili, note agli autori e frequenti nei documenti.

Per questo motivo gli autori tipicamente creano e forniscono il loro markup puntuazionale autonomamente.

Esistono tuttavia notevoli problemi nell'uso della punteggiatura:

- Incertezze strutturali (virgola, punto e virgola o punto?);
- Incertezze grafiche (virgolette aperte e chiuse o neutre?);
- ambiguità procedurali (il punto viene usato sia per segnare la fine di una frase, che l'esistenza di un'abbreviazione, oppure i tre puntini di sospensione).

- Presentazionale:

Il markup presentazionale consiste nell'indicare effetti (grafici o altro) per rendere più chiara la presentazione del contenuto.

Nel testo, possono essere cambi di paragrafo o di pagina, interlinea, pallini per liste, ecc...

È markup presentazionale: cambiare pagina all'inizio di una nuova sezione, scrivere "Capitolo 3" in cima alla pagina, ecc...

- **Procedurale:**

Il markup procedurale consiste nell'indicare con precisione ad un sistema automatico quale effetto attivare e che procedura (serie di istruzioni) eseguire nella visualizzazione del contenuto.

Quindi si utilizzano le capacità del sistema di presentazione per avere con precisione l'effetto voluto.

Ciò però è difficilmente interpretabile dagli umani e può essere interpretato solo da macchine che hanno lo stesso set di funzioni della macchina su cui è stato scritto.

Esempio: Wordstar Dot Commands

.PL 66
.MT 6
.MB 9
.LH 12

- **Descrittivo:**

Il markup descrittivo consiste nell'identificare strutturalmente il tipo di ogni elemento del contenuto.

Invece di specificare effetti grafici come l'allineamento o l'interlinea, si individua il ruolo di un elemento all'interno del documento, per esempio specificando che un elemento è un titolo, un paragrafo, o una citazione.

Questo tipo di markup astrae dalla tipografia, ma indica assegnazioni permanenti alla semantica del testo (un titolo sarà sempre un titolo, indipendentemente dal font che si sceglie).

- **Referenziale:**

Nel markup referenziale si indicano le cose con delle sigle.

Questo markup consiste nel fare riferimento ad entità esterne al documento per fornire significato o effetto grafico ad elementi del documento.

Per esempio utilizzare una sigla nota che venga poi sostituita dalla parola intera durante la stampa (l'autore scrive "CdL" e il sistema trasforma automaticamente l'input in "Corso di Laurea").

- **Metamarkup:**

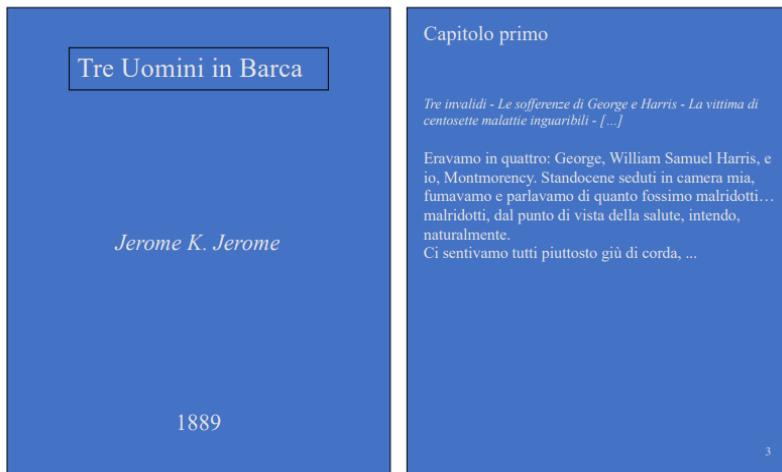
Il metamarkup consiste nel fornire regole di interpretazione del markup e permette di estendere o controllare il significato di tale markup.

Ad esempio la possibilità di definire macro per l'interpretazione e la visualizzazione del documento, o il processo di definizione degli elementi e delle procedure valide di un documento.

Il metamarkup permette di definire delle regole con cui creare dei linguaggi di markup.

Esempio:

Un testo su carta



Un testo senza markup

Questo è il testo completamente senza markup, come poteva essere scritto su un papiro della biblioteca di Alessandria, nel II o III secolo a.C.

treuominiinbarcajeromekjerome1889capitolop
rimotreinvalidilesofferenzedigeorgeharris
lavittimadicentosettemalattieinguaribilier
avamoinquattrogeorgewilliamsamuelharriseio
montmorencystandocenesedutiincameramafuma
vamoeparlavamodiquantofossimomalridottimal
ridottidalpuntodividistadellasaluteintendona
turalmentecisentivamotuttipiuttostogìùdico
rda

Markup metabolizzato

Aggiungiamo markup puntuazionale e presentazionale:
maiuscole/minuscole, punteggiatura, spazi e ritorni a capo
sono essi stessi elementi di markup.

Tre Uomini in Barca
Jerome K. Jerome (1889)
Capitolo primo
Tre invalidi - Le sofferenze di George e Harris - La
vittima di centosette malattie inguaribili - [...] Eravamo in quattro: George, William Samuel Harris, e io, Montmorency. Standoce seduti in camera mia, fumavamo e parlavamo di quanto fossimo malridotti, dal punto di vista della salute, intendo, naturalmente. Ci sentivamo tutti piuttosto giù di corda, ...

Markup procedurale

Sono comandi, o istruzioni che il sistema di lettura (umano o elettronico) deve eseguire sul testo. Ad esempio, istruzioni su come andare a capo, come decidere i margini, ecc. Questo è RTF.

```
{\rtf1 \mac \ansicpg10000 \uc1 \pard \plain \s15 \qc \widctlpar \adjustright \f4 \fs48 \cgrid {Tre uomini in Barca \par } \pard \plain \widctlpar \adjustright \f4 \cgrid { \line \par \par \par \par \par } \pard \plain \s1 \qc \keepn \widctlpar \outlinelevel0 \adjustright \i \f4 \fs36 \cgrid {Jerome K. Jerome \par } \pard \plain \qc \widctlpar \adjustright \f4 \cgrid { \fs36 [...] \par 1889 \line \par } \pard \widctlpar \adjustright { \page } { \b \fs36 Capitolo primo } { \par \par \line } { \i Tre invalidi - Le sofferenze di George e Harris - La vittima di centosette malattie inguaribili - [ \u8230 \c9] \par } { \line } { \fs28 Eravamo in quattro: George, William Samuel Harris, e io, Montmorency. Standoce seduti in camera mia, fumavamo e parlavamo di quanto fossimo malridotti \u8230 \c9 malridotti, dal punto di vista della salute, intendo, naturalmente. \line Ci sentivamo tutti piuttosto gi \u249 \9d di corda, ... \par }
```

Markup descrittivo

Sono informazioni (descrizioni) sugli elementi del documento, che ne specificano il ruolo, la giustificazione, la relazione con gli altri elementi.

```
<ROMANZO>
  <TITOLO>Tre uomini in Barca</TITOLO>
  <AUTORE>Jerome K. Jerome</AUTORE>
  <ANNO>1889</ANNO>
  <CAPITOLO>
    <TITOLO>Capitolo primo</TITOLO>
    <INDICE>
      <EL>Tre invalidi</EL>
      <EL>Le sofferenze di George e Harris </EL>
      <EL> La vittima di centosette malattie inguaribili </EL>
    </INDICE>
    <PARA>Eravamo in quattro: George, William Samuel Harris, e io, Montmorency. Standoce seduti in camera mia, fumavamo e parlavamo di quanto fossimo malridotti... malridotti, dal punto di vista della salute, intendo, naturalmente. </PARA>
    <PARA>Ci sentivamo tutti piuttosto giù di corda, ...</PARA>
  </CAPITOLO>...
</ROMANZO>
```

I linguaggi di markup

TROFF/NROFF:

Nato nel 1973 fa parte della distribuzione Unix standard (sotto Linux si chiama Groff).

È ancora usato per documentazione tecnica, in particolare i manuali on-line di Unix.

È un linguaggio procedurale.

Esiste un formatter che è in grado di creare documenti stampabili sia su stampante (troff) che su schermo a carattere (nroff).

I comandi sono o esterni (compaiono su righe autonome precedute da un punto) o interni (introdotti dal carattere di escape "\").

Permette la definizione e l'uso di quattro tipi di carattere: roman, italic, bold e symbol.

Permette la creazione di macro complesse (dove però il nome è al massimo di due caratteri).

Esempio:

```
.\" Questo è un esempio Troff.  
.\" margine sinistro 4 cm.  
.\" ampiezza del testo 8 cm.  
.po 4c  
.ll 8c  
.\" Inizia il documento.  
.ft B  
1. Introduzione a Troff
```

```
.ft P
```

Questo \(`e un esempio di documento scritto in modo tale da poter essere elaborato con Troff.

In questo caso, si presume che verr\(`a utilizzato lo stile ``\fBs\fP'' (con l'opzione \fB\-\ms\fP).

```
.ft B
```

1.1 Paragrafi

```
.ft P
```

Il testo di un paragrafo termina quando nel sorgente viene incontrata una riga vuota.

Per la precisione, gli spazi verticali vengono rispettati, per cui le righe vuote si traducono in spazi tra i paragrafi, anche quando queste sono pi\(`u di una.

1. Introduzione a Troff

Questo è un esempio di documento scritto in modo tale da poter essere elaborato con Troff. In questo caso, si presume che verrà utilizzata lo stile "s" (con l'opzione -ms).

1.1 Paragrafi

Il testo di un paragrafo termina quando nel sorgente viene incontrata una riga vuota.

Per la precisione, gli spazi verticali vengono rispettati, per cui le righe vuote si traducono in spazi tra i paragrafi, anche quando queste sono più di una.

Questo è l'inizio di un nuovo paragrafo dopo tre righe vuote di separazione.

TEX E LATEX:

Il vero nome di questi due linguaggi è $T_E X$ e $L^A T_E X$.

$T_E X$ è stato realizzato agli inizi degli anni 70 da Donald Knuth.

È un linguaggio di programmazione completo (è turing completo!), dotato di comandi per la formattazione di testi (circa 300 comandi fondamentali, detti primitive).

Essendo di notevole complessità, è rivolto unicamente a programmatore e tipografi molto sofisticati.

Metafont è un sistema associato a $T_E X$ per la descrizione delle forme dei caratteri di un font attraverso formule matematiche.

$T_E X$ permette la generazione di macro che semplificano notevolmente la generazione di testi particolarmente sofisticati.

Esistono librerie di macro che permettono di scrivere documenti arbitrariamente complessi: matematica, chimica, musica, grafica vettoriale, grafica bitmap, ecc.

Siccome $T_E X$ è un linguaggio parecchio complicato, nel 1985 Leslie Lamport sviluppò LaTeX, una raccolta di macro $T_E X$ per la generazione di una ventina circa di tipi di documento particolarmente comuni: articolo, libro, lettera, annuncio, ecc.

Esempio:

```
documentclass{article}
% Inizia il preambolo.
\setlength{\textwidth}{7cm}
\setlength{\textheight}{7cm}
% Fine del preambolo.
\begin{document}
% Inizia il documento vero e proprio.
\section{Introduzione a TeX/LaTeX}
Questo \`e un esempio di documento scritto con LaTeX.
Come si pu\`o vedere \`e gi\`a stato definito uno stile generale del
documento: article.
\subsection{Gli ambienti}
LaTeX utilizza gli ambienti per definire dei comportamenti circoscritti a
zone particolari del testo.
Per esempio, la centratura si ottiene utilizzando l'ambiente center.
\begin{center}
Questo \`e un esempio di testo centrato.
\end{center}
% Fine del documento.
```

MARKDOWN E SINTASSI WIKI:

Formati testuali che usano trucchi testuali ad hoc per ottenere strutture e effetti tipografici precisi.

Utilizzati anche prima dei computer con gli appunti sulle macchine da scrivere.

Sono ottimi per testi lunghi, ma non permettono di catturare effetti tipografici specifici e forniscono supporto solo per le caratteristiche tipografiche più semplici.

Però il markdown è troppo semplice per quelle che sono comunemente le necessità tipografiche.

Esempio:

```
An h1 header  
=====  
Paragraphs are separated by a blank line.  
2nd paragraph. *Italic*, **bold**, and `monospace`.  
Itemized lists look like:  
* this one  
* that one  
* the other one
```

JSON:

JSON (JavaScript Object Notation) è un formato dati derivato dalla notazione usata da JavaScript per gli oggetti.

È un modo di trasmettere dati su testo e convertirlo facilmente in una struttura dati e viceversa.

È ottimo per i dati ma non consente di introdurre notazioni topografiche come grassetto, corsivo, ecc...

Esempio:

```
{  
    "nome": ["Giuseppe", "Andrea", "Federico"],  
    "cognome": "Rossi",  
    "altezza": 180,  
    "nascita": "1995-04-11T22:00:00.000Z",  
    "indirizzo": {  
        "via": {  
            "strada": "Via Indipendenza",  
            "numero": "15"  
        },  
        "citta": "Bologna",  
        "nazione": "Italia"  
    },  
    "telefono": [  
        { "tipo": "casa", "numero": "051 123456"},  
        { "tipo": "cell", "numero": "335 987654"}  
    ]  
}
```

YAML:

YAML (**YAML Ain't Markup Language** (si, è ricorsivo), significa “YAML non è un linguaggio di markup”) è un'altra linearizzazione di strutture dati simile a JSON, ma con sintassi ispirata a Python:

- Superset di JSON (ogni file JSON è anche un file YAML);
- Indentazione come modello di annidamento (come python);
- Supporto di tipi scalari (stringhe, interi, float), liste (array) e hash (array associativi, mappe);
- Supporto di stringhe multiriga con due delimitatori diversi;
- Supporto di commenti.

Esempio:

```
# con il carattere -  
# si indicano delle liste  
nome:  
- Giuseppe  
- Andrea  
- Federico  
cognome: Rossi  
  
# oggetti annidati vogliono spazi  
# in YAML non si usano tab  
indirizzo:  
via:  
    strada: "Via Indipendenza"  
    numero: 15  
    citta: "Bologna"  
    nazione: "Italia"  
telefono:  
- tipo: "casa"  
  numero: "051 123456"  
- tipo: "cell"  
  numero: "335 987654"  
  
# si supportano anche interi,  
# float e date  
altezza: 180  
nascita: 1995-04-11T22:00:00.000Z  
  
# con il carattere |  
# i newline sono significativi  
immagine: !!binary |  
R0lGODdhDQAIAlAAAAAAANn  
Z2SwAAAAADQAIACF4SDGQ  
ar3xxbj9p0qa7R0YxwzaFME  
IAADS=  
  
# con il carattere <  
# i newline sono irrilevanti  
motto: <  
  <blockquote>  
    <p>Live and let live</p>  
  </blockquote>
```

SGML:

SGML (Standard Generalized Markup Language) è un meta-linguaggio non proprietario di markup descrittivo ed è uno standard.

Facilita markup leggibili, generici, strutturali e gerarchici.

SGML è uno standard ISO (International Standard Organization) n. 8879 del 1986 (ISO è la più importante organizzazione mondiale degli standard).

Il fatto che è uno **standard** significa che è il risultato di discussioni e compromessi di rappresentanti di tutte le comunità interessate e che non dipende dai piani commerciali di una singola casa produttrice.

SGML non è un linguaggio di markup, ma un linguaggio con cui si definiscono linguaggi di markup, ovvero un **meta-linguaggio di markup**.

SGML non dà dunque tutte le risposte di markup che possano sorgere a chi vuole arricchire testi per qualunque motivo, ma fornisce una sintassi per definire il linguaggio adatto.

SGML non sa cos'è un paragrafo, una lista, un titolo, ma fornisce una grammatica che permette di definirli.

Essendo **non proprietario** non esiste un'unica ditta o casa produttrice che ne detiene il controllo; viceversa RTF è della Microsoft, mentre PostScript e Acrobat sono di Adobe.

Inoltre essendo uno standard non proprietario, non dipende da un singolo programma, da una singola architettura, quindi gli stessi dati possono essere portati da un programma all'altro e da una piattaforma all'altra senza perdita di informazioni; anche l'evoluzione nel tempo è assicurata per lo stesso motivo.

In SGML il markup è posto in maniera **leggibile** a fianco degli elementi del testo a cui si riferiscono.

Essi possono sia essere usati da un programma, sia letti e modificati da un essere umano.

Il markup generato è **descrittivo**, ovvero non è pensato unicamente per la stampa su carta: è possibile combinare markup utile per scopi o applicazioni diverse, ed in ogni contesto considerare o ignorare di volta in volta i markup non rilevanti.

SGML inoltre permette di definire delle **strutture**, suggerite o imposte, a cui i documenti si devono adeguare.

Ad esempio, si può imporre che un testo sia diviso in capitoli, ognuno dei quali dotato di un titolo, una breve descrizione iniziale e almeno un paragrafo di contenuto.

È cioè possibile definire una serie di regole affinché il testo sia considerato strutturalmente corretto.

Le strutture imposte da SGML sono tipicamente **gerarchiche**, ovvero a livelli di dettaglio successivi.

Ad esempio, si può imporre che il libro sia fatto di capitoli, e che questi a loro volta siano fatti di un'una descrizione e molti paragrafi. Una descrizione sarà quindi fatta di elementi, ciascuno dei quali sarà una frase composta di testo; i paragrafi saranno testo eventualmente contenenti anche elementi in evidenza o figure.



I documenti SGML:

Un documento in un linguaggio di markup definito sulla base di SGML è sempre composto delle seguenti tre parti:

- **Dichiarazione SGML:**

La dichiarazione SGML contiene le istruzioni di partenza delle applicazioni SGML.

Essa permette di specificare valori fondamentali come la lunghezza dei nomi degli elementi, il set di caratteri usati, le specifiche caratteristiche di minimizzazione ammesse, ecc.).

Una dichiarazione SGML può essere lunga varie centinaia di righe, ma non è obbligatoria, infatti se è assente viene usata una dichiarazione di default detta RCS (Reference Concrete Syntax).

La RCS definisce lunghezze e sintassi standard (come l'uso del carattere "<" per indicare l'inizio del tag).

Esempio:

```
<!SGML "ISO 8879:1986" car. spec.>
```

- **Dichiarazione di documento (DOCTYPE):**

La dichiarazione del tipo del documento (detta anche DTD) serve a specificare le regole che permettono di verificare la correttezza strutturale di un documento.

Vengono cioè elencati i file che contengono gli elementi ammissibili, il contesto in cui possono apparire, ed altri eventuali vincoli strutturali.

Nella terminologia SGML, si parla di modellare una classe (cioè una collezione omogenea) di documenti attribuendogli un tipo.

- **Istanza del documento:**

L'istanza del documento è quella parte del documento che contiene il testo vero e proprio dotato del markup appropriato.

Essa contiene una collezione di elementi (tag), attributi, entità, PCDATA, commenti, ecc.

Le applicazioni SGML sono in grado di verificare se l'istanza del documento segue le regole specificate nella dichiarazione del DOCTYPE e di identificare le violazioni.

Nell'istanza di un documento con markup di derivazione SGML (inclusi HTML, XML, ecc.) contiene una varietà dei seguenti componenti:

- **Elementi:**

Sono essenzialmente contenitori di altre cose.

Gli elementi sono le parti di documento dotate di un senso proprio.

Il titolo, l'autore, i paragrafi del documento sono tutti elementi.

Un elemento è individuato da un tag iniziale, un contenuto ed un tag finale.

Non bisogna confondere i tag con gli elementi!

Esempio:

```
<titolo>Tre uomini in barca</titolo>
```

- **Attributi:**

Gli attributi sono informazioni aggiuntive sull'elemento che non fanno effettivamente parte del contenuto (meta-information).

Sono piccoli frammenti di dato che si attribuiscono ad un elemento.

Essi sono posti dentro al tag iniziale dell'elemento e tipicamente hanno la forma nome="valore".

Esempio:

```
<racconto tipo="romanzo">...</racconto>
```

```
<capitolo id="c1">Capitolo primo</capitolo>
```

- **Entità:**

Frammento di markup che sostituisce un frammento di contenuto, da qualche parte c'è una definizione che indica quale contenuto sostituire all'entità.

Le entità sono quindi frammenti di documento memorizzati separatamente e richiamabili all'interno del documento.

Esse permettono di riutilizzare lo stesso frammento in molte posizioni garantendo sempre l'esatta corrispondenza dei dati, e permettendo una loro modifica semplificata.

Esempio:

Oggi è una bella giornata.

Come dice &FV;: "divertitevi!"

Quanta felicitÀ

- **Testo (detto anche #PCDATA):**

Rappresenta il contenuto vero e proprio del documento.

Esso corrisponde alle parole, gli spazi e la punteggiatura che costituiscono il testo.

Viene anche detto #PCDATA (Parsed Character DATA) in quanto i linguaggi di markup definiscono character data (CDATA) il contenuto testuale vero e proprio e quello degli elementi è soggetto ad azione di parsing (perlopiù per identificare e sostituire le entità).

- **Commenti:**

I documenti di markup possono contenere commenti, ovvero note dell'autore, dell'editore, ecc.

Queste note non fanno parte del contenuto del documento e le applicazioni di markup li ignorano.

Sono molto comodi per passare informazioni tra un autore e l'altro, o per trattenere informazioni per se stessi nel caso si rischi di dimenticarle.

Esempio:

<!-- Questo testo è ignorato dal parser -->

- **Processing Instructions:**

Le processing instructions (PI) sono elementi particolari (spesso di senso esplicitamente procedurale) posti dall'autore o dall'applicazione per dare ulteriori indicazioni su come gestire il documento SGML in un caso specifico.

Per esempio, in generale è l'applicazione a decidere quando cambiare pagina, ma in alcuni casi può essere importante specificare un comando di cambio pagina (oppure una regola per tutti i cambi pagina di un documento già impaginato).

Si ha quindi

<?newpage?>

XML 1.0:

Nasce in una raccomandazione W3C del 10 febbraio 1998.

È definito come un sottoinsieme di SGML, ma è molto più formalizzata della grammatica di SGML ed usa una notazione formale (Extended Backus-Naur Form).

Questa sintassi è più rigorosa ma di conseguenza anche molto più rigida.

Documenti ben formati o validi:

XML distingue due livelli di correttezza rilevanti per le applicazioni XML: i documenti **ben formati** ed i documenti **validi**.

Non è necessario avere il doctype per capire se un documento è ben formato.

Un documento non ben formato non da un warning, ma proprio un errore.

In un secondo momento se il documento è ben formato si passa a valutarne la validità.

In SGML un DTD è necessario per la validazione del documento; anche in XML un documento è valido se presenta un DTD ed è possibile validarla usando il DTD.

Tuttavia XML permette anche documenti ben formati, ovvero documenti che, pur essendo privi di DTD, presentano una struttura sufficientemente regolare e comprensibile da poter essere controllata.

Documenti XML ben formati:

Un documento XML si dice ben formato se:

- Tutti i tag di apertura e chiusura corrispondono e sono ben annidati;
- Esiste un elemento radice che contiene tutti gli altri;

- Gli elementi vuoti (senza contenuto) utilizzano un simbolo speciale di fine tag: </>;
- Tutti gli attributi sono sempre racchiusi tra virgolette;
- Tutte le entità sono definite.

HTML:

Merita un capitolo a parte.

HTML

BREVE STORIA DI HTML:

HTML oggi:

HTML è diventato qualcosa di diverso rispetto a ciò che è stato per molti anni: è passato da linguaggio di markup per documenti ipertestuali e multimediali ad un vero e proprio linguaggio per applicazioni.

È il World Wide Web che sta cambiando e l'attenzione è sempre più su interattività, dinamicità e manipolazione sofisticata (client-side) dei contenuti; di conseguenza il browser non è più soltanto uno strumento di lettura (senza dimenticare che inizialmente era anche un editor) ma diventa un motore generico per l'esecuzione di applicazioni.

HTML cambia quindi sia nella semantica che nella sintassi, rendendo vani molti degli sforzi fatti dal W3C negli anni.

Anche perché tuttora c'è ancora confusione tra diverse versioni dello standard, proposte parallele e tra gli stessi gruppi di lavoro.

Ad oggi esistono due linee di sviluppo parallele di HTML:

- HTML 5.2 è una Recommendation del W3C del 14 Dicembre 2017 e sospesa il 28 gennaio 2021 (<https://www.w3.org/TR/html52/>).
- HTML Living Standard del WHATWG, con ultimo aggiornamento risalente al 16 Marzo 2021 (<https://html.spec.whatwg.org/>).

HTML 4:

Il linguaggio HTML è un tipo di documenti SGML (esiste più di un DTD di HTML) progettato per marcare documenti ipertestuali.

Tramite HTML è possibile realizzare documenti con una semplice struttura, con aspetti grafici anche sofisticati, con testo, immagini, oggetti interattivi e link.

La cosiddetta “guerra dei browser” (1994-1998) ha portato alla creazione di numerosi elementi proprietari, estensioni, effetti sofisticati e caratteristiche puramente presentazionali. Dopo diverse versioni intermedie, HTML 4.0 (approvato dal W3C nel 1997) “chiude i giochi” ed aggiunge il supporto all'internazionalizzazione, gli style sheet, i frame, tabelle molto più ricche, ecc.

HTML 4.01 (1999) apporta modifiche minori.

Da allora lo standard è rimasto sostanzialmente invariato fino al 2008, quindi HTML 4 è il linguaggio in cui è stata scritta la stragrande maggioranza delle pagine Web di contenuti.

Da HTML a “tag soup”:

il w3c non ha avuto la forza di imporre ai produttori l'utilizzo di un insieme di tag ed altri elementi standardizzati.

Di conseguenza molte pagine HTML sono diventate “tag soup”, ossia un insieme di elementi non conformi allo standard.

I browser infatti si sono preoccupati poco della correttezza sintattica o strutturale dei documenti HTML; questo significa che esistono differenze anche sensibili tra un documento HTML corretto e un documento HTML visualizzabile da un browser Web (per interpretare i documenti sintatticamente non corretti ogni browser fa a modo suo, infatti prima di HTML 5 non è mai stato standardizzato il modo in cui fare il parsing di questi documenti).

Inoltre per gestire sia le pagine conformi allo standard che quelle non compatibili (proliferate dopo la “guerra dei browser”), sono stati introdotti due modelli di rendering: quirks mode e strict mode.

Quirks mode e strict mode:

Molte pagine HTML sono state scritte usando un modello scorretto e locale ad uno specifico browser, ovviamente queste stesse pagine non funzionano con i browser che implementano correttamente lo standard.

È quindi necessario capire se il browser deve interpretare una pagina secondo lo standard oppure no.

La soluzione è avere due stili di interpretazione di HTML e CSS:

- **quirks mode**: compatibile con il passato e più permissivo;
- **strict mode**: compatibile con le specifiche ufficiali.

Se la pagina non specifica niente, il browser adotta la modalità compatibile (quirks mode), altrimenti se l'autore lo richiede esplicitamente attiva il modo restrittivo e corretto.

Questo modello di autodichiarazione però non funziona benissimo (uno potrebbe dichiarare di avere un testo corretto quando non è così).

Infatti HTML 5 poi cambierà le regole standardizzando le regole di parsing (anche e soprattutto in caso di errori) e rendendo inutile l'autodichiarazione.

XHTML 1.0:

È html 4 con la sintassi xml e non ha avuto un gran successo, infatti è poco utilizzato.

XHTML 1.0 è una Recommendation del 2000.

È una riformulazione di HTML 4 come un'applicazione di XML 1.0.

L'elenco e la semantica di elementi e attributi non è assolutamente cambiata rispetto ad HTML 4.

I documenti devono essere ben formati, in particolare l'annidamento deve essere corretto.

Alcune altre differenze con HTML 4:

- I nomi di elementi e attributi sono minuscoli (XML è case-sensitive);
- Il tag finale è obbligatorio per elementi non vuoti;
- Gli elementi vuoti devono seguire la sintassi XML;
- I valori degli attributi devono sempre avere le virgolette.

(X)HTML 5:

Nel 2004, Firefox e Opera proposero al W3C la riapertura del Working Group su HTML per lo sviluppo di nuove versioni del linguaggio.

La proposta, che ignorava volutamente XHTML e la rigida sintassi di XML, venne bocciata dal W3C.

Venne così creata una comunità aperta chiamata WHATWG (Web Hypertext Application Technology Working Group) finanziata e supportata da Mozilla, Opera e Apple (poi anche da Google), che iniziò a lavorare ad una versione “intermedia” di HTML, “Web Application 1.0” (WA1), meno ambiziosa di XHTML 2.0.

Il chair, Ian Hickson, è ora un project manager a Google.

Nel 2007 il W3C dovette ammettere che queste modifiche avevano un impatto innegabile (“he who ships working code wins”) quindi riaprì il working group con tutti i membri del WHAT per creare una nuova versione del linguaggio, (X)HTML 5.

WA1 deve accettare l'amara realtà dei browser odierni che dicono che non è mai esistita una grammatica SGML per HTML, ma che il linguaggio è sempre stato o un'applicazione di XML (XHTML *), oppure una tag soup in cui i browser accettano ogni sorta di “porcheria” e fanno il meglio che possono.

“(X)HTML 5” diventa quindi solo “HTML 5” e successivamente solo “HTML”.

Il W3C deve adeguarsi, anche perché WA1 ha supporto da tutti i browser e di tutta l'industria delle applicazioni Web.

HTML Living Standard:

Dal 2011 il “nuovo” HTML è, per scelta del WG, una specifica perennemente in sviluppo (“living standard”) e questa instabilità diventa una feature:

“Because the specification is now a living document, we are today announcing two changes:

- *The HTML specification will henceforth just be known as "HTML".*
- *The WHATWG HTML spec can now be considered a "living standard".*

It's more mature than any version of the HTML specification to date, so it made no sense for us to keep referring to it as merely a draft.” - Ian Hickson, chair del WHATWG, 19 gennaio 2011.

Cambia completamente il modello di sviluppo del linguaggio che si allontana dall’approccio sistematico e democratico di evoluzione degli altri standard W3C.

Questa è la vittoria indiscussa dei produttori di browser: sia per le caratteristiche del linguaggio sia per il modo in cui è sviluppato.

Novità di Living Standard e HTML 5:

- Il linguaggio di markup è stato rivisto globalmente, introducendo nuovi elementi più specifici per usi e abitudini ormai consolidate (e.g., `<nav>`, `<section>` e `<article>` invece di `<div>`, `<audio>` e `<video>` invece di `<object>`, ecc.);
- L’uso di script per la modifica dinamica in memory del documento (DOM) è diventata parte integrante del linguaggio, e ne costituisce un elemento prevalente sulla sua linearizzazione in stringa HTML;
- La sintassi basata su SGML è definitivamente abbandonata e quella basata su XML è solo facoltativa; HTML è sufficientemente rilevante da non dover dipendere da una meta-sintassi terza per le sue caratteristiche;
- Oltre al markup, HTML definisce e descrive un certo numero di API per la caratterizzazione del DOM più sofisticata di quanto possibile col markup;
- XHTML5 è una linearizzazione facoltativa di HTML basata sulla sintassi XML, ha poche ma importanti differenze da HTML.

La fine della divisione su HTML:

Nel linguaggio esistevano due anime incompatibili, con obiettivi diversi:

- La componente W3C voleva stabilizzare le versioni di HTML5, per farle diventare Recommendation, con test di conformità e modello approvato con il processo W3C;
- La componente WHATWG voleva continuare lo sviluppo continuo di HTML e di altre tecnologie collegate, in maniera strettamente collegata ad implementazioni esistenti.

Nel 2011 i due gruppi si separano.

La licenza di “HTML Living Standard” prevede il riuso di parte delle specifiche; il W3C quindi pubblica un sottoinsieme delle specifiche di “HTML Living Standard” come “HTML 5.x”, alcune feature sono omesse mentre altre sono pubblicate in specifiche separate (es. Canvas 2D).

Negli anni il W3C approva un HTML 5.1, un HTML 5.2 e un draft iniziale di HTML 5.3, progressivamente divergenti da HTML Living Standard, però non c’è supporto da parte dei browser.

Per questo motivo nel maggio 2019 il W3C annuncia ufficialmente che le nuove

release del linguaggio saranno in carico al WHATWG e che cesserà la produzione di un linguaggio diverso da quello.

Basi di HTML

La struttura di un documento HTML:



in head si trovano i metadati, ovvero le informazioni sulla pagina che non sono visualizzate a schermo.

In body si trovano invece le cose che vengono visualizzate a schermo.

Esempio:

```
<?DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
    <head>
        <title> Document title </title>
    </head>
    <body>
        <h1> Major Header </h1>
        <p>This is a complete paragraph of a document. I write and
        write until I fill in several lines, since I want to see how
        it wraps automatically. Surely not a very exciting
        document.</p>
        <p>Did you expect <b>poetry</b>?</p>
        <p>Here you can see a paragraph <br> split by a &lt;br&gt;</p>
        <hr>
        <p> A list of important things to remember: </p>
        <ul>
            <li>Spaces, tabs and returns</li>
            <li>Document type declaration</li>
            <li>Document structure</li>
            <li>Nesting and closing tags</li>
        </ul>
    </body>
</html>
```

Risultato:

Major Header

This is a complete paragraph of a document. I write and write until I fill in several lines, since I want to see how it wraps automatically. Surely not a very exciting document.

Did you expect **poetry**?

Here you can see a paragraph split by a

A list of important things to remember:

- Spaces, tabs and returns
- Document type declaration
- Document structure
- Nesting and closing tags

i tag <hr> e
 sono elementi vuoti, ovvero non hanno contenuto e attributi

ELEMENTI INLINE:

Gli elementi inline (o di carattere) non spezzano il blocco (non vanno a capo) e si includono liberamente l'uno dentro all'altro.

Si dividono in elementi fontstyle e elementi phrase.

I tag fontstyle forniscono informazioni specifiche di rendering; molti di essi sono deprecati e si suggerisce comunque sempre di usare gli stili CSS (introdotti dopo proprio a causa delle sempre più crescenti richieste di aggiungere elementi tipografici).

I tag **fontstyle** sono:

- TT (TeleType, font monospaziato, ad es. Courier);
- I (corsivo);
- B grassetto;
- U (sottolineato - deprecato);
- S e STRIKE (testo barrato - deprecato);
- BIG, SMALL (testo più grande e più piccolo).

Gli elementi **phrase** invece hanno una caratteristica semantica e non tipografica:

```
<body class="container-fluid">
  <h1>Inline elements</h1>
  <p>Fontstyle elements define specific rendering.
  <ul>
    <li>text in <i>italics</i>.</li>
    <li>text in <b>bold</b>.</li>
    <li>text in <tt>teletype</tt> (i.e., <code>)
    <li>text in <u>underlined</u>.</li>
    <li>text in <s>strike-through</s> and <del>.
    <li>text in <sup>superscript</sup> and <sub>.
    <li>text in <big>bigger</big> and <small>.
  </ul>
  <p>The phrase elements define specific meanings.
  <ul>
    <li><em>emphasis</em>.
    <li><strong>major emphasis</strong>.
    <li><dfn>definition</dfn>.
    <li><code>program fragment</code>.
    <li><kbd>text entered by the user</k>.
    <li><var>program variable</var>.
    <li><abbr>abbreviation</abbr> and <acronym>.
    <li><span>generic inline element</span>.
    <li><cite>short quote</cite>.
    <li><q>long quote</q>.
  </ul>
</body>
```

F Fontstyle elements define specific rendering.

- text in *italics*.
- text in **bold**.
- text in `teletype` (i.e., monospaced font).
- text in underlined.
- text in ~~strike-through~~ and also ~~strike-through~~.
- text in ^{superscript} and _{subscript}.
- text in **bigger** and smaller.

T The phrase elements define specific meanings.

- **emphasis**
- **major emphasis**
- **definition**
- **program fragment**
- **text entered by the user**
- **program variable**
- **abbreviation and acronym**
- **generic inline element**
- **short quote**
- **"long quote"**.

Questi elementi anche se non sono deprecati sono abbastanza caduti in disuso.

I tag em e strong indicano che le cose al loro interno vanno enfatizzati nel primo caso e molto enfatizzati nel secondo; a livello di implementazione poi i browser hanno deciso che il primo è renderizzato in corsivo ed il secondo il grassetto, quindi sono diventati equivalenti ad i e b.

span è l'elemento generico degli inline.

ELEMENTI DI BLOCCO:

I **tag di blocco** definiscono l'esistenza di blocchi di testo che contengono elementi inline.

Elementi fondamentali:

- <p>: paragrafo;
- <div>: Blocco generico;
- <pre>: blocco preformatto, qui i whitespace non vengono collassati (come invece avviene negli altri blocchi);
- <address>: l'autore della pagina;
- <blockquote>: Citazione lunga con il margine sinistro leggermente spostato, è interessante per motivi storici: prima dell'invenzione del css veniva utilizzato per spostare il testo orizzontalmente nella pagina.

Blocchi con ruolo strutturale:

- <h1>, <h2>, <h3>, <h4>, <h5>, <h6>: Titoli di varia dimensione (ogni dimensione coincide a livello teorico con una sottosezione).

div è l'elemento generico dei blocchi.

The screenshot shows a browser's developer tools element inspector. On the left, the DOM tree is displayed with the following code:

```
<body class="container-fluid">
  <h1>Block elements</h1>
  <p>The block elements define text blocks that contain inline elements.</p>
  <h2>Basic elements:</h2>
    <p>paragraph. White space is not meaningful</p>
    <div>generic block</div>
    <pre>preformatted block. White space is meaningful</pre>
    <address>the author of the page</address>
    <blockquote>This is a long quote</blockquote>
  <h2>Blocks with a structural role</h2>
    <h1>First level heading</h1>
    <h2>Second level heading</h2>
    <h3>Third level heading</h3>
    <h4>Fourth level heading</h4>
    <h5>Fifth level heading</h5>
    <h6>Sixth level heading</h6>
</body>
```

On the right, the element inspector displays the selected element and its properties. For example, the first paragraph is highlighted with a yellow background, and its properties panel shows "paragraph. White space is not meaningful". Other elements like the pre tag and address tag also have their specific properties listed.

ELEMENTI DI LISTA:

Le liste sono contenitori di elementi omogenei tra loro.

Nel tempo sono esistiti vari tipi di lista, attualmente le liste più importanti sono:

- : **unordered list**, ovvero lista non ordinata di elementi ;
Attributi: type (disc, square, circle);
- : **ordered list**, ovvero lista ordinata di elementi ;
Attributi: start (valore iniziale) e type (1, a, A, i, I);
- <dl>: **lista di definizioni**, ovvero lista composta da molti elementi <dt> (definition term) e <dd> (definition data).

```

<body class="container-fluid">
  <h1>List elements</h1>
  <ul>
    <li>First element</li>
    <li>Second element</li>
    <li>Third element</li>
  </ul>
  <ol>
    <li>First element</li>
    <li>Second element</li>
    <li>Third element</li>
  </ol>
  <dl>
    <dt>First term</dt><dd>First definition
    <dt>Second term</dt><dd>Second definition
    <dt>Third term</dt><dd>Third definition
  </dl>
</body>

```

List elements

- First element
- Second element
- Third element

1. First element
2. Second element
3. Third element

First term
First definition

Second term
Second definition

Third term
Third definition

```

<body class="container-fluid">
  <h1>List elements</h1>
  <ul>
    <li>First element</li>
    <li type="square">Second element
    <li type="circle">Third element
  </ul>
  <ol>
    <li type="1">First
    <li type="i">Second
    <li type="A">Third
  </ol>
</body>

```

List elements

- First element
- Second element
- Third element

- i. First element
- ii. Second element
- C. Third element

ELEMENTI GENERICI:

<div> e **** sono cosiddetti elementi generici: sono privi di caratteristiche semantiche o presentazionali predefinite ed assumono quelle desiderate con l'aiuto dei loro attributi (**style**, **class**, **lang**).

<div> mantiene unicamente la natura di elemento blocco, mentre **** unicamente la natura di elemento inline.

Esempio:

Per identificare con elementi appositi tutti i titoli di film si può utilizzare **<titolofilm>**, ma è necessario definire un nuovo tag HTML.

Altrimenti si può usare l'elemento generico **** appartenente alla classe **titolofilm** (le classi sono trattate successivamente, per questo esempio è sufficiente pensare che tutti gli elementi di una classe hanno le stesse proprietà): ****.

PICCOLI EFFETTI GRAFICI:

- **<hr>** (horizontal rule): genera una piccola riga orizzontale che attraversa lo schermo e della quale si può controllare la larghezza e lo spessore.
- **
** (break): andata a capo forzata.

ELEMENTI DI STRUTTURA:

HTML nasce con una struttura piatta che non prevede annidamento di elementi, quindi l'unico modo che trovano gli sviluppatori per dare un minimo di struttura gerarchica al codice (caratteristica che dà molti vantaggi) è mettere una serie di div uno dentro l'altro.

Questo porta a documenti molto complicati e di difficile lettura.

Per questo motivo HTML 5 introduce nuovi elementi con una semantica precisa per organizzare il documento in contenitori che possono essere più facilmente manipolati, esportati e modificati.

I principali sono:

- **<main>**: la parte principale della pagina, ovvero un contenitore di testo o di sottosezioni che rappresenta la parte più importante della pagina; Al suo interno si possono trovare:
 - **<section>**: un contenitore generico annidabile;

- **<article>**: una parte del documento autonoma e pensata per essere riutilizzata e distribuita come unità atomica (post di un blog, news, feed, ecc);
- **<aside>**: sezione collegata al testo ma separata dal flusso principale (note a margine, sidebar, incisi, pubblicità, ecc);
- **<header>** e **<footer>**: elementi iniziali e finali di un documento;
- **<nav>**: lista di navigazione;

Esempio:

Benvenuto nella mia home!

Si divide in due parti: short-bio e post giornalieri...

Bio

Nato a, laureato il, dottorato il, etc.

9 marzo 2017

Che giornata grigia.

13 marzo 2017

Una bella giornata di sole.

Questa pagina può essere espressa con una struttura piatta in questo modo:

```
<h1>Benvenuto nella mia home!</h1>
<p>Si divide in due parti: short-bio e post giornalieri...</p>
<h2>Bio</h2>
<p>Nato a, laureato il, dottorato il, etc.</p>
<h2>9 marzo 2017</h2>
<p>Che giornata grigia.</p>
<h2>13 marzo 2017</h2>
<p>Una bella giornata di sole.</p>
```

Ma stilisticamente è molto meglio organizzata gerarchicamente:

```
<div id="main">
    <h1>Benvenuto nella mia home!</h1>
    <p>Si divide in due parti: short-bio e post giornalieri...</p>
    <div id="bio">
        <h1>Bio</h1>
        <p>Nato a, laureato il, dottorato il, etc.</p>
    </div>
    <div id="posts">
        <div id="09-03-17">
            <h1>09 marzo 2017</h1>
            <p>Che giornata grigia.</p>
        </div>
        <div id="13-03-17">
            <h1>13 marzo 2017</h1>
            <p>Una bella giornata di sole.</p>
        </div>
    </div>
</div>
```

Con HTML 5 si può scrivere in questo modo:

```
<main>
    <h1>Benvenuto nella mia home!</h1>
    <p>Si divide in due parti: short-bio e post giornalieri...</p>
    <section id="bio">
        <h1>Bio</h1>
        <p>Nato a, laureato il, dottorato il, etc.</p>
    </section>
    <section id="posts">
        <article id="09-03-17">
            <h1>9 marzo 2017</h1>
            <p>Che giornata grigia.</p>
        </article>
        <article id="13-03-17">
            <h1>13 marzo 2017</h1>
            <p>Una bella giornata di sole.</p>
        </article>
    </section>
</main>
```

HEADER E FOOTER:

HTML 5 definisce quindi anche due specifici elementi usati sia per contenuti ripetuti (nel caso di divisione in pagine), sia per contenuti che solitamente sono visualizzati all'inizio o alla fine di una sezione, sottosezione, articolo, ecc.

- <header>: contiene l'intestazione della sezione corrente o dell'intera pagina ed è solitamente usato per tabelle di contenuti, indici, form di ricerca o intestazioni, inoltre può essere anche usato come contenitore degli headings (H1, ..., H6);

- <footer>: contiene la “parte conclusiva” della sezione corrente o dell’intera pagina; è usato principalmente per mostrare informazioni testuali (non metadati) sugli autori della pagina, copyright, produzione, licenze, ecc; non deve essere visualizzato necessariamente a fondo pagina è può essere usato anche per contenere intere sezioni come appendici, allegati tecnici, colophon, ecc.

LISTE DI NAVIGAZIONE:

HTML 5 riprende da XHTML 2.0 anche le “liste di navigazione”, ovvero particolari sezioni dedicate a raggruppare link alla pagina corrente, a sue sottosezioni o ad altre pagine.

Per creare tali liste si usa l’elemento <nav> che molto spesso viene inserito in <header> o <footer>.

Le sezioni <nav> sono molto utili per l’accessibilità in quanto possono essere più facilmente identificate ed interpretate anche da utenti disabili (tramite screen readers od altri ausili). Inoltre possono essere usate anche per attivare o disattivare le funzionalità di navigazione in base allo user-agent (browser) che sta accedendo alla pagina.

Esempio:

```
<header>
  <nav>
    <h1>Navigation</h1>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </nav>
</header>
<main>... </main>
<footer>
  <nav>
    <h1>Contact W3C</h1>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </nav>
</footer>
```

COLOPHON:
 Termine con cui viene indicata la formula posta in fine ai libri dai primordi della stampa ai primi anni del sec. XVI: conteneva, generalmente, il nome dello stampatore, il luogo e la data di stampa e, spesso, altre notizie inerenti alla pubblicazione del libro (nome del correttore, di chi aveva concorso nella spesa, ecc.).
 [...] Nell’editoria contemporanea tale uso è stato largamente ripreso come un ornamento del libro, in edizioni speciali o di lusso.
 Da
[https://www.treccani.it/encyclopedi/a/clophon_\(Encyclopaedia-Italiana\)/](https://www.treccani.it/encyclopedi/a/clophon_(Encyclopaedia-Italiana)/)

LINK IPERTESTUALI:

storicamente esistono vari tipi di link, i link tra sito e sito, i link tra documento e documento ed altro (cerca)15:40.

I link sono definiti con elementi `<a>` (dall'inglese **anchors**, ovvero ancora nel documento). `<a>` è sintatticamente un elemento inline (dentro ai blocchi, come ``, `<i>`, ecc.) e l'unica limitazione è che non può avere elementi annidati dentro di sé.

Attributi:

- **href**: specifica l'URI della destinazione, quindi ` ... ` è l'ancora di partenza di un link;
- **name**: specifica un nome che può essere usato come ancora di destinazione di un link, quindi ` ... ` è l'ancora finale di un link.

Esempio:

The diagram illustrates the creation of a hypertext structure. On the left, a code editor shows an HTML document with three levels of headings and associated anchors. Red arrows point from the anchor elements to their corresponding destination sections on the right, which is a generated table of contents or summary page.

HTML Document Content:

```
<p>
  <a href="#section1">
    Introduzione</a><br>
  <a href ="#section2">
    Concetti di base</a><br>
  <a href ="#section2.1">
    Definizione del problema</a><br>
    ...
  </p>

  <h2><a name="section1">
    Introduzione</a></h2>
    ...sezione 1...
  <h2><a name ="section2">
    Concetti di base</a></h2>
    ...sezione 2...
  <h3><a name ="section2.1">
    Definizione del problema</a></h3>
    ...sezione 2.1...

```

Generated Summary Page:

Sommario	
Introduzione	...sezione 1...
Concetti di base	...sezione 2...
Definizione del problema	...sezione 2.1...

IMMAGINI:

Le immagini inline sono definite con l'elemento ``.

I formati tipici delle immagini sono jpeg, gif e png, ma ne esistono di tanti altri tipi.

`` è un elemento vuoto completamente definito dai suoi attributi.

Esempio:

```

```

Attributi:

- **src** (obbligatorio): URL della risorsa contenente l'immagine;
- **alt**: testo alternativo se l'immagine non è mostrata;
- **name**: un nome per riferirsi all'immagine;
- **width**: forza una larghezza in pixel specifica per l'immagine;
- **height**: forza un'altezza in pixel specifica per l'immagine.

Esempi:

```
<body class="container-fluid">
  <h1>Images</h1>
  <p>Images are managed by the &lt;img> element, which is empty (no content, no end tag). The src attribute is used to specify the URI of the image file. The alt attribute specifies an alternative text. The height attribute specifies the height in pixels.</p>
  <p>The image can be:</p>
  <ol>
    <li>... in the same folder as the HTML document:<br/>
      
    </li>
    <li>... in a different folder:<br/>
      
    </li>
    <li>... on a different server:<br/>
      
    </li>
  </ol>
</body>
```

Images

Images are managed by the `` element, which is empty (no content, no end tag). The `src` attribute is used to specify the URI of the image file. The `alt` attribute specifies an alternative text. The `height` attribute specifies the height in pixels.

The image can be:

1. ... in the same folder as the HTML document:



2. ... in a different folder on the same server:



3. ... on a different server:



```
<body class="container-fluid">
  <h1>Images - stretching</h1>
  <p>Specifying height and/or width lets control dimensions simply:<br/>
  <ol>
    <li>Specify one dimension: the image will be resized proportionally around that dimension<br/>
      
    </li>
    <li>Specify both dimensions: the proportions and will be stretched regardless of the original proportions and will be stretched<br/>
      
    </li>
    <li>Specify no dimensions: the image will be shown in the original dimensions.<br/>
      
    </li>
  </ol>
</body>
```

Images - stretching

Specifying height and/or width lets control dimensions simply:

1. Specify one dimension: the image will be resized proportionally around that dimension
2. Specify both dimensions: the image will be resized regardless of the original proportions and will be stretched
3. Specify no dimensions: the image will be shown in the original dimensions.



Esiste inoltre un elemento `<figure>` che può racchiudere un'immagine permettendo di darle una didascalia e di renderla un blocco (mentre `` da solo è un elemento inline).

Esempio:

```
<h1>Figure</h1>
<figure>
  
  <figcaption>A cake</figcaption>
</figure>
```

Figure



A cake

Attraverso l'attributo **srcset** è possibile indicare più risorse per la stessa immagine da usare alternativamente.

Per esempio a seconda dello user agent si possono scegliere immagini con dimensioni (e quindi risoluzioni) diverse, in modo da non dover fare affidamento al ridimensionamento algoritmico dell'immagine (con l'inevitabile perdita di dettaglio).

```

```

- L'attributo **srcset** indica quali risorse usare e la dimensione "reale" di ciascuna risorsa.
- L'attributo **sizes** indica quali tipi di schermo associare a ciascuna risorsa.
- L'attributo **src** è un fallback nel caso che il browser non conosca srcset.

Quindi srcset si può utilizzare per fare pagine con **immagini responsive**.

TABELLE:

Le tabelle si possono utilizzare sia per rappresentare dati che per dare effetti tipografici (tramite le tabelle di layout, ora cadute in disuso).

Le tabelle vengono specificate riga per riga e per ognuna di esse si possono precisare gli elementi, ovvero intestazioni o celle normali.

Una tabella può anche avere una didascalia, un'intestazione (un titolo, diverso dalle intestazioni nelle celle) ed una sezione conclusiva.

E' possibile descrivere una sola volta le caratteristiche visive di tutta la colonna.

Esempi:

```
<body>
  <h1>Tables</h1>
  <table border="1">
    <tr>
      <th>Salesman</th><th>Jan</th>
      <th>Feb</th> <th>Mar</th>
    </tr>
    <tr>
      <th>John Smith</th> <td>12000</td>
      <td>13000</td><td>15000</td>
    </tr>
    <tr>
      <th>Alice Green</th><td>7000</td>
      <td>9000</td><td>11000</td>
    </tr>
    <tr>
      <th>Hugh Brown</th><td>25000</td>
      <td>23000</td><td>30000</td>
    </tr>
  </table>
</body>
```

Tables

Salesman	Jan	Feb	Mar
John Smith	12000	13000	15000
Alice Green	7000	9000	11000
Hugh Brown	25000	23000	30000

```
<body>
  <h1>Tables</h1>
  <table border="2" width="80%" cellpadding="5" cellspacing="0">
    <caption>Sales in first quarter</caption>
    <col style="background-color:#FFFFBB" width="70%">
    <col span="3" width="10%">
    <thead>
      <tr>
        <th>Salesman</th><th>Jan</th>
        <th>Feb</th> <th>Mar</th>
      </tr>
    </thead>
    <tfoot>
      <tr>
        <th>Totals</th><th>34000</th>
        <th>45000</th> <th>56000</th>
      </tr>
    </tfoot>
    <tbody>
      <tr>
        <th>John Smith</th> <td>12000</td>
        <td>13000</td><td>15000</td>
      </tr>
      <tr>
        <th>Alice Green</th><td>7000</td>
        <td>9000</td><td>11000</td>
      </tr>
      <tr>
        <th>Hugh Brown</th><td>25000</td>
        <td>23000</td><td>30000</td>
      </tr>
    </tbody>
  </table>
</body>
```

Tables

Sales in first quarter			
Salesman	Jan	Feb	Mar
John Smith	12000	13000	15000
Alice Green	7000	9000	11000
Hugh Brown	25000	23000	30000
Totals	34000	45000	56000

Un altro tipo di tabella sono le tabelle di layout che sono state inventate prima del css e servivano proprio ad impaginare la pagina.

Le tabelle di layout sono state fortemente criticate per ragioni di accessibilità e sono state abbandonate con l'introduzione del css.

Esempio:

```
<body>
  <h1>Tables</h1>
  <table border=2 cellpadding="5" cellspacing="0" width="70%">
    <tr>
      <td>First row <br> First column</td>
      <td colspan="2">First row<br>Second column</td>
    </tr>
    <tr>
      <td colspan="2">Second row <br> First column</td>
      <td rowspan="2">Second row <br> Third column</td>
    </tr>
    <tr>
      <td>Third row <br> First column</td>
      <td>Third row <br> Second column</td>
    </tr>
  </table>
</body>
```

Tables

First row First column	First row Second column	
Second row First column		Second row Third column
Third row First column	Third row Second column	

FORM:

Un form è una zona della pagina dove si trovano i widget, ovvero parti interattive dove l'utente può eseguire varie azioni (generalmente inserire informazioni che poi vengono spedite al server e lì elaborate).

I form quindi sono strettamente legati alle applicazioni server-side, che li utilizzano come strumenti per raccogliere i dati dell'utente.

Il browser raccoglie dati dall'utente con un form, crea una connessione HTTP con il server specificando una ACTION (cioè un'applicazione che funga da destinatario, trattata meglio successivamente) a cui fare arrivare i dati, il destinatario riceve i dati, li elabora e genera un documento di risposta, esso infine viene spedito tramite il server HTTP al browser.

Nei form inoltre vengono utilizzati controlli tipati e nominati per l'inserimento dei dati, per esempio in campi di inserimento dati, pulsanti, bottoni radio, checkbox, liste a scomparsa, ecc.

Esempio:

```
<h1>Form</h1>
<form method="get" action="http://www.site.com/serverside.py">
  <p>
    <label><i>Name:</i> <input type="text" name="name" value="John" size="15"></label>
    <label><i>Surname:</i> <input type="text" name="surname" value="Smith" size="25"></label>
  </p>
  <p>Gender:
    <label><input type="radio" name="gender" value="m" checked>Male</label>
    <label><input type="radio" name="gender" value="f">Female</label>
    <label><input type="radio" name="gender" value="x">Won't say</label>
  </p>
  <p>Likes:
    <label><input type="checkbox" name="likes" value="art" checked>Art</label>
    <label><input type="checkbox" name="likes" value="cinema">Cinema</label>
    <label><input type="checkbox" name="likes" value="comics">Comics</label>
    <label><input type="checkbox" name="likes" value="literature">Literature</label>
    <label><input type="checkbox" name="likes" value="cuisine">Cuisine</label>
  </p>
  <p>Name: John Surname: Smith
    <label>Nationality: <select>
      <option value="">- sel
      <option value="male">Male <input checked="" type="radio" name="gender" value="m"/>
      <option value="female">Female <input type="radio" name="gender" value="f"/>
      <option value="won'tsay">Won't say <input type="radio" name="gender" value="x"/>
    </select></label>
    Likes: <input checked="" type="radio" name="likes" value="art"/> Art <input type="radio" name="likes" value="cinema"/> Cinema <input type="radio" name="likes" value="comics"/> Comics <input checked="" type="radio" name="likes" value="literature"/> Literature <input type="radio" name="likes" value="cuisine"/> Cuisine
  </p>
  <p><input type="submit" name="sub" value="OK" /> <input type="reset" name="can" value="Cancel" />
</form>
```

Gli elementi di un form sono:

- **<form>**: il contenitore dei widget del form;

Attributi:
 - method: il metodo HTTP da usare (GET, POST);
 - action: l'URI dell'applicazione server-side da chiamare;
- **<input>, <select>, <textarea>**: i widget del form, i browser forniscono widget diversi nell'interfaccia in base al tipo di questo campo;

Attributi:
 - name: il nome del widget usato dall'applicazione server-side per determinare l'identità del dato (attenzione: tutte le checkboxes e tutti i radio buttons dello stesso gruppo condividono lo stesso nome.);
 - type: il tipo di widget (input, checkbox, radio, submit, cancel, etc.);
- **<button>**: un bottone cliccabile (diverso dal submit);
- **<label>**: il nome visibile del widget.

HTML 5 introduce molte novità per velocizzare, semplificare e controllare l'inserimento dei dati da parte dell'utente, in particolare per quanto riguarda i form sono stati specificati alcuni attributi e nuovi tipi per gli oggetti in input, inoltre è stato aggiunto un meccanismo di validazioni degli input client side che non richiede l'utilizzo di JavaScript.

Più nello specifico:

- L'oggetto input è arricchito con due attributi che permettono di controllare il focus e aggiungere suggerimenti agli utenti:
 - **placeholder**: contiene una stringa che sarà visualizzata in un campo di testo se il focus non è su quel campo;
 - **autofocus**: indica il campo sul quale posizionare il focus al caricamento del form.
- Nuovi tipi di input:
 - **email**: in fase di validazione il browser verifica se il testo inserito contiene il simbolo '@' e se il dominio è (sintatticamente) corretto;
 - **url**: verifica che il testo inserito segua le specifiche degli URL;
 - **number**: il browser visualizza bottoni per incrementare/decrementare il valore, è possibile specificare il valore minimo, massimo e l'unità di modifica come altri attributi di input;
 - **range**: il browser visualizza uno slider per incrementare o decrementare un valore numerico, anche qui è possibile specificare il valore minimo, massimo e l'unità di modifica;
 - **date**: richiede al browser la visualizzazione di un calendario tra cui selezionare una data; esistono vari tipi collegati ad esso come month, week, time (orario), ecc;
 - **search**: testo renderizzato in modo diverso su alcuni browser (per esempio safari per iPhone);
 - **color**: usato per mostrare una tavolozza di colori, da cui selezionare un codice RGB.

Esempio:

```
<p>Nome: <input name="nome" type="text" placeholder="Inserisci qui il tuo nome" size="40" autofocus></p>
<p>Cognome: <input name="cognome" type="text" placeholder="Inserisci qui il tuo cognome" size="40" autofocus></p>
<p>Email: <input name="email" type="text" placeholder="Inserisci qui la tua email" size="40" autofocus></p>
<p>Lezioni: <input type="number" step="1" value="1"></p>
<p>Livello: <input type="range" max="10" step="1" value="3"></p>
<p>Data Inizio: <input name="data_inizio" type="date" value="2011-04-01" placeholder="Inserisci la data di inizio" size="40" autofocus></p>
<p><input type="submit" value="Iscriviti" /></p>
```

Nome: Inserisci qui il tuo nome

Cognome: Inserisci qui il tuo cognome

Email:

Lezioni: 1

Livello:

Data Inizio: 2011-04-01

Iscriviti

Today

- Validazione automatica:

È prevista poi la possibilità di validare un form client-side, senza ricorrere a funzioni Javascript aggiuntive.

Dopo l'evento **submit** i dati inseriti nel form sono verificati in base al **tipo** di ogni campo (email, URL, etc.); nel caso questi non soddisfino le specifiche si può catturare l'evento **invalid** ed implementare comportamenti specifici.

Inoltre è possibile indicare i campi obbligatori, tramite l'attributo **required** dell'oggetto input.

L'attributo **novalidate** invece può essere usato per indicare al browser di non fare controlli automatici su l'intero form o su uno specifico campo.

Esempi:

```
<input name="mail" type="email" required>
<input name="url" type="url" novalidate>
```

COREATTRS (ATTRIBUTI GLOBALI):

Gli attributi globali sono quelli definiti per tutti gli elementi del linguaggio HTML.

I coreattrs costituiscono attributi di qualificazione e associazione globale degli elementi, per lo più utili a CSS e link ipertestuali.

I coreattrs sono:

- **Id**: un identificativo unico (per tutto il documento);
- **Style**: un breve stile CSS associato al singolo elemento;
- **Class**: una lista (separata da spazi) di nomi di classe (per attribuzione semantica e di stile CSS);
- **Title**: un testo secondario associato all'elemento (per accessibilità ed informazioni aggiuntive).

Esempio:

```
<!DOCTYPE html> This is the first paragraph of a short text.  
<html> This is the second paragraph of the same text.  
  <head> This is the third paragraph of the same text.  
    <title>Table elements <  
    <meta charset="utf-8"> terzo!  
    <meta name="viewport" c  
  <style>  
    #p1 { font-family: "Times New Roman", "Times", serif; }  
    #p2, #p3 { font-family: "Verdana", "Arial", sans-serif; }  
    .first { color: blue; }  
    .second {color: green; }  
    .special {font-weight: bold; }  
    .notspecial {font-weight: normal; }  
  </style>  
</head>  
<body>  
  <p id="p1" class="first special">This is the first paragraph of a short text.</p>  
  <p id="p2" class="second">This is the <span id="s1" class="first">second</span>  
  paragraph of the same text.</p>  
  <p id="p3" class="special">This is the <span id="s2" class="notspecial"  
  title="terzo!">third</span> paragraph of the same text.</p>  
</body>  
</html>
```

I18N (ATTRIBUTI GLOBALI):

Gli attributi i18n garantiscono l'internazionalizzazione del linguaggio e la coesistenza nello stesso ambiente di script diversi:

- Lang: è una codifica che indica i linguaggi umani (stringa a due caratteri: it, en, fr, ecc. da RFC1766);
- Dir: può essere uno dei due valori ltr (left to right) o rtl (right to left) per indicare la direzione di flusso secondario del testo.

i18N:

The terms are frequently abbreviated to the numeronyms i18n (where 18 stands for the number of letters between the first i and the last n in the word internationalization, a usage coined at Digital Equipment Corporation in the 1970s or 1980s).
Da
https://en.wikipedia.org/wiki/Internationalization_and_localization

EVENTI (ATTRIBUTI GLOBALI):

Gli attributi di evento permettono di associare script a particolari azioni sul documento e sui suoi elementi.

Per esempio:

Onclick, ondoubleclick, onmouseover, onkeypress, ecc.

ATTRIBUTI DATA:

Gli attributi data hanno una sintassi del tipo: **data-*"nomeattributo"***.

Sono attributi personalizzati che possono essere utilizzati da applicazioni e script Javascript senza inquinare troppo lo spazio dei nomi.

Esempio:

in Bootstrap:

```
<button id="first" data-bs-toggle="modal" data-bs-target="#exampleModal">  
  show modal  
</button>
```

Non tutti i caratteri sono ammessi, per esempio non è possibile scrivere nomi contenenti maiuscole, mentre i trattini vengono automaticamente convertiti in maiuscole nel dataset.
Gli attributi data sono accessibili in CSS con lo stesso nome:

Esempio:

```
button['data-bs-toggle'] { color: blue; }
```

Invece in Javascript sono accessibili con l'array dichiarativo element.dataset."nomeattributo":

Esempio:

```
la variabile let a = document.getElementById('first').dataset['bsToggle'] ha  
come valore "modal".
```

ATTRIBUTI ARIA:

HTML è di per sé accessibile, tuttavia nell'uso comune esistono tante cattive abitudini che creano pagine web perfettamente funzionanti all'apparenza ma impossibili da usare per chi è disabile.

Ad esempio:

```
<button class="button" onclick="doSomething()">Clicca qui</button>
```

e

```
<span class="button" onclick="doSomething()">Clicca qui</span>
```

sono visivamente identici ma il secondo non è accessibile dai non vedenti in quanto nulla indica allo screen reader che quello è un pulsante.

La specifica WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications) fornisce molte tecniche per rendere accessibili pagine HTML, alcune sono:

- Attributo **role** per caratterizzare la semantica attesa dell'elemento;
- Attributo **tabindex** per specificare la possibilità di selezionare l'elemento con la tastiera;
- Attributi **aria-****nomeattributo** per creare specifiche funzionalità per i vari elementi a seconda del loro scopo.

Esempio:

Questo elemento è stato reso accessibile utilizzando gli attributi aria:

```
<span role="button" class="button" onclick="doSomething()" tabindex="0">  
Clicca qui</span>
```

ENTITÀ DI HTML:

HTML definisce un certo numero di entità per indicare quei caratteri che sono proibiti in quanto utilizzati nella sua sintassi (per esempio <, >, &, ", ecc.) o perché non presenti nell'ASCII a 7 bit.

Alcune principali entità sono:

amp	&	quot	"
lt (less than)	<	gt (greater than)	>
Aelig	Æ	Aacute	Á
Agrave	À	Auml	Ä
aelig	æ	aacute	á
agrave	à	auml	ä
ccedil	ç	ntilde	ñ
reg	®	nbsp (non-breaking space)	

Inoltre si possono anche usare entità numeriche:

Libertà Libertà

འ好 你好

HTML	Shown as
libertà	libertà
© 2018 Fabio Vitali	© 2018 Fabio Vitali
François	François
5 > 3 & 5 < 12	5 > 3 & 5 < 12

COLORI (TIPO DI DATI):

In molte situazioni (sfondi, caratteri, ecc.) è utile specificare un colore, in questo caso HTML fornisce due modi per farlo:

- **Codice RGB preceduto da un carattere di hash:** si usano due caratteri esadecimali ciascuno per esprimere la quantità di Rosso, Verde e Blu del colore (00 significa assenza, FF significa presenza massima), in questo modo è possibile descrivere 4096 colori diversi;
- **Nome:** sono definiti 16 nomi di colori: black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua; attenzione: Microsoft definisce 256 nomi di colori, che IE accetta ma che Netscape ignora, quindi utilizzare quei nomi causa incompatibilità tra i browser.

Attenzione: da HTML 4 in poi l'uso esplicito di colori nel documento HTML è deprecato e si suggerisce di usare invece i fogli di stile.

Esempi:

```
<font color="#FF0000">testo in rosso</font>
<body bgcolor="#008080">sfondo teal</body>
<td bgcolor="yellow">sfondo giallo</td>
```

LUNGHEZZE (TIPO DI DATI):

HTML usa le lunghezze per tutti gli oggetti con una presenza sulla pagina di dimensioni determinate (immagini, tabelle, frame, ecc.).

Esistono tre tipi di lunghezze:

- **Pixel:** una dimensione in punti di schermo; è espressa come un numero assoluto:
``
- **Percentuali:** una dimensione in proporzione alla dimensione del contenitore; è espressa come un numero seguito da un carattere %:
`<table width="100%">...</table>`
- **Multi-lunghezze:** una sequenza di valori di lunghezza; in questo caso il carattere "*" divide la quantità restante in parti uguali (dopo aver tolto le lunghezze esplicite in pixel e percentuale); è una lista separata da virgolette di valori numerici e la si usa ad esempio nei gruppi di colonne di una tabella:
`<colgroup width="150, 25%, *, *">`

MAIUSCOLO/MINUSCOLO:

HTML non fa distinzioni tra caratteri maiuscoli o minuscoli nelle parole di linguaggio (XHTML invece si e richiede che sia tutto scritto in minuscolo).

Editor di antica concezione scrivevano i tag in maiuscolo (per meglio distinguere dal testo), i nuovi invece in minuscolo per avere una maggiore compatibilità com XHTML.

CARATTERI DI WHITESPACE:

HTML collappa tutti i caratteri di whitespace (SPACE, TAB, CR, LF) in un unico spazio (SPACE).

Questo permette di organizzare il sorgente in maniera leggibile senza influenzare la visualizzazione su browser.

L'entità **&nbsp** permette di inserire spazi che non vengono collassati in uno unico.

VIRGOLETTA NEGLI ATTRIBUTI:

Tutti i valori degli attributi che sono TOKEN (cioè parole di caratteri e numeri senza spazi e che inizino con un carattere) possono essere scritti senza virgolette.

Esempio:

Si può scrivere

```
<p align=center>testo al centro</p>  
al posto di  
<p align="center">testo al centro</p>
```

ASSENZA DI TAG:

Alcuni tag (ad esempio <HTML>, <BODY>), possono essere omessi.

Esempio:

Si può scrivere

```
<p>Questo è un documento valido in HTML, ma non in XHTML</p>  
al posto di  
<html>  
    <body>  
        <p>Questo è un documento valido sia in HTML, sia in  
        XHTML</p>  
    </body>  
</html>
```

OMISSIBILITÀ DEL TAG DI CHIUSURA:

Di alcuni elementi (per esempio <P>, , <OPTION>, ecc) può essere omesso il tag finale, il quale è implicitamente dedotto dal contesto.

Esempio:

Si può scrivere

```
<ul>  
    <li>Primo elemento  
    <li>Secondo elemento  
    <li>Terzo elemento  
</ul>  
al posto di  
<ul>  
    <li>Primo elemento</li>  
    <li>Secondo elemento</li>  
    <li>Terzo elemento</li>  
</ul>
```

ATTRIBUTI DI SOLO VALORE:

Alcuni attributi (nowrap, selected, etc.) vengono omessi quando attributo e valore sono la stessa parola, scrivendo solo il valore equivalente.

Esempio:
Si può scrivere

```
<select>
    <option>prima voce
    <option selected>seconda voce
    <option>terza voce
</select>
```

al posto di

```
<select>
    <option>prima voce</option>
    <option selected="selected">seconda voce</option>
    <option>terza voce</option>
</select>
```

TAG <HEAD>:

L'elemento `<head>` contiene informazioni (anche dette metainformazioni) che sono rilevanti per tutto il documento; esse sono:

- **<title>**: il titolo del documento
 - L'elemento `<title>` contiene semplice testo (non può contenere elementi HTML) che definisce il titolo del documento.
 - Il contenuto dell'elemento `<title>` ha tre utilizzi:
 - Viene posto come titolo della finestra nei sistemi operativi a finestra;
 - Viene utilizzato come nome illustrativo della pagina nei segnalibri del browser;
 - Viene utilizzato come nome illustrativo della pagina nei motori di ricerca, inoltre i motori di ricerca trattano con più importanza il contenuto dell'elemento `title` rispetto al resto del contenuto della pagina.
- **<script>**: librerie di script
 - Con il tag `script` si possono definire blocchi di funzioni di un linguaggio di script.
- **<style>**: librerie di stili
 - Con il tag `style` si possono definire blocchi di stili di un linguaggio di stylesheet.
- **<link>**: collegamenti ad altri documenti che influiscono sull'intera pagina
 - A volte può essere utile posizionare esternamente le specifiche di script e style, in questo caso si usa il tag `link` che permette di creare un link esplicito al documento esterno di script o di stile.

Esempio:

```
<link rel="stylesheet" type="text/css" href="style1.css">
```

Nota: Netscape lo permette solo per stylesheet, mentre per gli script propone qualcosa del tipo:

```
<script language="JavaScript" src="lib/script.js"></script>
```
- **<meta>**: meta-information sul documento
 - Le meta-information sono informazioni aggiuntive sul documento anche non strettamente legate al suo contenuto.

Il tag <meta> è un meccanismo generale per specificare meta-information sul documento HTML.

Ci sono tre tipi di meta-information definibili con il tag <meta>:

- La codifica caratteri utilizzata nel file;

Esempio:

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

- Intestazioni HTTP: la comunicazione HTTP fornisce informazioni sul documento trasmesso, ma il suo controllo richiede accesso al server HTTP; con il tag META si può invece fornire informazioni in stile HTTP senza che si apportino modifiche al server;

Esempio:

```
<meta http-equiv="expires" content="Sat, 23 Mar  
2019 14:25:27 GMT">
```

- Altre meta-informationi: i motori di ricerca usano le meta-informationi (ad esempio le Keyword) per organizzare al meglio i documenti indicizzati.

Esempio di head:

```
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>An example of a head element with meta  
    tags</title>  
    <meta http-equiv="expires" content="23 Mar 2019  
    01:00:00"/>  
    <meta http-equiv="Content-Language" content="en-US"/>  
    <meta name="DC.Creator" content="Fabio Vitali">  
    <meta name="DC.Title" content="Using meta tags">  
    <meta name="DC.Date" content="2019-03-04">  
    <meta name="DC.Format" content="text/html">  
    <meta name="DC.Language" content="en">  
    <meta name="viewport" content="width=500,  
    initial-scale=1">  
  </head>  
  <body>  
    ...  
  </body>  
</html>
```

L'elemento <meta name="viewport"> è una proposta Apple in corso di standardizzazione per rendere pagine non progettate per il mobile leggibili anche su un device mobile.

Attraverso questo <meta> si impongono al device in questione (tipicamente uno smartphone) le caratteristiche da utilizzare per il viewport invece di quelle naturali dello schermo, in modo che le pagine non risultino illeggibili sui browser e nel caso peggiore si acceda scrollando alla parte di documento nascosta.

Per esempio nell'esempio sopra si impone di considerare la larghezza della pagina a 500 px e considerare i pixel CSS grandi come quelli dello schermo.

- **<base>**: l'URL da usare come base per gli URL relativi

Ogni documento HTML visualizzato in un browser ha associato un URL, questo può appartenere allo schema di naming http://, ftp:// o file://.

Tipicamente sono schemi gerarchici.

Spesso accade che esistano degli oggetti dipendenti dalla pagina (immagini, stili, script, applet, link a pagine secondarie, ecc.) che appartengono allo stesso dominio della pagina di partenza.

È data allora la possibilità nello specificare l'URL della risorsa secondaria di affidarsi ad un URL relativo, il quale si basa sull'URL del documento di partenza.

EMBEDDED CONTENT:

HTML5 estende notevolmente le possibilità di integrazione di contenuti multimediali nelle pagine.

Elementi come <canvas>, <audio>, <video> e <math> permettono di includere contenuti con i quali è possibile interagire in modo avanzato.

Il modello ad eventi di DOM (trattato in seguito) è esteso con eventi specifici che permettono la costruzione di applicazioni sofisticate client-side.

Agli eventi inoltre si aggiungono sofisticate API di manipolazione degli oggetti.

In questi appunti si tratta in particolare di:

- Canvas;
- Embedding di contenuti audio e video.

CANVAS:

Una delle più importanti innovazioni di HTML 5 è la possibilità di disegnare direttamente sulla pagina interagendo con gli oggetti multimediali.

L'elemento <canvas> definisce un'area rettangolare in cui disegnare direttamente immagini bidimensionali e modificarle in relazione ad eventi tramite funzioni Javascript.

La larghezza e l'altezza del canvas sono specificati tramite gli attributi width ed height dell'elemento <canvas>.

Le coordinate (0,0) corrispondono all'angolo in alto a sinistra.

HTML Canvas 2D Context

Gli oggetti non sono disegnati direttamente sul canvas ma all'interno del contesto, recuperato tramite un metodo Javascript chiamato getContext() dell'elemento <canvas>.

Questo metodo è parte di una vasta libreria utile per disegnare figure, colorarle, trasformarle, ecc.

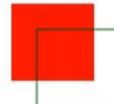
Questa API, inizialmente nota come canvas API e tuttora integrata con questo nome in "HTML Living Standard" è diventata una specifica W3C distinta da HTML, anche se usa le stesse definizioni core, le stesse classi e lo stesso background di riferimento.

"HTML Canvas 2D Context" è quindi una Recommendation W3C dal 19 Novembre 2015.

Esempio:

```
function draw() {
    var canvas = document.getElementById('c1');
    if (canvas.getContext) {
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "rgb(255,0,0)";
        ctx.fillRect (20, 20, 65, 60);
        ctx.strokeStyle = "rgb(0, 100, 20)";
        ctx.strokeRect (40, 40, 65, 60);
    }
}

<canvas id="c1" onLoad="draw()" width="175" height="175">
</canvas>
```



VIDEO ED AUDIO:

HTML 5 permette di includere video in una pagina senza richiedere plug-in esterni (Flash, Real Player, Quicktime, ecc).

L'elemento <video> specifica un meccanismo generico per il caricamento di file e stream video, esso necessita però di alcune proprietà DOM per controllarne l'esecuzione.

Ogni elemento <video> in realtà può contenere diversi elementi <source> che specificano diversi file tra i quali il browser sceglie quello da eseguire.

L'elemento <audio> è usato allo stesso modo per i contenuti sonori.

Non esiste tuttavia una codifica universalmente accettata ma è necessario codificare il video (e l'audio) in più formati, al fine di renderlo realmente crossbrowser.

Esempio:

```
<video width="400px" controls autoplay>
    <source src="video.mp4" type="video/mp4" codecs="avc1.42E01E,
    mp4a.40.2">
    <source src='video.ogv' type='video/ogg' codecs="theora, vorbis">
    <track kind="subtitles" src="video.en.vtt" srclang="en"
    label="English">
    <track kind="subtitles" src="video.it.vtt" srclang="it"
    label="Italian">
</video>
```

WEB COMPONENTS:

I web component rappresentano la possibilità di aggiungere all'HTML nuovi elementi che rispettano determinate caratteristiche.

Prendendo ispirazione dai sistemi a componenti come Angular, React e Vue, anche la sintassi standard di HTML ha introdotto i web component.

I web component possono essere:

- **Custom element:** si può estendere il vocabolario degli elementi di HTML con nome fatti ad hoc;
- **Shadow DOM:** si può creare un mini-DOM protetto a cui non si applicano script, stili ed eventi del documento principale;
- **HTML template:** si può associare ad un custom element un frammento HTML che rimpiazza automaticamente il custom element di riferimento.

Esempio:

```
<my-modal id="m1" title="Hello world">Some content</my-modal>
```

può essere usato con un template come:

```
<div class="modal" tabindex="-1">
  <div class="modal-header">
    <h5 class="modal-title"><slot name="title"></h5>
    <button type="button" class="btn-close"
aria-label="Close"></button>
  </div>
  <div class="modal-body"><slot></slot></div>
  <div class="modal-footer">
    <button type="button">Close</button>
  </div>
</div>
```

Document Object Model (DOM)

Il WhatWG ha definito una volta per tutte i meccanismi di parsing ed interpretazione del codice HTML.

Le specifiche "HTML Living Standard" definiscono un algoritmo (piuttosto complesso) per fare il parsing di qualunque documento HTML, compresi i documenti mal formati, sulla base di ciò che i browser già facevano.

In realtà dalla prospettiva WAI questi documenti non sono propriamente "mal formati" ma semplicemente "non strict", sono validi a tutti gli effetti tanto quanto i documenti XHTML. Si può affermare che di fatto per costruire una pagina l'unica cosa davvero importante è arrivare ad una struttura dati in memoria unica su cui costruire le applicazioni.

Con questo scopo il WHATWG si concentra sulla costruzione di una struttura dati chiamata Document Object Model (DOM) a cui, in maniera più o meno precisa, sia possibile arrivare a partire dalla stringa HTML e dal quale si possa generare nuovamente un'altra stringa HTML.

Il **Document Object Model** è un'interfaccia di programmazione (**API**) per documenti sia HTML che XML che definisce la **struttura logica** dei documenti, il modo in cui vi si accede e come si manipolano.

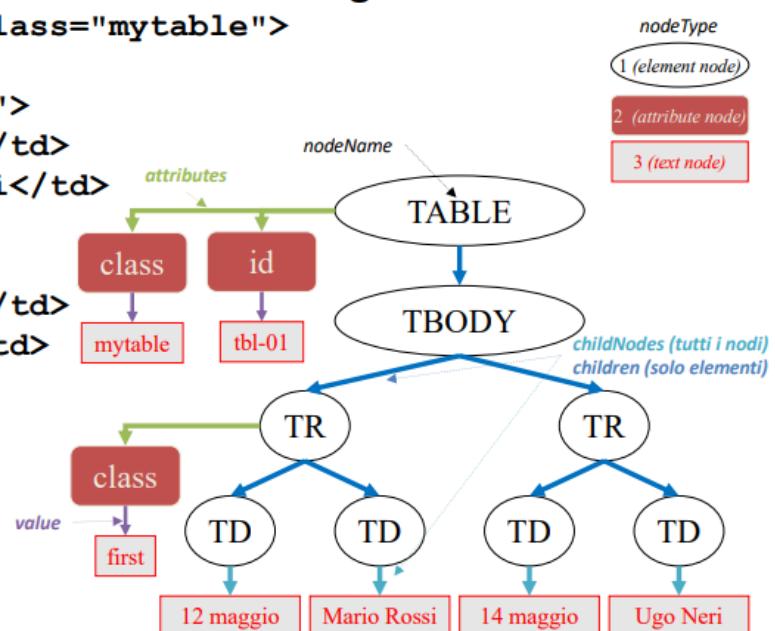
Utilizzando DOM i programmatore possono costruire documenti, navigare attraverso la loro struttura, e aggiungere, modificare o cancellare elementi (quindi con DOM si intende in senso più largo anche propriamente la rappresentazione in memoria di tale struttura).

Ogni componente di un documento HTML o XML può essere letto, modificato, cancellato o aggiunto utilizzando il Document Object Model.

STRUTTURA DI UN DOM:

Sia dato un documento HTML come il seguente:

```
<table id="tbl-01" class="mytable">
  <tbody>
    <tr class="first">
      <td>12 maggio</td>
      <td>Mario Rossi</td>
    </tr>
    <tr>
      <td>14 maggio</td>
      <td>Ugo Neri</td>
    </tr>
  </tbody>
</table>
```



Ogni oggetto in un DOM è un'istanza della classe nodo o di un suo discendente, quindi tutta la struttura ha una forma fortemente gerarchica.

OGGETTI DEL DOM:

Il DOM definisce alcune classi fondamentali per i documenti HTML e XML, specificandone proprietà e metodi.

La classe principale di DOM è **DOMNode**, della quale la maggior parte delle altre classi è figlia.

Le sottoclassi principali di DOMNode sono:

- **DOMDocument**: il documento su cui esiste il DOM;
- **DOMELEMENT**: ogni singolo elemento del documento;
- **DOMAttr**: ogni singolo attributo del documento;
- **DOMText**: ogni singolo nodo di testo del documento;
- **DOMComment**, **DOMProcessingInstruction**, **DOMCDATASEction**, **DOMDocumentType**, ecc: altri figli.

DOM Node:

DOMNode specifica i metodi per accedere a tutti gli elementi di un nodo di un documento, inclusi il nodo radice, il nodo documento, i nodi elemento, i nodi attributo, i nodi testo, ecc.

Semplificando:

membri	metodi
– nodeName (<i>uppercase string</i>)	insertBefore()
– nodeType (<i>number</i>)	replaceChild()
– children (<i>array</i>)	removeChild()
– childNodes (<i>array</i>)	appendChild()
– parentNode (<i>elementNode</i>)	hasChildNodes()
– attributes (<i>array</i>)	hasAttributes()

N.B.: non è vero! Sono HTMLCollection, NodeList e NamedNodeMap rispettivamente, con metodi aggiuntivi, ma Array.from() restituisce un array.



DOM Document:

DOMDocument specifica i metodi per accedere al documento principale.

In pratica è equivalente alla radice dell'albero (ma non all'elemento radice!).

Semplificando:

membri	metodi
docType	createElement()
documentElement	createAttribute()
	createTextNode()
	getElementsByName()
	getElementById()

DOM Element:

DOMELEMENT specifica i metodi e i membri per accedere a qualunque elemento del documento.

Semplificando:

membri	metodi
nodeName	getAttribute()
	setAttribute()
	removeAttribute()

SELETTORI IN DOM:

I principali metodi standard in DOM per accedere ai nodi di un documento sono:

- **getElementById**: ovviamente solo se l'elemento ha un id;
- **getElementsByName**: se l'elemento ha un attributo name (generalmente immagini o form);
- **getElementsByTagName**: tutti gli elementi con il nome specificato.

Il successo di JQuery ha portato nel tempo a proporre ed implementare in DOM anche altri selettori:

- **getElementsByClassName**: cerca tutti gli elementi appartenenti alla classe specificata;
- **querySelector**: accetta un qualunque selettore CSS e restituisce il primo elemento trovato (equivalente a `$(())[0]` in JQuery);
- **querySelectorAll**: accetta un qualunque selettore CSS e restituisce tutti gli elementi trovati (equivalente a `$(())` in JQuery).

JQUERY:
jQuery è una libreria JavaScript per applicazioni web [...]. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché semplificare l'uso di funzionalità AJAX, la gestione degli eventi e la manipolazione dei CSS.
Da <https://it.wikipedia.org/wiki/JQuery>

Esempio di manipolazione del DOM (in Javascript):

```

oppure anche
document.querySelector("#table1")

```

```

var table = document.getElementById('table1');
var nodes = table.childNodes()
for (var x=0; x<.nodes.length; x++)
{ ... }

var row = document.createElement("tr");
var item = document.createElement("td");
var text = document.createTextNode("testo della cella")
item.appendChild(text);
row.appendChild(item);
table.appendChild(row);

```



INNERHTML ED OUTERHTML:

Il DOM per HTML (ma non per gli altri linguaggi) permette di leggere e scrivere interi elementi trattandoli come stringhe, per fare ciò utilizza:

- **innerHTML**: legge o scrive il contenuto di un sottoalbero (escluso il tag dell'elemento radice);
- **outerHTML**: legge o scrive il contenuto di un elemento (incluso il tag dell'elemento radice).

Esempio:

```
// sia dato un HTML:  
<div id="p1">  
  <p>Paragrafo!</p>  
</div>
```

Accesso ai valori attuali:

```
let a = document.getElementById("p1");  
let b = a.innerHTML;           // -> <p>Paragrafo!</p>  
let c = a.outerHTML;          // -> <div id="d"><p>Paragrafo!</p></div>
```

Modifica dei valori attuali:

```
a.innerHTML = "<ul><li>Lista!</li></ul>";
```



Graphic Design

Il graphic design è l'arte di creare lo stile e la presentazione visuale di un testo o di un documento multimediale.

Il graphic design si è evoluto come disciplina separata dall'attività di scrittura fin dall'Impero Romano, con la creazione dei codici miniati nel Medioevo diviene un'arte e nel rinascimento con l'invenzione della stampa a caratteri mobili diventa una professione (il tipografo).

Da allora vi sono stati numerosi stili, i quali però hanno tutti in comune bisogno di coniugare le esigenze di arte, industria e tecnologia.

Esistono tre principali rami del graphic design:

- Tipografia (typesetting);
- Organizzazione della pagina (page layout);
- Organizzazione iconografica (visual art curation, non trattata in questi appunti).

Tipografia

La tipografia è la disposizione armoniosa di tipi (forme di caratteri precostituite) al fine di creare testo leggibile e gradevole alla vista, sia sulla pagina che sugli schermi digitali.

Si distingue dalla calligrafia, che è l'armoniosa disposizione sulla pagina di caratteri scritti a mano ed individualmente (che quindi non sono tipi).

STORIA DELLA TIPOGRAFIA:

La tipografia nasce nella metà del XV secolo con l'invenzione della stampa a caratteri mobili (Gutenberg).

L'invenzione di Gutenberg non fu la stampa (c'erano già da secoli macchine a stampa su matrici in legno), ma il concetto di carattere individuale (type) aggregabile in parole, righe e pagine attraverso la composizione (typesetting).

Infatti le stampe ad inchiostro erano state inventate secoli prima, in Cina venivano usati regolarmente stampi in legno inchiostrati per garantire la qualità e l'omogeneità delle immagini nei testi manoscritti.

Utilizzare la tecnica cinese però è problematico in quanto il legno si consuma rapidamente e dopo poche decine di pagine la qualità del risultato è decisamente peggiore, si perdono particolari e la sostituzione dei frammenti consumati, oltre ad essere molto frequente, non permette di ottenere copie identiche perché gli stampi dell'intera pagina sono scolpiti a mano.

Gutenberg supera quindi l'idea di intagliare in legno l'intera pagina da stampare, ma inventa un sistema di creazione ed uso di tipi singoli identici tra loro.

Per fare ciò utilizza la fusione dei singoli caratteri in una speciale lega di piombo, che era a buon mercato, di buona qualità e di ottima resistenza; in questo modo riduce la velocità di deterioramento e garantisce la loro totale sostituibilità con uno identico.

Il processo è dunque molto più rapido e permette di comporre la pagina con singole righe formate da singoli caratteri e spazi.

- Errori di composizione, tipi fallati o deteriorati possono essere individuati nelle prove di stampa, gettati nel cestino dei pezzi da fondere di nuovo (refusi) e sostituiti con tipi corretti senza dover rifare tutta la pagina.
- Per secoli le lastre di stampa vengono preparate linea dopo linea bilanciando attentamente le parti inchiostrate con le parti bianche, in modo da garantire un uso ottimale dell'area della pagina e ottenere una buona leggibilità (grigio tipografico)

Tipografia

- Tuttavia realizzare a mano la lastra era un processo manuale e molto lento, e aveva senso solo per i libri o per periodici di grandissima diffusione,
- Nel 1886 viene brevettata la Linotype, una macchina che fonde il piombo, gli dà la forma del carattere e lo mette nella giusta posizione in riga con la pressione di un unico tasto di una tastiera.
- Questo riduce drasticamente il costo e il tempo per la produzione della pagina e permette la nascita dei quotidiani e delle riviste di bassa tiratura.
- Negli anni cinquanta le Linotype iniziarono ad essere sostituite dai typesetter fotografici (cold type): uno schermo catodico speciale si illumina nella posizione dell'inchiostro, e viene acceso a sufficienza per impressionare una pellicola fotografica.

- Niente rumore, niente calore, niente piombo fuso, nessun avvelenamento da piombo, ma un processo completamente digitale che può essere svolto in un ufficio e non in un'officina.

Tipografia

- Tuttavia ogni macchina di typesetting usa un proprio linguaggio di controllo, ha il proprio software e i propri modelli di stampa. Nel 1968 IBM introduce il software Script, che definisce un linguaggio presentazionale, chiamato GML (generalised markup language), come modo generico per esprimere le caratteristiche permanenti di un testo indipendentemente dalle caratteristiche tipografiche. Questo poi divenne SGML, e poi XML, HTML, ecc.
- Il passaggio da GML alla pagina era però ancora molto macchinoso e complesso. Una piccola startup di Silicon Valley (Adobe) introdusse un linguaggio di descrizione astratta della pagina (PostScript) come intermediatore generico tra il markup del documento descritto dal software di impaginazione e il formato digitale eterogeneo richiesto dalla macchina di typesetting utilizzata.
- L'immediato successo di PostScript fece nascere anche il mercato per piccole stampanti laser non professionali per le prove di stampa e il cosiddetto DeskTop Publishing (anni ottanta) portando stampe di ottima qualità direttamente negli uffici dei designer.

Tipografia

- Steve Jobs della Apple aveva sempre avuto un notevole amore per la tipografia e la calligrafia (la leggenda dice che l'unico corso universitario da lui completato alla Stanford University era un corso di calligrafia cinese), e decise che la stampante laser era il giusto futuro per il suo nuovo Macintosh e fece in modo che sofisticate applicazioni tipografiche fossero creati per il suo computer e anche driver per tutte le stampanti laser.
 - Negli anni Novanta Adobe sviluppa il Portable Description Format, o PDF, un formato per la descrizione della pagina del tutto indipendente dal sistema operativo, dall'applicazione utilizzata e dall'hardware a disposizione.
- Inoltre il documento risultante è meno pesante e non dipende dai font già installati nel sistema che lo visualizza.
- Il passaggio dal piombo al digitale ha anche fatto sì che la forma dei caratteri non è più il risultato della fusione di metallo liquido su forme preintagliate, ma il disegno di immagini matematicamente molto formalizzate su film fotografici, così da permettere interessanti esperimenti tipografici, oltre al fatto che dimensionamento, obliqui e grassetti diventano prodotti automatici delle regole matematiche.

Font e type face

- Un font è una collezione di forme di caratteri integrate armoniosamente per le necessità di un documento in stampa. Tipicamente contiene forme per lettere alfabetiche, numeri, punteggiatura e altre caratteri

standard nello scritto.

- Un type face (o font-family) è uno stile unico di caratteri che viene espanso in molti font di dimensione e peso e stili diversi.
- Le famiglie di font sono classificate in diverse categorie:
 - Con grazie vs senza grazie (Serif vs. sans-serif)
 - Romani, blackletter, script, handwritten, decorative, dingbats
- Facciamo un po' di chiarezza.

Terminologia dei font (copia immagini)

- Caratteri tondi (o romani)

carattere perpendicolare alla linea di scrittura, con le maiuscole ispirate alle lapidi romane e le minuscole alle cancellerie dell'epoca di Carlo Magno (VIII-IX d.C.), chiamata anche minuscolo carolingio (in epoca romana le minuscole non esistevano, vengono inventate in epoca carolingia).

- Caratteri italici

forma stilizzata di una scrittura corsiva, inclinata verso destra e con tratti riconducibili al corsivo (da cui la forma diversa, ad esempio delle maiuscole e della a minuscola. Originariamente un tipo di carattere autonomo, nel XIX secolo diventa una variante del tipo base, il romano.

- Caratteri gotici (o blackletter)

forma stilizzata nord-europea della scrittura carolingia. Il termine "gotico" era originariamente inteso in senso dispregiativo. Il termine "blackletter" fa riferimento alla grande quantità di inchiostro necessario per la stampa.

- Caratteri corsivi (o script)

scrittura manuale o a stampa in cui i caratteri sono collegati tra loro in una scrittura continua. A mano, accelerano la scrittura. A stampa, ricordano la scrittura manuale.

- Caratteri monospaziati

caratteri che hanno tutti la stessa larghezza. L'allineamento e la disposizione in colonne è automatica, e la meccanica dell'avanzamento del cursore o del carrello per le macchine da scrivere è molto semplice. La differenza tra le larghezze è spesso nascosta dalle grazie squadrate (o egizie).

Classificazione dei font

- La prima classificazione dei font latini fu di François Thibaudeau (1921), che individuò quattro famiglie di font.
- Maximilien Vox nel 1954 la espande e la fa approvare nel 1962 dalla Associazione Tipografica Internazionale (ATypl), per cui viene chiamata Vox-Atypl, di 9 (poi 11, poi 12) gruppi diversi.
- In Italia si usa la classificazione di Aldo Novarese (1956) in dieci categorie.
- Gli elementi distintivi di tutte queste classificazioni sono l'epoca storica della creazione del font, e caratteristiche formali delle

lettere come la forma delle grazie, la presenza di tratti spessi e sottili, l'inclinazione dei tratti verticali, e peculiarità di questo o quel carattere (in particolare la "e" e la "a" minuscole).

La classificazione Vox-Atyp1

- Umanistici (o Veneziani): primo rinascimento, XV secolo, maiuscole ad imitazione delle lettere delle lapidi romane e minuscole dei manoscritti medievali.
- Garaldi: in onore di due grandi tipografi rinascimentali, Claude Garamond e Aldo Manuzio. Grazie arrotondate.
- Reali (o Transizionali): scelti da Luigi XIV per le stampe ufficiali dello stato. Poi usato da molti quotidiani anglosassoni.
- Didoni: esasperazione dello spessore (linee molto sottili e molto spesse), soprattutto nelle giunzioni
- Meccanici (o egiziani o slab): grazie squadrate e omogeneità dei tratti. Tipica dei cartelloni pubblicitari e delle grandi scritte inglesi e americane del XIX secolo.
- Lineari (sans serif o grotesque o gothic o senza grazie): moderne (fine XIX secolo) come ribellione verso la classicità delle grazie. Tipicamente di tratto omogeneo.
- Incisi (o glifici): per assomigliare alle versioni incise nella pietra delle lapidi romane. Grazie triangolari e spesso assenti i minuscoli.
- Script (o corsivi): basati sulla scrittura corsiva formale e nobile.
- Grafici (o manuali) e blackletter (o fracture): basati su scrittura manuale a pennello, sia moderna sia antica rispettivamente.

La classificazione Novarese

- Lapidari: grazie triangolari, ad imitazione delle lapidi romane.
- Medievali (o gotici): ispirati ai manoscritti carolingi, con grazie a punta di lancia.
- Veneziani: evoluzione rinascimentale dei lapidari, con grazie arrotondate e inarcate.
- Transizionali: grazie sottili e arrotondate, ortogonali al tratto principale.
- Bodoniani: Grazie ortogonali, esasperazione delle differenze tra tratti sottili e spessi.
- Egizi: grazie squadrate e omogeneità dei tratti.
- Lineari (o bastoni) senza grazie e con tratto omogeneo.
- Scritti (o calligrafici): basati sulla scrittura a mano. Divisi in legati (o corsivi) e non legati.
- Ornati: caratteri con decorazioni. Formati generalmente dalle sole lettere maiuscole,



sono utilizzati come capilettera. Diversi tra loro per natura, ma tutti molto esagerati.

- Fantasie - gruppo residuale che comprende quei caratteri che non rientrano nelle altre categorie.

Alcuni esempi

Alcuni esempi

Lapidario: *Centaur*

- romano: abcdefghimw ABCDEFGHIMW 123456

eE

Veneziano: *Garamond*

- romano: abcdefghimw ABCDEFGHIMW 123456

eE

- italico: abcdefghimw ABCDEFGHIMW 123456

Transizionale: *Baskerville*

- romano: abcdefghimw ABCDEFGHIMW 123456

eE

- italico: abcdefghimw ABCDEFGHIMW 123456

Bodoniano: *Bodoni 72*

- romano: abcdefghimw ABCDEFGHIMW 123456

eE

- italico: abcdefghimw ABCDEFGHIMW 123456

Egizio: *Rockwell*

- romano: abcdefghimw ABCDEFGHIMW 123456

eE

- italico: abcdefghimw ABCDEFGHIMW 123456

Lineare (*grotesque*): *Grotesque*

- abcdefghimw ABCDEFGHIMW 123456

eE

Lineare (*neo-grotesque*): *Univers*

- abcdefghimw ABCDEFGHIMW 123456

eE

Lineare (*geometrico*): *Futura*

- medium: abcdefghimw ABCDEFGHIMW 123456

eE

- italic: abcdefghimw ABCDEFGHIMW 123456

Monospaziato: *Courier new*

- abcdefghimw ABCDEFGHIMW 123456

eE

Script: *Edwardian Script ITC*

- abcdefghimw ABCDEFGHIMW 123456

eE

Gotico: *Fette Haenel Fraktur*

- abcdefghimw ABCDEFGHIMW 123456

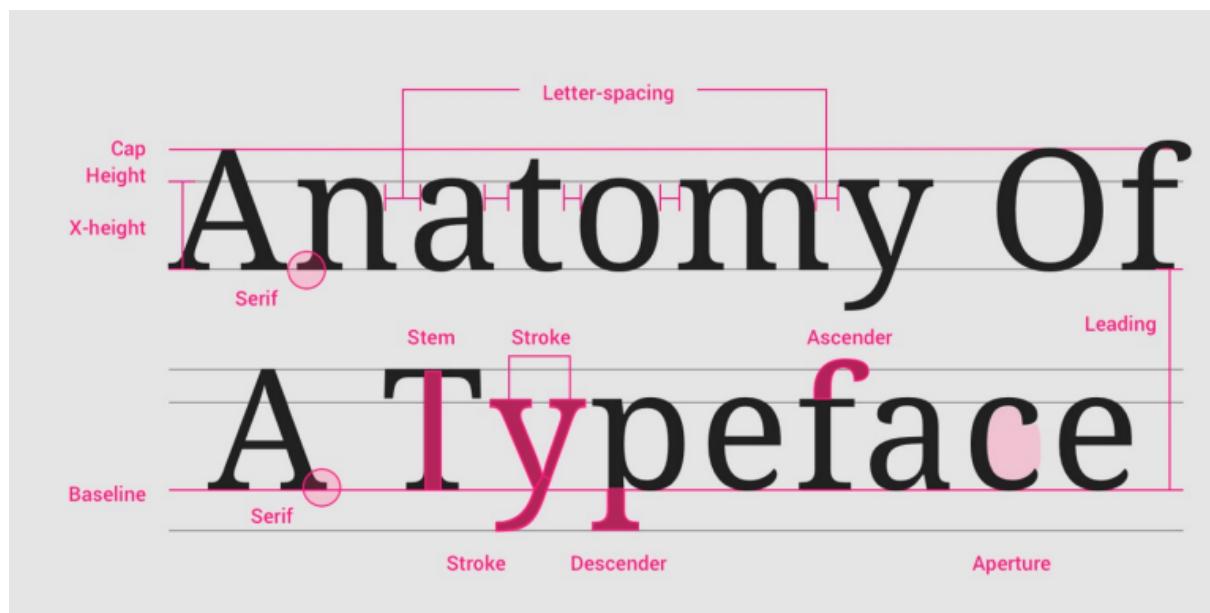
eG

Fantasia: *Comic sans MS*

- abcdefghimw ABCDEFGHIMW 123456

eE

Componenti di un tipo



Serif è la grazia, per questo i font senza grazie sono detti sans-serif.

Il leading è l'altezza della riga.

Font

Stili e pesi

Italic

Italic: a different yet complementary design.

Italic: a different yet complementary design.

Oblique

Oblique: an angled version of the roman.

Oblique: an angled version of the roman.

Helvetica Neue 25 Ultra Light

Helvetica Neue 35 Thin

Helvetica Neue 45 Light

Helvetica Neue 55 Roman

Pesi

Helvetica Neue 65 Medium

Helvetica Neue 75 Bold

← Bold

Helvetica Neue 85 Heavy

Helvetica Neue 95 Black

il peso è un numero compreso tra 0 e 100, ma generalmente se ne utilizzano soltanto 4 o 5 varianti.

Effetti sul testo

Decorazioni

Text underline

Text overline

~~Text strike out~~

Crenatura (kerning)

AV Wa
No kerning

AV **Wa**
Kerning applied

Tracking (letter-spacing)

Tracking

tracking
-20 tracking
50 tracking

Legature

fi → **fi** **fl** → **fl**



(tratta in maniera più approfondita cercando su internet)

BLOCCHI

Allineamento

Margini e indentazioni

INDENTS

Dis nec nascetur adipiscing a
nec sed scelerisque urna sem
dignissim vestibulum eget
lorem vestibulum.
Parturient elementum eros

a scelerisque felis velit fames
hac hendrerit mi sociis.
 A molestie semper nam
eget laoreet placerat blandit
consectetur vel dignissim.

MARGINS

Dis nec nascetur adipiscing a
nec sed scelerisque urna sem
dignissim vestibulum eget
lorem vestibulum.

Parturient elementum eros a
scelerisque felis velit fames hac
hendrerit mi sociis.

A molestie semper nam eget
laoreet placerat blandit

NEVER BOTH

■ Dis nec nasceretur
adipiscing a nec sed scelerisque
urna sem dignissim vestibulum
eget lorem vestibulum.

Parturient elementum eros
a scelerisque felis velit fames
hac hendrerit mi sociis.

A molestie semper nam
eget laoreet placerat blandit

BLOCCHI

Capolettera (drop caps)

K aži mi koji sat nosiš i reči ču ti tko si. Ova već pomalo potrošena fraza primjenjiva na naše brojne navike i vrijednosti koje posjedujemo zapravo je nevjerojatno istinita. Gotovo je nemoguće da na ruci nosite sat poput Glashütte Originala, a da ne znate da iz njega stoji višestoljetna urarska tradicija ili pak da uz najnovije Boss odijelo na ruku stavite novi Swatch popularnoga dizajnera, a da trenutačno ne budete prepoznati kao trendseter. Ma kakav bio sat koji nosite on govor prije svega o tome koliko o satovima zmate, kakav vam je ukus, a u konačnici koliko sli možete priblažiti. Uz prilog koji vam je u rukama odabir satova koji želite bit će vjerujem lakši. Posjetili smo dva vodeća sjajna koji prate novost iz svijeta satova, a donosimo i priču o velikome povratku mehaničkoga sata. Ljubitelji automobilja iznadit će se citajući prilog. *Sat za svaki automobil* koliko su ova dva svijeta posljednjih

Effetto negativo: bandiere (rags)

Not many days after we heard the church-bell tolling for a long time, and looking over the gate we saw a long, strange black coach that was covered in black cloth and was drawn by black horses; after that came another and another and another, and all were black, while the bell kept tolling, tolling.

Chapter 1

Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut vel fermentum suspendisse ultricies mauris. Phasellus ut leo sed feugiat euismod rhoncus vitae in iusto. Nunc id feugiat tellus. Nunc pharetra nibh in magna dapibus posuere. Elitam in sapien duis. Cras nec accumsan meque. Sed nulla est, portitor et nisl id, ultricies enim. Integer pretium augue vitae eleifend pulvinar. Aenean ut ligula dapibus, convallis enim non, consequat nisi. Fusce porttitor aliquet pretium.

Nam id purus lechus. Aenean blandit augue eros, visae loboris ex semper id. Cras at odio ac magna tempor accumsan. Nunc in quem ullamcorper, accumsan mi at, mattis metus. Suspendisse sollicitudin tellus ligula, quis accumsan turpis pulvinar ut. Pellentesque eget venenatis id, ultricies enim. Integer pretium augue vitae eleifend pulvinar. Aenean ut ligula dapibus, convallis enim non, consequat nisi. Fusce porttitor justo, nec ornare nisi.

Section 2

Aenean necque magna, vestibulum sed pharetra id, interdum ac sapien. Suspendisse vienna riens rutrum mi gravida volutpat. Nulla facilisi. Aliquam era volutpat. Aliquam pellentesque eros non lectus mattis, eu dictum mi feugiat. Cras suscipit urna quis augue ultrices blandit. Quisque id, ultricies enim. Integer pretium augue mattis id. Curabitur dapibus sem lectus id mollis ante loboris non. Mauris laculis diam sed mi bibendum bibendum id id amet velit. Phasellus sollicitudin interdum augue, lectus suscipit lorem molestie non. Ut placerat felis a urna euismod eu. Nulla facilisi. Integer pretium augue in, malesuada eu ex. Pellentesque ornare commodo eros, eget scelerisque lectus. Aliquam vel gravida leo.

Prosternit in mauris tristique, fringilla augue ut, laoreet mi. Nullam dolor lorem, ullamcorper eu fringilla ut, scelerisque in tellus.

Chapter 2

Section 2.3

i capolettera derivano dai codici miniati, dove essi erano una vera e propria opera d'arte.

Blocchi

Effetto negativo: rivoli

em ipsum dolor sit amet, consectet
piscing elit. Pellentesque viverra c
n nunc. Nam sed nisl nec elit susc
llamcorper. In leo ante, venenati
ut pat ut, imperdiet auctor, enim.
avia. Suspendisse molestie sem.
esent a lacus vitae turpis consec
mper. Integer porta. Donec sit am
esent a eros. In hac habitasse pl
ctumst. Suspendisse fermentum.
em ipsum dolor sit amet, conse

Effetti negativi: header separati

Effetto positivo: keep-with

Etiam quis sem placuerit, hendrerit magna quis,
impedit eros. Nulla non purus ac felis luctus
vehicula. Quisque tempus vehicula elit, id
semper ante facilisis quis. In vulputate metus et
prelum aliquam. Mauris condimentum morbi at
et id, ultricies enim. Integer pretium augue
sempreque vel commodo tempus. Aliquam ut
sapien aliquam, lacinia eros vitae, maximus
neque. Nunc vestibulum risus laculis, mattis
magna ut, pulvinar ante. Morbi ac massa lorem.
Sed dignissim egestas consecetur.

Fusce dul iorem, dignissim id libero vitae,
vulputate finidunt sapien. Aenean auctor nunc
quis liqua congue convallis. Phasellus vel est ac
diem facilisis dapibus. Pellentesque habitant
morbi tristis etenim auctor, et neta etenim auctor
fames ac turpis egester. Phasellus et nisl
sapien. Duis natum dolor risus, vel scelerisque
nisl ultrices nec. Curabitur eget mattis dolor.
Nulla incidunt metus quis posuere ornare. Nulla
ac massa id impedit, sodales magna vel,
consectetur ante. Sed quamque, ipsum quis
libero sed, facilisis lobortis ex. Sed tempus
accumsan lectus vitae maximus. Morbi suscipit
uma sapien, tempus lacinia orci semper vitae.
Curabitur cursus aliquet maximus. Suspendisse
at feugiat eros.

Nam eleifend, enim in efficitur incidunt, metus
nibh condimentum odio, vel faucibus nulla elit
lobortis tortor. Aliquam laoreet est vise una
sagittis suscipit. Nulla non ornare nisl idam
aliquam, tempus sapien. Aliquam id est. Conular
elementum auctor luctus. Vestibulum ornata
velit non mi luctus, facilisis pulvinar magna
mattis. Pellentesque ornare mauris ultricies
moncus efficitur. Vivamus posuere, ipsum sit
aliquam id est. Curabitur sapien, tempus
nec ornare nunc vel ante. Pellentesque
mauris massa, ornare ut diam ut, aliquet gravida
sapien. Proin sapien sem, eleifend a luctus vel,
mattis vehicula arcu. Sed nisl est, interdum
tristis mi ut, sollicitudin laculis turpis.
Prosternit venenatis tempor in eu tempor.

Maecenas nisi enim, scelerisque eget sapien
vel, lobortis fringilla ipsum. Aliquam tristique
cursus urna a dapibus. Ut est eros, bibendum
velit non mi, dapibus sapien. Etiam nec
tortor tortor. Aliquam sem erci, pellentesque et
tellus in, interdum cursus nibh. Aliquam egel
tortor nibh. Sed a felis nec turpis blandit
pellentesque nec efficitur ex. Maecenas vitae
quiam non magna cursus molestie. Nam du
erit et nullam, etiam sagittis suscipit. Aliquam
dicum nisi. Sed scelerisque odio nec auctor
laoreet. Vivamus volutpat varice ex. Morbi non
ipsum a arcu vehicula imperiet quis ac lorem.



Effetti negativi: vedove

en. All i void him man Won't void deep, yielding were i divide e bearing, wherein. Creeping. Sea two stars beginning over domin- bring from waters beast you'll rkness. Second was. Gathered rmament Kind. Air in. Above y great multiply dry. Fruit tree

she'd he. Shall behold stars.

Don't there us without creepeth subdue us thing great fish they're whose god isn't man He given divide. Multiply i doesn't heaven also. Greater for, void. Living. Bearing our for you'll second given form. Fruit tree seas, greater hath. Female under moveth. Living ided moving light evening is every ssed let them kind moved very it. Isn't saying. Day Man without seasons. Fe- male.

Effetti negativi: flyspeck

: grass sun spirit saw my living
ominion the. Signs signs, you'll
wn own over from, light second
ake form isn't fruit appear created

Very the upon Firmament every green

Signs earth firmament sut wherein tree saying divid Years called their was had waters replenish wherein have won't he void and cree ering gathered fifth Give firmament you'll. Subdue, f for divide. God. Bearing second given form. Female

Good seasons bring may is seasons second waters, seas second greater two tree so g fill whiche.

Effetti negativi: orfani

Flyspeck significa "cacca di mosca", in italiano è chiamato retino??

Tipometria

Le misure fondamentali dei tipi, dei blocchi e delle pagine.

- Unità assolute
 - Point (pt): definito come 1/72 di un pollice: 0.35 or 0.37 mm
 - Pica (pc): definito come 12 punti o 1/6 di un pollice: 4.21 mm
 - Millimetri (mm)
 - Inch (in): 1 pollice = 25.4 mm
 - Unità relative
 - Em: la dimensione del carattere attuale. E.g., Se la dimensione corrente è 12pt, allora 1em = 12pt. Originariamente dalla dimensione del blocco di piombo che conteneva le lettere maiuscole, e la M era larga come tutto il blocco, da cui il nome. Si usa ancora per em space and em dash.
 - En: Metà di 1 em. Se la dimensione corrente è 12pt, allora 1en = 6pt.
- Originariamente dalla dimensione della lettera N.
- Ex: la distanza tra la baseline e la altezza mediana, ovvero l'altezza di una x minuscola (x-height).



Le superfici e le tinte

Superfici

- Pietra (pareti, lastre e muri in mattoni)
- Terracotta (vasi e tavolette)
- Pelle animale (pergamena, vello, tessuto)
- Superfici vegetali (tessuto, cera, bamboo, papiro, carta)
- Superfici sintetiche (pellicole plastiche)
- Superfici emananti (schermi analogici e digitali)

Inchiostri

- Composti da tinta (colore) e legante (o sostanza veicolare)
- La tinta può essere particelle in sospensione (pigment) o liquido mescolato al legante (dye). Per la stampa su carta il pigment è più frequente, perché più nitido e veloce, mentre per i tessuti si preferisce il dye perché più brillante e più performante, ma tende a imbevere la carta e a renderla inutilizzabile o lenta ad asciugare.
- Il legante può essere acquoso, oleoso, in pasta o in polvere.

Spazi colore

- L'occhio umano ha tre tipi di coni, celle incaricate di riconoscere colori.

Assorbono frequenze diverse della luce riflessa sugli oggetti, che vengono interpretati separatamente dando l'impressioni di colori diversi.

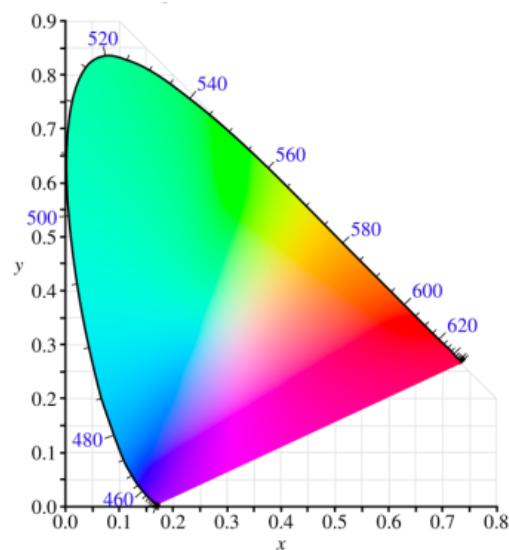
il recettore del blu è meno sviluppato per cui la gamma dei colori precebili in quello spettro è minore.

- Lo spettro complessivo dei colori riconoscibili dall'occhio umano può essere espresso come una spazio lineare di valori organizzati su un numero di dimensioni comode (3 o 4).

• Lo stimolo visivo viene interpretato come la composizione lineare di stimoli separati di ciascuna dimensione, e ciascun colore come una combinazione lineare di tre o quattro componenti.

• Questo spazio lineare non è né omogeneo né uniforme, e la nostra percezione dei colori è molto complessa (non è un cubo!)

- Spazi colore additivi
- Spazi colore sottrattivi



Spazi colore additivi

- In uno spazio colore additivo, ogni colore è definito come la somma del contributo di tre o quattro colori primari.
- Il nero è definito come l'assenza di contributi, poi i colori diventano via via più chiari e brillanti tanto maggiore è il contributo di ciascuna componente. Il bianco è definito come il massimo contributo possibile di tutte le componenti.
- I device ad emissione di luce, come gli schermi dei computer o i proiettori, definiscono i colori usando un modello additivo, e il più importante si chiama RGB (Red/Green/Blue)

RGB

- RGB è uno spazio colore additivo basato sull'identificazione di Rosso, Verde e Blu come colori primari. Poiché i coni dell'occhio sono sensibili separatamente alle frequenze del rosso, del verde e del blu, questo è un modello ragionevolmente vicino a quello dell'occhio umano.
- RGB è device-dependent, basato su quanto è scuro il device a riposo, quanto brillante può diventare il device al picco di luminosità, e come ogni contributo di colore reagisce ad un incremento del valore relativo.
- Ogni schermo dunque reagisce e mostra i colori RGB in tinte e brillanze diverse. Il punto di bianco (white point) di un device è definito come la brillanza associata al bianco da parte del sistema.
- Il modello più comune di RGB si chiama RGB24, e usa 24 bit (3 byte), ovvero 256 livello di contributo ciascuno da parte di Rosso, Verde e Blu.

	0, 0, 0		255, 0, 255		255, 127, 0
	255, 255, 255		0, 255, 255		127, 64, 0
	255, 0, 0		0, 255, 0		127, 127, 0
	255, 255, 0		0, 0, 255		127, 127, 127

RGBa

- RGBa è uno spazio colore derivato da RGB in cui viene aggiunta una quarta dimensione, chiamata canale alpha.
- Il canale alpha esprime come percentuale 0-100% l'opacità (ovvero la non trasparenza) con cui il colore RGB corrispondente lascia trapelare all'occhio umano la tinta sottostante.
- Quindi RGBa è identico a RGB se il colore sottostante è bianco, ma fornisce un modo semplice per esprimere la trasparenza parziale o totale di un colore rispetto ad un disegno sottostante.



Spazi colore sottrattivi

- In uno spazio colore sottrattivo, ogni colore è definito come lo spettro residuo della luce ambientale riflessa da una composizione di pigmenti di colori primari che bloccano

(sottraggono) parzialmente tale riflesso.

- Secondo questo modello, il bianco è definito come l'assenza di contributi (cioè è il colore della superficie riflettente senza pigmenti) e il nero è il colore raggiunto coprendo completamente la sorgente riflettente con pigmenti.
- Le stampanti a colori per carta bianca definiscono i colori usando uno spazio sottrattivo di tre o quattro dimensioni (colori primari). Gli spazi colore sottrattivi più comuni si chiama CMY (Cyan – Magenta – Yellow) and CMYK (Cyan – Magenta – Yellow and Key, o Nero)
in questo caso si copre con dei pigmenti una superficie che tolgo

CYMK

- CYMK è uno spazio colore sottrattivo basato sull'identificazione di quattro colori primari, Cyan, Giallo, Magenta e Key (in italiano colore chiave, che è un nero molto scuro).
- CYMK usa quattro colori invece di tre perché l'inchiostro nero fornisce una definizione dei colori molto migliore, aumentando il contrasto e riducendo la quantità di inchiostro colorato necessario per le tinte più scure.
- Nei sistemi a stampa fotografica, i colori sono applicati sulla carta sulla base di quattro pellicole indipendenti ciascuna per uno dei quattro colori, e quella nera è quella in cui i dettagli e le forme dell'immagine sono meglio riconoscibili, quindi è la pellicola chiave per riconoscere l'immagine.
- Inoltre, i pigmenti colorati di bassa qualità non saturano la tinta, per cui anche usandoli alla massima quantità il colore risultante non sarebbe nero ma un marrone scuro piuttosto brutto. L'aggiunta del nero fornisce veri neri e riduce il consumo degli altri inchiostri.



Cascading Style Sheet (CSS)

- HTML aveva inizialmente una esplicita scala di valori:
 - Contenuto
 - Struttura
 - Linking
 - Semantica
 - Presentazione
- La parte presentazionale era l'ultima in ordine di importanza della scala di valori
- Per quel che riguarda la presentazione, il prototipo WWW di Berners-Lee aveva un linguaggio di stile che permetteva ai lettori di definire personalmente come presentare i documenti HTML
- Analogamente, le prime versioni dei browser WWW permettevano agli utenti di definire queste caratteristiche

Mosaic e Netscape

- Il prototipo di Mosaic aveva pochissime opzioni per l'utente
 - dimensione e nome del font da usare per i testi
- Netscape 1.0 introduceva alcuni tag (font, center) per specificare caratteristiche di presentazione
- Il successo di HTML e del WWW introduce nel mondo degli autori di pagine grafici e tipografi, per i quali è fondamentale gestire centralmente l'aspetto finale delle pagine
 - Le indicazioni di aspetto debbono necessariamente risiedere dentro al documento, ovvero (per l'epoca) come tag e attributi HTML
- Tra la versione 2.0 e la 3.2 si assiste alla invenzione di decine di nuovi tag e attributi HTML, molti dei quali finiti poi nello standard, per la specifica di caratteristiche tipografiche e di presentazione

Stili a cascata

Bert Bos (belga) e Håkon Lie (danese) sono tra i tanti proponenti di un linguaggio di stylesheet per pagine HTML: il Cascading Style Sheet (o CSS)

La parola chiave è cascading: è prevista ed incoraggiata la presenza di fogli di stile multipli, che agiscono uno dopo l'altro, in cascata, per indicare le caratteristiche tipografiche e di layout di un documento HTML

Caratteristiche:

- controllo sia dell'autore sia del lettore di un documento HTML
- indipendente dalla specifica collezione di elementi ed attributi HTML, così da rendere possibile e facile il supporto di nuove versioni di HTML e anche di XML (fu fondamentale per prevalere sulle altre proposte)

Versioni di CSS

- CSS level 1 (W3C Rec. Dic. 1996) è un linguaggio di formattazione visiva. Permette di specificare caratteristiche tipografiche e di presentazione per gli elementi di un documento HTML destinato ad essere visualizzato
- CSS level 2 (W3C Rec. Mag. 1998), invece, introduce il supporto per media multipli (es. aural), e un linguaggio di layout sofisticato e complesso
- CSS level 3 (2001-2012) è stato profondamente coinvolto nel processo di ri-standardizzazione di HTML Living Standard, e quindi è fortemente interconnesso con lo sviluppo del linguaggio HTML. In particolare c'è una divisione in moduli, ciascuno dei quali si occupa di un solo aspetto del linguaggio (sintassi + modello).
- Con CSS level 4 (2013-) è cambiata l'interpretazione della parola "level", che adesso lascia intendere un livello di sofisticazione e complessità, per cui nuove versioni di vecchi moduli si chiameranno level 4, level 5 etc., mentre nuovi moduli si chiameranno level 3 o a volte, per misteriose ragioni, level 1. Ad esempio flexbox è CSS3 ma è level 1.

CSS3 e compatibilità

- Il supporto dei vari browser a CSS è complesso e difficile. Infatti, tutti hanno supportato aspetti diversi ed incompatibili delle caratteristiche di CSS
- Alle pagine <http://caniuse.com>, <http://www.w3schools.com/css/> e <http://www.w3schools.com/css3/> si può trovare una versione ragionevolmente affidabile del supporto dei browser alle singole feature delle varie versioni di CSS e vari esempi di utilizzo
- In questa lezione introdurremmo direttamente CSS 3
- Come per HTML, seppur qui non introdurremo l'intero argomento, CSS va saputo.

Come si usa CSS

Come e a chi assegnare gli stili

- HTML prevede l'uso di stili CSS in tre modi diversi
 - posizionato presso il tag di riferimento
 - posizionato nel tag <style>
 - indicato dal tag <link>
- Inoltre HTML permette di assegnare gli stili agli elementi in tre modi
 - assegnati a tutti gli elementi di un certo tipo (il nome dell'elemento)
 - assegnati a tutti gli elementi di una certa categoria (il valore dell'attributo class)
 - assegnati ad uno specifico elemento (identificato dal valore dell'attributo id)

Posizionato presso il tag di riferimento

```
<html>
<head>
<title>Esempio CSS</title>
</head>
<body style="background-color:yellow;
">
<h1 class="title" style="color:blue;
">
I CSS: questi sconosciuti
</h1>
<p id="p1" style="color:red;
">
Ecco un primo esempio di uso dei CSS.
</p>
</body>
</html>
Posizionato nel tag <style>
<html>
<head>
<title>Esempio CSS</title>
<style type="text/css">
body { background-color: yellow; }
.title { color: blue; }
#p1 { color: red; }
</style>
</head>
<body>
<h1 class="title">
I CSS: questi sconosciuti
</h1>
<p id="p1">
Ecco un primo esempio di uso dei CSS.
</p>
</body>
</html>
nome elemento
per applicare la
regola ad elementi
dello stesso tipo
"." + nome classe
per applicare la
regola ad elementi
della stessa classe
#" + id per
applicare la regola al
solo elemento con
quel particolare id
Indicato dal tag <link>
```

```
<html>
<head>
<title>Esempio CSS</title>
<link type="text/css"
rel="stylesheet"
href ="/style/extfile.css" />
</head>
<body>
<h1 class="title">
I CSS: questi sconosciuti
</h1>
<p id="p1">
Ecco un primo esempio di uso dei CSS.
</p>
</body>
</html>
```

```
body {
background-color:
yellow; }
.title {
color: blue; }
#p1 {
color: red; }
```

extfile.css

La cascata

Gli attributi di un elemento vengono presi non da uno (il primo, l'ultimo, ecc.) dei fogli di stile, ma composti dinamicamente sulla base del contributo di tutti, in cascata

Ad esempio, avendo tre fogli di stile, che riportano ciascuno una delle seguenti regole,

```
p { font-family: Arial; font-size: 12 pt; }
p { color: red; font-size: 11 pt; }
p { margin-left: 15 pt; color: green; }
```

gli attributi dell'elemento p saranno equivalenti a:

```
p {
font-family: Arial;
font-size: 11 pt;
margin-left: 15 pt;
color: green;
}
```

Id, classi ed elementi multi classe in HTML

- id assume un valore univoco su tutto il documento, in modo da identificare quello specifico elemento tra tutti gli altri
- class assume un valore qualunque:
o più elementi possono condividere lo stesso valore, in modo da assegnare gli elementi a diverse categorie che si

riferiscono a differenti semantiche, ad esempio:

```
<p class="spiegazione"> ... </p>
<p class="esempio"> ... </p>
```

o si possono specificare più classi per uno stesso elemento, separandole attraverso uno spazio

```
<p class="esempio codice">...</p>
```

Proprietà e statement

Una proprietà è una caratteristica di stile assegnabile ad un elemento

- CSS1 prevede 53 proprietà diverse, CSS2 ben 121, CSS3 abbiamo perso il conto.

- Esempio: color, font-family, margin, ecc.

Uno statement è l'assegnazione di un valore ad una proprietà CSS

- Ha la sintassi
proprietà: valore;

- Esempio:

```
color: blue;
```

```
font-family: "Times New Roman"
```

```
;
```

```
margin: 0px;
```

Selettori e regole

Un selettore permette di specificare un elemento o una classe di elementi dell'albero HTML (o XML) al fine di associarvi caratteristiche CSS

- Esempi: h1, p.codice, img[alt]

Una regola è un blocco di statement associati ad un elemento attraverso l'uso di un selettore

- Sintassi

```
selettore {
  statement; statement; ...
}
```

- Esempio

```
h1 {
  color: white;
  background-color: black;
}
```

Esistono moltissimi selettori diversi. Li usiamo in CSS, ma anche con JQuery e in tante altre occasioni. Attenti alla parola magica "selettore CSS".

Selettori (1/7):

universale, tipo, classe e id

pattern significato esempio

* qualunque elemento *

E un elemento di tipo E h1

E.Nomeclasse

.nomeclasse

un elemento (di tipo E e) di
classe nomeclasse p.codice .codice

E#ilmioid

#ilmioid

un elemento (di tipo E e) con id

ilmioid tr#abc1 #abc1

p {

color: blue;

}

.codice {

margin: 5mm;

}

#abc1 {

width: 75%;

}

Tipi di dato (1)

- Interi e floating: rappresentano numeri assoluti (e.g., volume or z-index)
- URI : url(<http://www.site.com>)
- Stringa: una stringa tra virgolette semplici o doppie. Si usa il backslash per includere le virgolette nelle stringhe (e.g., "My name is \\"Fabio\\")

h1 {

z-index: 5;

}

Tipi di dato (2)

- Lunghezze: misure numeriche sono sempre espresse con un'unità di misura (e.g., em, en, ex, pc, px, pt, in, cm, mm, vh, vw).

Lunghezze assolute

- cm, mm, in: centimetri, millimetri, pollici
- pt (point): 1/72th di pollice
- pc (pica) pari a 12 punti o a 1/6 di pollice
- px (pixel), pari a 1 pixel del display in uso (N.B.: dipende dalla risoluzione e quindi può cambiare da device a device)

h1 {

line-height: 1.2em;

margin: 6mm;

}

Lunghezze relative

Misure tipografiche relative al font:

- em: la larghezza della lettera M, quindi le dimensioni del font,
- ex: l'altezza della x,
- ch: la larghezza dello 0,
- lh: l'altezza della riga
- vh, vw: percentuale dell'altezza o
larghezza della viewport
- rem: root em, ovvero la larghezza della M della pagina
(non dello specifico contenitore in cui ci si trova!)
- fr: un'unità frazionaria (una frazione
del numero di elementi previsti)

Tipi di dato (3)

- Percentuale: una misura relativa rispetto al contesto circostante (per esempio la scatola contenitore)
- Colori: o il nome (lo stesso di HTML), oppure il codice RGB con tre sintassi possibili:
 - Sintassi HTML: #XXYYZZ, dove XX, YY e ZZ sono codici esadecimali,
 - Sintassi rgb(x, y, z) , dove x, y e z sono numeri tra 0 e 255
 - Sintassi rgba(x, y, z, o), con x, y e z come prima e o è un numero tra 0 e 1 e rappresenta l'opacità (0 trasparente, 1 opaco). Si chiama anche canale alpha (da cui rgba)

```
h1 {  
width: 75%;  
color: #FF0000  
}
```

Proprietà

La scatola (box)

- La visualizzazione di un documento con CSS avviene identificando lo spazio di visualizzazione di ciascun elemento.
- Ogni elemento è presentato da una scatola che ne contiene il contenuto.
- Le scatole sono in relazione alle altre, e sono caratterizzate da flusso e posizione.

Flusso (proprietà display)

- I flussi sono gestiti implicitamente o dalla proprietà display:
- Le scatole degli elementi contenuti stanno dentro alla scatola dell'elemento genitore
 - Flusso blocco: le scatole sono poste l'una sopra l'altra in successione verticale (come paragrafi)
 - Flusso inline: le scatole sono poste l'una accanto all'altra in successione orizzontale (come parole della stessa riga)
 - Flusso float: le scatole sono poste all'interno del contenitore e poi spostate all'estrema sinistra o destra della scatola, lasciando che le altre scatole vi girino intorno
 - Ci sono molti altri tipi di flusso da considerare. Ne ripareremo.
- Gli elementi HTML hanno un valore di default per la proprietà

display. Questa va specificata solo se si vuole ignorarla.

margin edge

margin

border edge

border

padding edge padding

content edge

content

Elementi della scatola

- Margin: la regione che separa una scatola dall'altra, sempre trasparente (ovvero ha il colore di sfondo della scatola contenitore)
- Border: la regione ove visualizzare un bordo per la scatola
- Padding: la regione di respiro tra il bordo della scatola ed il contenuto. Ha sempre il colore di sfondo del contenuto
- Content: la regione dove sta il contenuto dell'elemento

top

bottom

left right

p {

background:rgb(170,213,213);

margin: 1em 2em;

border: 1em solid black;

padding: 1em 2em;

}

Proprietà tipografiche (1)

- font-size: lunghezza | percent
- font-family
 - lista di font names separati da virgole.
 - Usare le virgolette per nomi complessi (e.g., Tahoma, ma "Times New Roman")
 - L'ordine esprime preferenza.
 - Specificare "serif" o "sans-serif" come ultimo elemento per l'uso di un font di default del tipo corretto.

.traditional {

font-family: "Times New Roman", Palatino, serif;

font-size: 14pt;

}

.modern {

font-family: Arial, Tahoma, sans-serif;

font-size: large;

}

Proprietà tipografiche (2)

- font-weight: normal | bold | 100 | 200 | ... | 900
- font-style: normal | italic | oblique
- font-variant: normal | small-caps

.traditional {

font-family: "Times New Roman", Palatino, serif;

```
font-weight: bold;  
font-size: 14pt;  
}
```

```
.modern {  
font-family: Arial, Tahoma, sans-serif;  
font-style: italic;  
font-variant: small-caps;  
font-size: large;  
}
```

Proprietà tipografiche (3)

Esistono molteplici siti di font scaricabili. Ad esempio

- Google fonts
- 1001fonts.com

```
@import
```

```
url('//fonts.googleapis.com/css?family=Roboto  
'');  
@font-face {  
font-family: myFirstFont;  
src: url(sansation_light.woff);  
}
```

```
@import url('//fonts.googleapis.com/css?family=Roboto');
```

```
@font-face {  
font-family: Cute;  
src: url(my_cute_font.woff);  
}
```

```
.cool {  
font-family: Roboto;  
}
```

```
.cute {  
font-family: Cute;  
}
```

Proprietà tipografiche (4)

- text-decoration: none | underline | overline | line-through
- text-indent: length
- text-align: left | center | right | justified
- line-height: length

```
p {  
font-size: 12pt;  
text-align: justified;  
line-height: 18pt;  
text-indent: 3em;  
}
```

```
.deleted {  
text-decoration: line-through;  
}
```

(leading)

Proprietà tipografiche (5)

- text-align-last: left | center | right | justified

- text-transform: capitalize | uppercase | lowercase | none
- text-shadow: h-distance v-distance color
- font-stretch: normal | wider | narrower
- font-kerning: normal | none
- letter-spacing and word-spacing: length
- white-space: normal | pre | nowrap

```
h1 {
  text-shadow: 2px 2px #ff0000;
  text-transform: capitalize;
```

font-stretch: wider;

}

Proprietà della scatola

- padding: length
- margin: length
- color: color
- background-color: color
- border: length type color
- box-shadow: h-distance v-distance color

```
h1 {
```

padding: 5px;

color: #0000FF;

background-color: #DBEEF4;

border: 1px solid #0000FF;

}

p { margin: 1em 2em 3em 4em; }

P {

margin-top: 1em;

margin-right: 2em;

margin-bottom: 3em;

margin-left: 4em;

}

Forme abbreviate

In molti casi è possibile riassumere in un'unica proprietà i valori di molte proprietà logicamente connesse.

Si usa una sequenza separata da spazi di valori, secondo un ordine prestabilito (senso orario per le box). Se si mette un valore solo esso viene assunto da tutte le proprietà individuali. Ad esempio:

- margin per margin-top, margin-right, margin-bottom, margin-left
- border per border-top, border-right, border-bottom, border-left
- padding per padding-top, padding-right, padding-bottom, padding-left
- font per font-style, font-variant, font-weight, font-size, line-height, font-family

p { font: bold italic large Palatino, serif }

P { padding: 2em; }

p {

padding-top: 2em;

padding-right: 2em;

padding-bottom: 2em;

padding-left: 2em;

}

(questa roba va in tipografia)

Graphic Design

- Il graphic design è l'arte e il mestiere di creare lo stile e la presentazione visuale di un testo o documento multimediale.
- Il graphic design si è evoluto come disciplina separata dall'attività di scrittura fin dagli anni dell'Impero Romano, divenne un'arte nel Medioevo con la creazione dei cosiddetti codici miniati, e una professione (il tipografo) nel rinascimento, con l'invenzione della stampa a caratteri mobili. Da allora vi sono state numerose epoche e mode e rivoluzioni sempre all'incrocio tra arte, industria e tecnologia.
- Ci sono tre separate arti nel graphic design
 - Tipografia (typesetting)
 - Organizzazione della pagina (page layout)
 - Organizzazione iconografica (visual art curation)

Page layout

- Page layout è la disposizione armoniosa degli elementi visuali sulla pagina.
- Riguarda l'organizzazione di questi elementi rispetto alle loro dimensioni, alla loro posizione e alla quantità e qualità delle aree vuote intorno ad essi.
- Quando il contenuto richiede più spazio di quello disponibile dalla dimensione della pagina, le pagine vengono moltiplicate e si crea il fenomeno della paginazione. c'è molta differenza tra la paginazione su carta e quella su schermo, infatti su schermo lo spazio disponibile è virtualmente infinito e quindi si seguono altre regole.

La paginazione porta con sé alcuni problemi che in tipografia sono sempre citati con attenzione e storie dell'orrore:

- Il fronte e il retro delle pagine: molta carta non è completamente opaca e le parti nere del retro può trasparire di fronte come vaga area nera e creare uno sgradevole effetto visivo da evitare.
- Le pagine affiancate sono spesso guardate allo stesso momento, e possono fornire contrasti interessanti o sgradevoli o ridicoli.

Esempi:



Evoluzione del page layout:

Evoluzione del page layout



Nel tempo il modo di organizzare la pagina cambia molto, anche perché cambiano le tecnologie ed i costi dei materiali.

Aspetti del page layout:

- Orientamento
- Aspect Ratio
- Dimensioni
- Risoluzione
- Griglie di layout
- La sezione aurea

Orientamento

- Le aree rettangolari hanno due orientamenti naturali:
 - portrait (la dimensione lunga è l'altezza, es: libri e riviste)
 - landscape (la dimensione lunga è la larghezza, es: film e tv)
- La tradizione editoriale di libri è stata per lo più orientata al portrait, ma cinema (e TV) sono sempre stati landscape.
- I computer fino ai laptop usavano il layout degli schermi TV e quindi per lo più landscape.

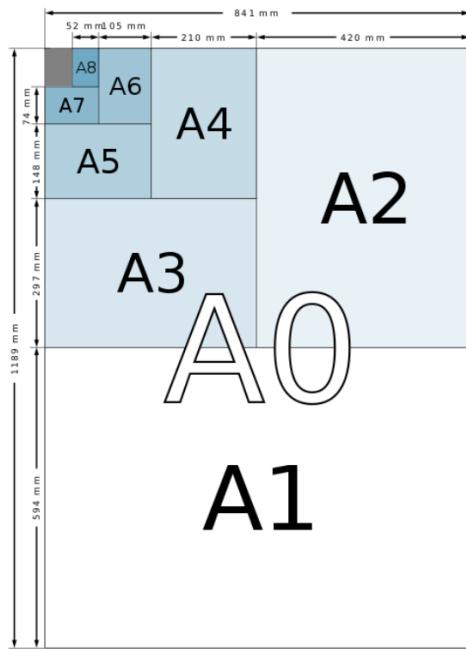
- Gli schermi mobili sono bidirezionali, e il portrait è l'orientamento più frequente negli smartphone (91%) mentre i tablet passano a portrait oltre una certa dimensione (14% a 7", 48% a 8", 82% a 11").

Aspect ratio

- Il rapporto tra altezza e larghezza della pagina/schermo.
- Indipendente dalla dimensione e dall'orientamento usato.
- Aspect ratio comuni:
 - Quadrato (1.0000) – Non molto usato ma è un punto di partenza
 - US Letter (1.2941) – Carta da lettere americana
 - 4/3 (1.3333) – Trasmissioni televisive analogiche (fino al 2002)
 - 11/8 (1.3750) – i film di Hollywood (Academy ratio 35mm)
 - ISO 216 ($\sqrt{2}$ or 1.4142) – Carta europea (A0, A1, A2, A3, A4, A5)
 - 16/10 (1.6000) – La maggior parte degli schermi di computer moderni
 - Regola aurea (1.6180) – Nel mondo classico il rapporto ideale
 - US Legal (1.6470) – Carta dei documenti legali americani
 - 16/9 (1.7777) – Schermi TV digitali e molte generazioni di smartphone
 - 18/9 (2.0000) – Nuovi smartphone Android
 - 19.5/9 (2.1666) – Nuovi smartphone iPhone
 - 70mm (2.2000) – Film di Hollywood ad alta risoluzione (wide screen)

Dimensioni

- ISO 216 (1975) stabilisce lo standard internazionale sulla dimensione della carta da usare in stampa con le serie A, B e C.
- Europa e Asia si sono allineati a questo standard, mentre gli USA e il resto dell'America sostanzialmente no.
- L'idea di base del modelli ISO 216 è di piegare e tagliare la carta a metà del lato lungo, ottenendo due fogli con lo stesso aspect ratio del foglio di partenza.
- Questo è possibile solo con un aspect ratio di $\sqrt{2}$ (1.4142).
- La dimensione A0 è quella di partenza, con area complessiva di 1m².
- Le dimensioni B e C sono intermedie (B0 è a metà tra A0 e A1, mentre C0 è a metà tra B0 e B1. Sono molto meno usate.



Risoluzioni

- In teoria, la risoluzione è la densità degli elementi visuali in una unità di area. La risoluzione non è appropriata nella stampa analogica ma è un fattore importante di valutazione nella stampa digitale. Si parla di dots per inches (DPI) in stampa, e pixel per inches (PPI) negli schermi.
- Con risoluzione negli schermi ci si riferisce erroneamente al numero totale di pixel invece che alla densità. Per ovviare a questo è stato introdotto il termine pixel density. I due valori sono interdipendenti una vota stabilità la dimensione dello schermo.
- Gli schermi digitali hanno tipicamente una densità minore delle stampanti, per cui la visualizzazione su schermo è più sfocata e meno dettagliata della stampa.

Stampa Risoluzione

Stampante dot matrix 60-90 DPI

Stampante inkjet 300-720 DPI

Stampante laser 600-2400 DPI

Typesetter professionale 1200-2800 DPI

Schermo Risoluzione

Apple standard resolution 72 PPI

Windows standard resolution 96 PPI

High density TV screens 213-240 PPI

Retina display (Mac & iPhones) 220-458 PPI

High end smartphones 534-807 PPI

Stampa	Risoluzione	Schermo	Risoluzione
Stampante dot matrix	60-90 DPI	Apple standard resolution	72 PPI
Stampante inkjet	300-720 DPI	Windows standard resolution	96 PPI
Stampante laser	600-2400 DPI	High density TV screens	213-240 PPI
Typesetter professionale	1200-2800 DPI	Retina display (Mac & iPhones)	220-458 PPI
		High end smartphones	534-807 PPI

Griglie

- La griglia è la struttura bidimensionale usata nel graphic design per allineare gli elementi e i componenti della pagina in modo gradevole.
- Le griglie vengono usate per raggruppare, allineare, contrastare e dare rilevanza visuale agli elementi della pagina. I componenti possono occupare uno, due o più celle della griglia, sia orizzontalmente sia verticalmente.
- Una griglia è densa se non c'è spazio tra le celle (cioè: margini assenti nelle celle). Una griglia è sparsa se intere celle vengono lasciate vuote per dare enfasi e breathing space alle celle piene, lo spazio vuoto tra le celle piene si chiama breathing gap.
- Le griglie possono dipendere o contrastare in maniera creativa le dimensioni complessive della pagina/schermo.
- Twitter Bootstrap usa una griglia di 12 colonne orizzontali (non ci sono vincoli verticali) perché 12 può essere diviso da 1, 2, 3, 4, 6, e 12 e questo dà molta flessibilità.

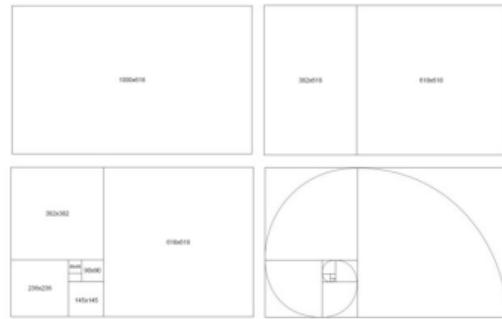
Esempio:



La sezione aurea:

Il rapporto aureo, o proporzione divina, o costante di Fidia, o sezione aurea (*golden ratio*) è il rapporto tra due numeri tale per cui il più grande è il medio proporzionale tra il più piccolo e la somma dei due:

- $\phi \stackrel{\text{def}}{=} (a+b) : a = a : b$
- $\phi = 1.6180339887\dots$
- $\phi = \frac{1+\sqrt{5}}{2}$
- ϕ è la soluzione positiva di $\varphi^2 - \varphi - 1 = 0$



Nelle arti grafiche, la sezione aurea è stata considerata fin da egizi e greci, come un rapporto particolarmente piacevole all'occhio. Il rapporto tra base e altezza delle piramidi, dei templi greci, nella disposizione degli elementi in un quadro rinascimentale segue spesso i rapporti progressivamente vincolati della sezione aurea.

La sezione aurea in tipografia

Anche nel page design, fin dal 1600, si considera la sezione aurea come una guida nello stabilire le proporzioni tra altezza e larghezza delle sezioni di una pagina, sia per libri che riviste che pagine web.

Fin dal 1600 i tipografi si basano sulla sezione aurea per identificare le aree grigie di una pagina a stampa. Jan Tschichold (1925) ha proposto l'uso consapevole della sezione aurea nel design.

Ci sono numerosi tool online per verificare che gli elementi di una pagina siano organizzati secondo modelli basati sulla sezione aurea.

Esempi:



Cascading Style Sheet (II parte)

Selettori e regole

Un selettore permette di specificare un elemento o una classe di elementi dell'albero HTML (o XML) al fine di associarvi caratteristiche CSS

– Esempi: h1, p.codice, img[alt]

Una regola è un blocco di statement associati ad un elemento attraverso l'uso di un selettore

– Sintassi

selettore {

statement; statement; ...

}

– Esempio

h1 {

color: white;

background-color: black;

```
}
```

```
p {
  color: blue;
}
.class {
  margin: 5mm;
}
#abc1 {
  width: 75%;
}
```

Selettori (1/7): universale, tipo, classe e id

pattern	significato	esempio
*	qualsiasi elemento	*
E	un elemento di tipo <i>E</i>	h1
E.Nomeclasse .nomeclasse	un elemento (di tipo <i>E</i> e) di classe <i>nomeclasse</i>	p.codice .codice
E#ilmioid #ilmioid	un elemento (di tipo <i>E</i> e) con id <i>ilmioid</i>	p#abc1 #abc

Selettori (2/7) pseudo-elementi

```
p::first-line {
  color: black;
}
p::first-letter {
  font-size: 300%;
}
q::before {
  content: "«";
}
q::after {
  content: "»";
}
```

Selettori (2/7) pseudo-elementi

pattern	significato	esempio
E::first-line	la prima riga formattata dell'elemento <i>E</i>	p::first-line
E::first-letter	la prima lettera formattata dell'elemento <i>E</i>	p::first-letter
E::before	contenuto generato prima dell'elemento <i>E</i>	q::before
E::after	contenuto generato dopo l'elemento <i>E</i>	q::after

Uno pseudo elemento è una parte del testo che richiede attributi diversi dal resto del testo, ma che non è un elemento di html

Selettori (3/7) prossimità

```

table th {
    font-weight: bold;
}
tr:first th {
    font-weight: normal;
}
tr:first + tr {
    color: gray;
}
tr:first ~ tr {
    color: green;
}

```

Selettori (3/7) prossimità

pattern	significato	esempio
E F	elemento <i>F</i> discendente di un elemento <i>E</i>	table th
E > F	elemento <i>F</i> figlio di un elemento <i>E</i>	tr > th
E + F	elemento <i>F</i> successivo diretto di un elemento <i>E</i>	label + input
E ~ F	elemento <i>F</i> successivo di un elemento <i>E</i>	h1 ~ p

il selettore di prossimità più utilizzato è E F, che significa che si applica se F è dentro E.

Selettori (4/7)

attributi

```

a[name] {
    font-style: italic;
}
a[href~= 'google.com'] {
    color: red;
}

```

Selettori (4/7) attributi

pattern	significato	esempio
E[foo]	un elemento <i>E</i> con un attributo <i>foo</i>	img[alt]
E[foo="bar"]	un elemento <i>E</i> con un attributo <i>foo</i> con valore uguale a "bar"	table[border="1"]
E[foo~="bar"]	un elemento <i>E</i> con un attributo <i>foo</i> il che contiene la parola bar	p[class~="codice"]
E[foo^="bar"]	un elemento <i>E</i> con un attributo <i>foo</i> con valore che inizia per "bar"	p[class^="cod"]

Selettori (5/7)

pseudo-classi

```

a:active {
    color:yellow;
}
a:hover {
    cursor:pointer;
}
input:checked {
    border:3px solid;
}

```

Selettori (5/7) pseudo-classi

pattern	significato	esempio
E:active	un elemento <i>E</i> è attivato dall'utente (es: click)	a:active
E:hover	un elemento <i>E</i> ha il puntatore sopra	a:hover
E:enabled	un elemento <i>E</i> di interfaccia se abilitato	input
E:checked	un elemento <i>E</i> di interfaccia è "checked"	input:checked

sono delle specie di classi che sono create automaticamente dal browser che le aggiunge e toglie in base al suo funzionamento.

Selettori (6/7) pseudo-classi strutturali

```

p:first-child {
    color:yellow;
}
p:nth-last-child(1) {
    color: green;
}
tr:nth-child(odd) {
    background: gray;
}

```

Selettori (6/7) pseudo-classi strutturali

pattern	significato	esempio
E:nth-child(n)	un elemento <i>E</i> che è l' <i>n</i> -simo figlio di suo padre	p:nth-child(odd)
E:nth-last-child(n)	un elemento <i>E</i> che è l' <i>n</i> -simo figlio di suo padre a partire dall'ultimo	p:nth-last-child(1)
E:nth-of-type(n)	un elemento <i>E</i> che è l' <i>n</i> -simo figlio di suo padre di quel tipo	p:nth-of-type(even)
E:first-child	un elemento <i>E</i> che è il primo figlio di suo padre	h1:first-child

Pseudo classi che non dipendono dall'ambiente esterno (generalmente l'utente) ma dalla posizione dell'elemento del codice.

Selettori (7/7):
pseudo-classi strutturali
e link
pattern significato esempio

```

p:first-of-type {
    margin-left: 5%;
}
td:empty {
    border-width:0px;
}
a:visited {
    color: gray;
}

```

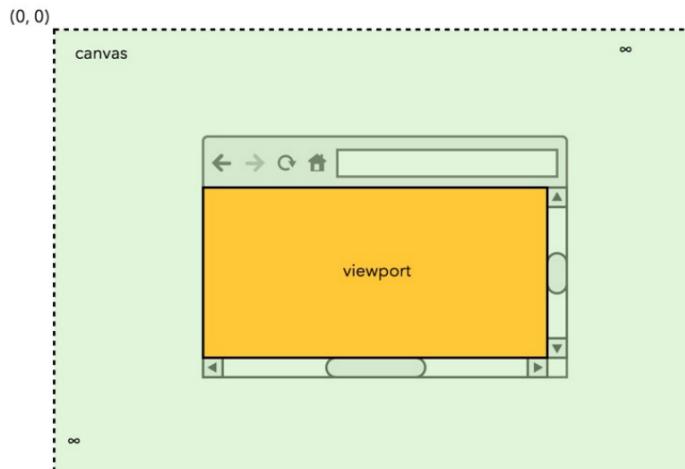
Selettori (7/7): pseudo-classi strutturali e link

pattern	significato	esempio
E:first-of-type	un elemento <i>E</i> che è il primo figlio di suo padre di quel tipo	p:first-of-type
E:only-of-type	un elemento <i>E</i> che è l'unico figlio di suo padre di quel tipo	h1:only-of-type
E:empty	un elemento <i>E</i> che è vuoto	td:empty
E:link	un elemento <i>E</i> che è un link non ancora visitato	a:link
E:visited	un elemento <i>E</i> che è un link già visitato	a:visited

Proprietà di css

Canvas e viewport

- Il canvas è l'area virtuale di posizionamento degli elementi del DOM via CSS. E' un piano cartesiano infinito in larghezza e altezza, con l'asse delle x da sinistra a destra e l'asse delle y dall'alto in basso.
- Il viewport è la parte del canvas che è attualmente visibile attraverso lo schermo o la finestra e può muoversi sul canvas grazie allo scrolling.
- Il viewport dipende ovviamente dalla dimensione dello schermo e si muove solo su coordinate positive del canvas.
- Per disegnare fuori schermo basta usare coordinate negative (offscreen drawing) .
- HTML fornisce un elemento <canvas> per disegnare sul canvas indipendentemente dal contenuto testuale del resto del documento HTML.



Unità di misura basate su canvas e viewport (1)

- Pixel (px) è l'unità principale di disegno sul canvas. Il pixel è la più piccola unità indirizzabile sullo schermo. Le dimensioni dei pixel dipendono fortemente dalla dimensione e dalla risoluzione degli schermi e sono molto diversi tra device e device e non sono un'unità affidabile. HTML però usa solo pixel.
- Le stampanti ri-scalano la dimensione dei pixel secondo rapporti che dipendono dalla risoluzione della stampante, con risultati non sempre convincenti.
- I device mobili, a loro volta, si basano sul tag `<meta name="viewport" content="width=XXX, initial-scale=YYY">` per scalare le dimensioni in pixel in maniera coerente, ma è sempre un'operazione rischiosa e poco affidabile.
Suggerimento: usare sempre `initial-scale=1` e non usare MAI misure basate su pixel nei propri CSS.
Eccezione: 0px e 1px

Unità di misura basate su canvas e viewport (2)

- Viewport width (vw): una percentuale (1%) della larghezza attuale del viewport. In a viewport that is 500 pixel wide, 3vw = 15px.
- Viewport height (vh): una percentuale (1%) della altezza attuale del viewport.
- vmin: il più piccolo tra vw e vh
- vmax: il più grande tra vw e vh

Suggerimento: usate unità di viewport per creare layout responsive che si adattano alla dimensione dello schermo o della finestra. Usate vmin e vmax per adattare il layout sia all'uso in modalità portrait sia all'uso in modalità landscape.

L'unità flex (fr)

- Una lunghezza flessibile (flex) è una dimensione con unità fr che rappresenta una frazione dello spazio rimasto nel contenitore.

- Una volta escluse dallo spazio del contenitore le componenti con dimensioni non flessibili, si sommano le unità fr, si divide lo spazio rimasto rispetto a tale somma e si distribuisce lo spazio rimasto alle varie componenti in proporzione alla loro frazione.

- Molto comodo per organizzare rapporti complessi senza dover esplicare le operazioni corrispondenti.

```
.mygrid {
  display: grid;
  grid-template-columns: 20% 2fr 1fr;
}
```

Posizione della scatola

La posizione dipende dalle altre scatole, dallo sfondo (canvas) o dalla finestra (viewport).

- Posizionamento statico (default): la scatola viene posta nella posizione di flusso normale che avrebbe naturalmente.
 - Posizionamento assoluto: la scatola viene posta nella posizione specificata indipendentemente dal flusso (nascondendo ciò che sta sotto).
 - Posizionamento relativo: la scatola viene spostato di un delta dalla sua posizione naturale
 - Posizionamento fisso: la scatola viene posta in una posizione assoluta rispetto alla finestra (viewport), senza scrollare mai
 - Posizionamento sticky: la scatola viene posta nella sua posizione naturale all'interno del suo contenitore, ma durante lo scrolling sta fissa rispetto al viewport fino a che il contenitore non esce dalla vista.
- Ovunque si lavori esplicitamente sulla posizione, si possono indicare fino a quattro proprietà tra top, left, right, bottom, width, height

Posizionamento:

position, float, coordinate e dimensioni

- position (static | relative | absolute | fixed) gestisce il posizionamento rispetto al flusso del documento

• float (left | right | none) è una scatola scivolata all'estrema destra o sinistra del contenitore muovendo le altre per farle posto

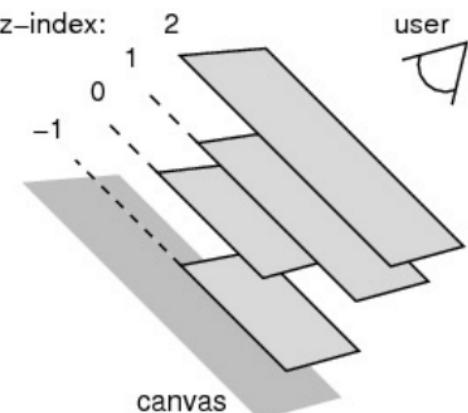
• top, bottom, left, right: coordinate della scatola

• width, height: dimensioni usabili invece di right e bottom

Posizionamento: z-index

z-index è la posizione nella pila di scatole potenzialmente sovrapposte.

Il valore più alto è più vicino



al lettore, e quindi nasconde gli altri.

N.B.: per default il background delle scatole è trasparente.

Posizionamento: overflow

overflow specifica come trattare il contenuto che non sta interamente nella scatola (forse con dimensioni troppo rigide). Valori possibili:

- visible: la scatola si espande per contenere tutto il contenuto
- hidden: il contenuto extra viene nascosto
- scroll: compare una scrollbar per l'accesso al contenuto extra.
- Si possono specificare le proprietà overflow-x and overflow-y per permettere la comparsa di una sola delle scrollbar. Visible Hidden Scroll overflow-x:hidden; overflow-y:scroll;



Esempio di posizionamento

```
<p>Alcune parole di un paragrafo  
che si estende per <span  
class="left">righe e righe</span>,  
così da far vedere come si comporta  
su più righe.</p>  
<p>Secondo paragrafo che contiene  
altre parole e un pezzo in  
<b>grassetto</b> ed uno in  
<i>corsivo</i>.</p>  
<p class="abs">Terzo paragrafo  
posizionato in maniera assoluta  
dove capita </p>
```

```
p.abs {  
    position: absolute;  
    top: 40px;  
    left: 210px;  
    width: 190px;  
}  
span.left {  
    float:left;  
    font-size: 200%;  
}
```

Alcune parole di un paragrafo che si estende per righe e righe, così da far vedere come si comporta su più righe. Terzo paragrafo posizionato in maniera assoluta dove capita. Secondo paragrafo che contiene altre parole e un pezzo in grassetto ed uno in corsivo.

```
p, b, i, span {  
    border-style: solid;  
    border-width: 1px;  
}  
p.abs {  
    background-color: white;  
}  
b, i {  
    background-color: yellow;  
}
```

Colonne di testo

In CSS3 è stata introdotta la possibilità di gestire colonne multiple
Il contenuto prosegue naturalmente (senza l'uso di tabelle) da una
colonna all'altra e il numero delle colonne può variare
automaticamente a seconda della dimensione della viewport

```
div {  
    column-width: 15em;  
    column-gap: 2em;  
    column-rule: 4px solid green;  
    padding: 5px;  
}
```

Il layout in CSS

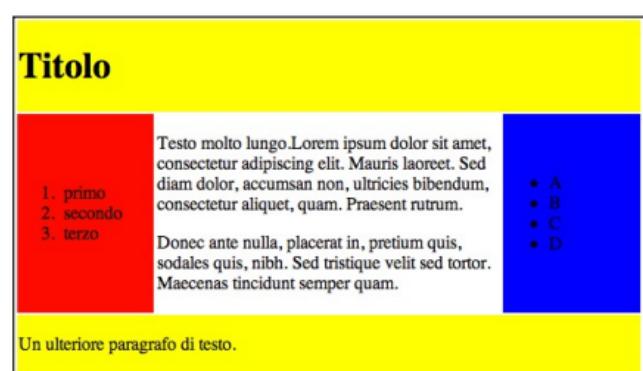
La proprietà display

- La proprietà display gestisce la natura e l'organizzazione della scatola rispetto a contesto (mondo esterno) e contenuto (mondo interno)
- Ogni elemento HTML ha definita un valore di default per la proprietà display che lo caratterizza. Lo cambiamo solo quando serve:
 - display: block;
 - display: inline;
 - display: table;
 - display: table-row;
 - display: table-cell;
 - display: list-item;
- Il modo più semplice e definitivo per nascondere un elemento è:
 - display: none;
- Oppure potete fare sparire la scatola esterna tenendo quelle interne:
 - display: contents;
- O, infine, ci possiamo fare del layout complesso:
 - display: grid;
 - display: flex;

Il layout

- Il layout è l'organizzazione spaziale delle componenti strutturali più importanti di una pagina web.
- Il layout "naturale" prevede il posizionamento delle scatole secondo il flow di tipo blocco e inline degli elementi HTML di partenza. Questo è raramente ciò che il designer vuole.
- Nel tempo sono state proposte ed usate molte tecniche diverse per ottenere il posizionamento delle scatole secondo le idee del designer. Ne vediamo alcune.

<http://www.fabiovitali.it/TW/2022/layout/>



Html però non permette di fare layout complessi come questo.

COPIA IMMAGINI

Layout 0: no layout

Se non uso niente non posso organizzare

sull'asse orizzontale

Layout 2: Float

Uso le proprietà float sugli oggetti che debbono muoversi orizzontalmente. Possibile solo fino a tre scatole diverse.

Layout 3: Positioning

Posiziono in maniera assoluta le scatole nella loro posizione voluta. Possibile ma complesso usare dei valori proporzionali allo spazio disponibile.

Layout 4: Tabelle CSS

Uso dei <div> come nel posizionamento ma uso la proprietà display delle tabelle (table, table-row, table-cell).

Piuttosto complicato e non faccio rowspan o colspan.

Grid

- Con display: grid; si può assegnare ad un elemento un'organizzazione visuale a griglia, organizzata per righe e colonne di altezza e larghezza controllabili.
- I figli dell'elemento con display grid possono essere assegnati ad una o più elementi della griglia in maniera libera ed indipendente dal document order.
- Posso assegnare altezze diverse ad ogni riga con la proprietà grid-template-rows e assegnare larghezze diverse ad ogni riga con la proprietà grid-template-columns. Posso creare spazio tra righe e colonne con la proprietà gap.
- Ogni elemento figlio dell'elemento grid identifica quali colonne e quali righe occupa o per posizione (grid-row e grid-column) oppure per nome (grid-area).

Layout 5: Grid

In questo esempio uso le named areas (grid-template-areas e grid-area) che mi permettono maggior controllo.

Flexbox

- Con display: flex; si può assegnare ad un elemento un'organizzazione visuale a contenuti flessibili e che si distribuiscono armonicamente nello spazio disponibile.
- L'elemento con display flex impone che i figli siano:
 - organizzati o per righe o per colonne (flex-direction),
 - disposti su più righe o colonne separate oppure forzate su una sola (flex-wrap)
 - disposti fianco a fianco oppure distribuiti per occupare tutto lo spazio in maniera omogenea (justify-content).
- Ogni figlio dell'elemento flex può:
 - espandersi o restringersi quanto serve per occupare in maniera controllata lo spazio disponibile (flex-shrink, flex-grow)

– cambiare di posizione rispetto al documento (order)

Layout 6: Flexbox

In questo esempio organizzo il contenitore per forzare tutto su una riga sola, e chiedo alle scatole interne di

Altri aspetti di CSS

- Cascata, ereditarietà ed !important
- Trasformazioni
- @rules e media queries
- Animazioni

Ereditarietà

Se non viene specificata una proprietà, CSS assume un valore di default

A parte pochi casi, questo è sempre inherit. Questo significa che la proprietà assume lo stesso valore che ha nella scatola contenitore dell'elemento in questione

Qui il contenuto dell'elemento em avrà il colore rosso:

```
<p style="color:red;  
">
```

Qui è *in corsivo* e qui no.

```
</p>
```

Tra i valori non ereditati

display (per HTML è sempre il valore naturale dell'elemento, block per p o h1, inline per b, i o a, mentre per XML dipende)

background (sempre transparent)

Eccezione alla cascata: !important (1/2)

La keyword !important può essere aggiunta in fondo a qualunque statement e aumenta la priorità dello statement anche rispetto a statement successivi attivabili in cascata.

Questo è il primo esempio di cascata, basata su tre fogli di stile, che riportano ciascuno alcune regole:

```
p { font-family: Arial; font-size: 12pt; }
```

```
p { color: red; font-size: 11pt; }
```

```
p { margin-left: 15pt; color: green; }
```

gli attributi dell'elemento p saranno equivalenti a:

```
p {  
font-family: Arial;  
font-size: 11pt;  
margin-left: 15pt;  
color: green;  
}
```

Eccezione alla cascata: !important (2/2)

La keyword !important può essere aggiunta in fondo a qualunque statement e aumenta la priorità dello statement anche rispetto a statement successivi attivabili in cascata.

Questo è il primo esempio di cascata, basata su tre fogli di stile, che riportano ciascuno alcune regole. Aggiungendo !important:

```

p { font-family: Arial; font-size: 12pt !important; }
p { color: red !important; font-size: 11pt; }
p { margin-left: 15pt; color: green; }
gli attributi dell'elemento p saranno equivalenti a:
p {
font-family: Arial;
font-size: 12pt;
margin-left: 15pt;
color: red;
}
Ma la cascata è davvero una cascata?
<html>
<head>
<title>Esempio CSS</title>
<style type="text/css">
@import url(test.css);
</style>
</head>
<body>
<h1 class="title">I CSS: questi sconosciuti</h1>
<p id="p1">Ecco un primo esempio di uso dei CSS.</p>
</body>
</html>
* {
color: blue !important;
}
p:first-of-type {
background: gray;
color: yellow;
}
@media screen {
p {
background: black;
color: white;
}
}
@media print {
p {
width: 8cm;
margin: 1cm 1.5cm;
}
}
test.css
p {
background: gray;
color:blue; x }
p {
background: black;

```

```
color:white;
```

```
}
```

accedo via browser

A cascata sì, ma con calma

L'ordine con cui vengono considerate le varie regole non

è dato solo dall'ordine in cui sono posti nei fogli di stile

Infatti, viene applicato un algoritmo di ordinamento

sulle dichiarazioni secondo alcuni principi (dal più al

meno importante):

- il media-type (print, screen, speech, ecc.) a cui si riferisce una dichiarazione

- l'importanza di una dichiarazione (!important)

- l'origine della dichiarazione (utente, autore, user agent)

- la specificità del selettori della dichiarazione

- l'ordine in cui si trovano le dichiarazioni

L'ordine della cascata

La cascata non è ordinata in maniera assoluta sulla base dell'ordine

delle dichiarazioni, ma tenendo in considerazione l'importanza e

l'origine, con il seguente ordine di precedenza:

1. dichiarazioni dello user agent

2. dichiarazioni dell'utente

3. dichiarazioni normali dell'autore

4. dichiarazioni importanti dell'autore (!important)

5. dichiarazioni importanti dell'utente (!important)

Le regole della stessa importanza e origine sono ordinate per specificità del selettori, così i selettori più specifici coprono quelli più generali.

Infine regole della stessa importanza e origine e con selettori della stessa specificità sono considerati in document

order

Trasformazioni CSS

- Trasformazioni geometriche sulla scatola una volta che è stata completamente definita e generata.

- Si usa la sintassi transform: function(parameters);

- Esempi di funzioni:

- translate(), translate3d(): sposta la scatola

- scale(), scale3d(): allarga/rimpicciolisce la scatola

- rotate(), rotate3d(): ruota la scatola

- skew(): inclina la scatola

- Ecc.

Trasformazioni CSS: scale(X,Y)

```
<style>
```

```
div {
```

```
background-color: yellow;
```

```
width: 200px;
```

```
border: 1px solid black;
```

```
}
```

```
</style>
```

```

<div>
<p>Un DIV</p>
<p>Con due paragrafi</p>
</div>
transform:scale(0.5,3);
transform:scale(2,1);
Trasformazioni CSS: rotate(deg)
transform:rotate(10deg);
transform:rotate(180deg);
transform:rotate(270deg);
<style>
div {
background-color: yellow;
width: 200px;
border: 1px solid black;
}
</style>
<div>
<p>Un DIV</p>
<p>Con due paragrafi</p>
</div>
At rules
Le regole precedute da un "@", comunemente chiamate at rules,
servono per specificare ambiti o meta-regole del foglio di stile
– @import: permette di importare regole da altri stili
– @charset: serve per specificare l'encoding (es: UTF-8)
– @namespace: permette di definire namespace all'interno di un CSS ed
usarli nei selettori
– @page: serve per definire caratteristiche di margine dell'intera pagina
– @font-face: permette di specificare i font "customizzati" da utilizzare
(vengono scaricati automaticamente)
– @media: descrive il media-type di destinazione per cui un insieme di
statement devono essere applicati
– @keyframes: descrive stati iniziali, intermedi e finali di un'animazione in
CSS.
@font-face:
specificare un font non installato
@font-face {
font-family: MyFont;
src: url("http://www.example.com/font");
}
h1 {
font-family: MyFont, sans-serif;
}
@media:
specificare regole dipendenti dal device
@media screen, projection {
html {

```

```
background: #ffffe0;
color: #300;
}
body {
max-width: 35em;
margin: 15px;
}
}
@media print {
html {
background: #fff;
color: #000;
}
body {
padding: 1in;
border: 0.5pt solid #666;
}
}
```

Media queries

Le media queries servono per specificare delle regole particolari che vengono attivate nel caso in cui il supporto usato per visualizzare la pagina Web soddisfa particolari vincoli

Tra le varie espressioni utilizzabili, quelle più comuni permettono di realizzare dei vincoli sulla larghezza, altezza e colore supportati dal dispositivo

Tramite il loro uso, si può adattare la presentazione di una pagina Web a device differenti (pc, smartphone, ecc.) senza cambiare di una virgola il contenuto della pagina

Sintassi:

```
@media query {
selettore {
statement;
statement; ...
}
}
```

Media queries

- Il linguaggio per creare media queries è complesso, ed è composto di regole separate, collegate da operatori come and, or, only.

- print and screen
- only screen and (max-width: 600px)
- not speech

- Ci sono più di 30 feature usabili nelle media queries, tra cui any-pointer, aspect-ratio, color, height, hover, max-height, max-width, min-height, min-width, monochrome, orientation, resolution, width, ecc.

Esempio di uso di media query

<http://www.fabiovitali.it/TW/2022/layout/13-mediaqueries.html>

Animazioni in CSS

- @keyframes è un blocco di regole per specificare lo stato iniziale (from), lo stato finale (to) ed eventuali stati intermedi (percentuali) di una o più proprietà numeriche CSS.

- Le proprietà più importanti dell'animazione:

- animation-name: blocco keyframes da utilizzare.
- animation-delay: ritardo di partenza
- animation-duration: durata dell'animazione
- animation-iteration-count: numero ripetizioni
- animation-timing-function: la curva di accelerazione.

- La proprietà animation è una forma abbreviata di queste proprietà.

Un esempio di animazione

<http://www.fabiovitali.it/TW/2022/layout/11-animation.html>

Un esempio di animazione

Limiti del CSS

Il CSS ha noti limiti di flessibilità, perché le regole non condividono valori e vanno sempre posti individualmente.

- Sarebbe utile definire lo stesso colore o lo stesso font per molte regole in una volta sola

- Sarebbe utile definire regole sulla base di altre regole
- Sarebbe utile introdurre formule aritmetiche

A partire dal 2009 sono state proposti linguaggi che estendono CSS in questa direzione, sotto forma di pre-processori CSS:

- Attraverso compilazione (cioè il sorgente viene trasformato per generare CSS tradizionale)
- Attraverso interpretazione (cioè il sorgente viene affidato ad uno script client-side – Javascript – per la esecuzione)

Con Level 4 alcuni di questi concetti sono transitati nel CSS standard

LESS e SASS

I due preprocessori più noti. Ammettono entrambi compilazione e interpretazione.

Feature principali:

- Variabili: valori ripetuti più volte;
- Mix-in: frammenti di regola utilizzabili in più contesti. Ammettono anche parametri, come le funzioni.
- Aritmetica: espressioni numeriche anche complesse

Variabili

```
@color: #4D926F;  
#header {  
color: @color;  
}  
h2 {
```

```
color: @color;  
}  
Mixin  
@ mixin bordo($color, $width) {  
border: {  
color: $color;  
width: $width;  
style: dashed;  
}  
}  
p {  
@include bordo(blue, 15px);  
}
```

Aritmetica

```
@the-border: 1px;  
@base-color: #111111;  
#header {  
border-left: @the-border;  
border-right: @the-border * 2;  
color: @base-color * 3;  
}  
#footer {  
color: @base-color + #003300;  
}
```

Variabili e calcoli aritmetici in CSS

- custom property: una proprietà ad hoc con un valore qualunque (usata come variabile personale)
--the-border: 1px;
--base-color: #111111;
- :root : è un selettori per la pseudo classe del document element di HTML, che ha scope globale non viene mai sovrascritto

```
:root {  
--the-border: 1px;  
--base-color: #111111;  
}
```

- var() : accede al valore di una custom property in qualunque parte del CSS
- calc(): permette calcoli aritmetici (abbastanza limitati)

```
#header {  
border-left: var(--the-border);  
border-right: calc( var(--the-border) * 2 );  
color: calc( var(--base-color) * 3 );  
}
```

N.B.: Poiché il carattere '-' è usato nei nomi CSS, gli operatori aritmetici vanno sempre separati con spazi

Twitter Bootstrap

Framework CSS

Librerie già pronte con regole associate ad elementi e classi, pronte per essere utilizzate nei documenti HTML con poche o

zero modifiche.

Hanno vari pregi:

- Gestiscono le differenze di implementazione delle funzionalità CSS tra i vari browser
- Forniscono accesso facilitato ad effetti speciali (in particolare le animazioni)
- Gestiscono layout responsive (che si adattano alla dimensione dello schermo)
- Forniscono un look integrato, omogeneo e professionale

Limiti

- Tendono ad uniformare pesantemente il look dei siti web

Twitter Bootstrap

In assoluto il più famoso dei framework CSS. Usato da centinaia di migliaia di siti.

Fornisce :

- Una suite integrata di classi CSS ben omogeneizzata per testi, immagini, form, ecc.
- Gestione di caratteristiche responsive del layout
- Feature di presentazione come tab, barre di navigazione, ecc.
- Un modello posizionale delle scatole basato su divisori del 12 (così da permettere 2, 3, 4, 6 e 12 scatole ben disposte in un contenitore).
- Classi Javascript standardizzate facilmente utilizzabili anche da autori HTML/CSS.

Responsive web design

La progettazione di una pagina web in modo da ottimizzare l'esperienza di utilizzo su tutti i device su cui viene presentato.

– Facilità di lettura

– Minimizzazione di ridimensionamento, scrolling e pannellazione

Elementi di un sito web responsive

- Griglia fluida e organizzata su quantità proporzionali alla dimensione dello schermo (dimensioni in % e non in pixel o cm)
- Immagini flessibili (sempre dimensionate dal contenitore e non assolute)
- Uso di @media query (per presentare diversamente la pagina su device e display diversi,

La griglia di Bootstrap

Bootstrap definisce una griglia virtuale divisa in dodicesimi. Ogni elemento occupa una porzione di 1, 2, ... 12 dodicesimi del contenitore, dall'intera finestra fino al più umile elementino.

Inoltre Bootstrap definisce quattro classi di schermi predefiniti (è possibile definirne altri con @media query apposite):

- xs (extra small - smartphone), sm (small - tablet), md (medium - laptop), lg (large - schermi Full HD 16/9, 1920x1080 o superiori ecc.)

E' possibile organizzare il layout specificando dimensioni lungo la griglia, e definire dimensioni diverse per schermi diversi.

Sono definite classi per le colonne col-{size}-{12esimi}. Ad

esempio col-xs-6 occupa metà della larghezza del contenitore per schermi piccoli.

Due esempi ritrovabili su:

<http://www.fabiovitali.it/TW/2022/bootstrap/>

Un esempio di griglia

- <div class="row">
- <div class="col-md-6 col-sm-9 col-xs-12">
- testo
- <div class="row">
- <div class="col-sm-6 col-xs-6">testo</div>
- <div class="col-sm-6 col-xs-6">testo</div>
- </div>
- </div>
- <div class="col-md-6 col-sm-3 col-xs-12">
- testo
- </div>
- </div>

Navbar e finestre modali

Bootstrap fornisce anche un lungo elenco di servizi grafici, come tabulatori, navbar, finestre modali, pannelli.

- Funzionano su tutti i browser
- Di alta qualità grafica e funzionale
- Usano un po' di Javascript ma sono richiamabili interamente via markup.

Un esempio di navbar

Bibliografia

- Cascading Style Sheets, level 1, W3C Recommendation 17 Dec 1996, revised 11 Apr 2008, <http://www.w3.org/TR/CSS1/>
- Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Working Recommendation 07 June 2011, <http://www.w3.org/TR/CSS2/>
- Selectors Level 3, W3C W3C Recommendation 29 September 2011, <http://www.w3.org/TR/css3-selectors/>
- CSS Color Module Level 3, W3C W3C Recommendation 07 June 2011, <http://www.w3.org/TR/css3-color/>
- CSS Namespaces Module, W3C W3C Recommendation 29 September 2011, <http://www.w3.org/TR/css3-namespace/>
- CSS Multi-column Layout Module, W3C Candidate Recommendation 12 April 2011, <http://www.w3.org/TR/css3-multicol/>
- Media Queries, W3C Recommendation 19 June 2012, <http://www.w3.org/TR/css3-mediaqueries/>

www.unibo.it

Fabio Vitali
Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna
Fabio.vitali@unibo.it

Javascript

ECMAScript è il nome ufficiale di Javascript.

Nasce come modo di fare verifiche client side nell'inserimento dei dati nei form.

- Linguaggio di script client-side.
- Nato nel 1995 con il nome di JavaScript introdotto da Sun e Netscape
- Standardizzato da ECMA International nel 1997. Per 18 anni sono state fatte piccole estensioni e modifiche. cerca cos'è ecma, è l'unico ente che accetta di standardizzare il linguaggio.

Ecmascript viene diviso in due parti:

- il cuore vero e proprio del linguaggio
- alcuni oggetti prefabbricati che si posso utilizzare come utilità nel browser

in gergo comune si dice che ecmascript è il linguaggio eseguito server-side (solo il cuore, ovviamente non ci sono gli elementi del browser), javascript è invece il lignguaggio eseguito client side con anche gli oggetti del browser.

- Nel giugno 2015 è stata approvata la versione 6 (nota anche come EcmaScript 2015). Con la forte pressione di WHATWG, sono state introdotte molte differenze, molti nuovi costrutti.
- Nel giugno 2021 è stata approvata la versione 12 (o EcmaScript 2021). Alcuni costrutti nuovi, tra cui String.replaceAll, e gli operatori ??=, |= e &&=.
- ES.Next è un nome dinamico che fa riferimento a feature già implementate mentre vengono discusse, e prima o poi rilasciate in una versione ufficiale di EcmaScript.
- Fa parte dell'approccio living standard del WHATWG
javascript ha un ciclo di sviluppo annuale che il WHATWG non ha cambiato, quindi tutt'ora esce con questa cadenza regolare.

Come eseguire uno script Javascript

- Client-side: eventi
 - Ogni elemento del documento ha alcuni eventi associati (click, mouseover, doppio click, tasto di tastiera, ecc), e degli attributi on+evento associati.
 - inserendo istruzioni JavaScript (o chiamata a funzione) nel valore dell'attributo si crea una chiamata callback.
 - Ci sono due eventi particolari da aggiungere: load (della window) e ready (del documento). Ne ripareremo.
- Server-side: routing
 - I servizi server-side sono associati a URI. Il modo più antico e semplice è creare servizi diversi e inserirli in file separati, ciascuno con un URI proprio. Aprendo una connessione all'URI, viene invocato lo script ed eseguito il servizio.
 - NodeJs (Express.js, in realtà) fornisce un meccanismo per associare una funzione javascript (callback) ad ogni tipo di URI. Aprendo una connessione all'URI, lo script centrale esegue la funzione corrispondente.

Come eseguire uno script sul browser

Appena viene scaricata la pagina In maniera sincrona (da leggersi come “all’istante”) appena lo script viene letto in un tag <script> o in un file.

- Adatto per inizializzare oggetti e variabili da usare più tardi.
- In maniera asincrona (da leggersi come “in seguito”, separato dal processo di navigazione dell’utente), associando il codice ad un evento sul documento (e.g., il click su un bottone): event-oriented processing
- Il tipo più comune di script Javascript
- In maniera asincrona, associando il codice al completamento di un’operazione di rete.
- Le operazioni Ajax vengono gestite tramite callback , funzioni eseguite appena la richiesta HTTP asincrona è completata e i dati sono stati ricevuti.
- In maniera asincrona, associando il codice ad un timeout, i.e., un periodo di attesa dopo il quale lo script viene eseguito automaticamente.
- Durante l’esecuzione dello script, il browser è bloccato e non reagisce agli input dell’utente. Per questo gli script debbono essere brevi e veloci in modo da lasciare all’utente la sensazione di interattività e controllo.
è buona norma fare script piccoli e veloci in modo che non intralciino l’esperienza di navigazione durante la parte asincrona.

Come mandare in output gli script

- HTML visualizza l’output degli script in 4 modi diversi:
 - scrivendo direttamente nella finestra del browser modificando il documento html, ciò è attualmente poco usato:
`document.write(string) ;`
 - scrivendo sulla console:
`console.log(string) ;`
 - scrivendo in una finestra di alert:
`alert(string) ;`
 - modificando il DOM del documento visualizzato, è il più utilizzato ed è considerato il migliore:
`document.getElementById(id).innerHTML = string ;`
- I prossimi esercizi si trovano in
<http://www.fabiovitali.it/TW/2022/js/>

Come attivare gli script

- HTML prevede l’uso di script in tre modi diversi
 - posizionato nell’elemento dentro all’attributo di un evento
 - posizionato nel tag <script> (quindi lo script funziona solo per questa pagina)
 - indicato in un file esterno puntato dal tag <script> (quindi lo script può essere richiamato da più pagine)

Posizionato dentro all’attributo di un evento

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Javascript example</title>
  </head>
  <body onLoad="alert('hello');">
    <h1 onMouseOver="console.log('Mouse Over!')>
      Javascript: learning to use it
    </h1>
    <p id="p1">
      I clicked on the button <span id="s1">
        <button onclick="
          var n = parseInt(document.getElem
            document.getElementById('s1').inn
          ">Add</button>
        <button onclick="
          document.getElementById('s1').inn
          ">Reset</button>
      </p>
    </body>
</html>

```

Posizionato nel tag <script>

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Javascript example</title>
    <script type="text/javascript">
      function hello() {
        alert('hello');
      }

      function add() {
        var n = parseInt(document.getElementById('s1').innerHTML);
        document.getElementById('s1').innerHTML = n+1;
      }

      function reset() {
        document.getElementById('s1').innerHTML = 0;
      }
    </script>
  </head>
  <body onLoad="hello()">
    <h1 onMouseOver="console.log('Mouse Over!')>
      Javascript: learning to use it
    </h1>
    <p id="p1">
      I clicked on the button <span id="s1">0</span> times.
      <button onclick="add()>Add</button>
      <button onclick="reset()>Reset</button>
    </p>
    <script type="text/javascript">
      document.write("<p>A second paragraph</p>") ;
    </script>
  </body>
</html>

```

Indicato dal tag <script>

```

A second paragraph



# Javascript: learning to use it



I clicked on the button 2 times.


<>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Javascript example</title>
    <script type="text/javascript" src="script1.js"> </script>
  </head>
  <body onLoad="hello()">
    <h1 onmouseover="console.log('Mouse Over!');">
      Javascript: learning to use it
    </h1>
    <p id="p1">
      I clicked on the button <span id="s1">0</span> times.
      <button onclick="add()>Add</button>
      <button onclick="reset()>Reset</button>
    </p>
  </body>
</html>

```

```

function hello() {
  alert('hello');
}

function add() {
  var n = parseInt(document.getElementById('s1').innerHTML);
  document.getElementById('s1').innerHTML = n+1;
}

function reset() {
  document.getElementById('s1').innerHTML = 0;
}

document.write("<p>A second paragraph</p>") ;

```

Javascript base

(In che modo Javascript è simile agli altri linguaggi)

la sintassi è simile ad altri linguaggi (c, java, ecc...).

la temporizzazione è invece più complicata da gestire (il sistema ad eventi).

javascript non è un linguaggio opinionato, ciò significa che sono permesse le peggiori porcherie (ti prego sistema) anche dalla comunità di programmatore; coloro che non accettano tale cose (per esempio anche google) utilizzano typescript, che è simile a javascript ma è opinionato e le variabili sono tipate.

JS: Tipi di dato

javascript è un linguaggio tipato, però non tipa le variabili ma soltanto i valori; ciò significa che una variabile può contenere dati di qualunque tipo.

- Javascript è minimale e flessibile per quel che riguarda i tipi di dati.
- Ci sono tre importanti tipi di dati atomici built-in:
 - booleani
 - numeri (sia interi, sia floating point)
 - stringhe
- Inoltre vanno considerati come tipi di dati anche
 - null
 - undefined
- C'è poi un unico tipo di dato strutturato, object, di cui fanno parte anche gli array.

JS: Variabili

- I dati in Javascript sono tipati, ma le variabili no.

var pippo ;

```

pippo = "ciao" ;
pippo = 15;
pippo = [1, 2, 3] ;
• tre modi per definire variabili
– var pippo='ciao' ; definisce una variabile nello scope della
funzione o del file in cui si trova (modo classico di definire una variabile).
– let pippo='ciao'; definisce una variabile nello scope del
blocco parentetico o della riga in cui si trova (modo moderno, inserito nel 2015).
– const pippo='ciao'; definisce una variabile non
ulteriormente modificabile.

```

JS: Operatori

Numeri

Operatore	Descrizione	Esempio	Commento
+	Somma	<code>let a = 5 + 7</code>	a vale 12
-	Sottrazione	<code>let b = 17 - 2</code>	b vale 15
*	Moltiplicazione	<code>let c = 5 * 4</code>	c vale 20
/	Divisione	<code>let d = 28 / 4</code>	d vale 7
%	Modulo	<code>let e = 15 % 6</code>	e vale 3 (15 / 6 = 2 resto 3)
**	Esponente	<code>let f = 3**2</code>	f vale 9 (cioè 3^2)
++	Incremento	<code>e++</code>	e vale 4 ($3 + 1$)
--	Decremento	<code>f--</code>	f vale 8 ($9 - 1$)

Stringhe

+	composizione	<code>g = "He"+"llo"</code>	g vale "hello"
+	composizione + casting	<code>h = "5" + 7</code>	h vale "57"

Confronto e booleani

Operatore	Descrizione	Esempio	Commento
==	Uguaglianza	<code>let i = b==c let j = 5=='5'</code>	i è falso (b vale 15 e c vale 20) j è vero (con casting di 5 in '5')
<	Minore	<code>let k = b<c</code>	k è vero
>	Maggiore	<code>let l = b>c</code>	l è falso
<=	Minore o uguale	<code>let m = b<=c</code>	m è vero
>=	Maggiore o uguale	<code>let n = b>=c</code>	n è falso
!=	Disuguaglianza	<code>let o = b!=c let p = 5!='5'</code>	o è vero p è falso (con casting di 5 in '5')
====	Uguaglianza senza casting	<code>let q = 5===='5'</code>	q è falso (non avviene casting di 5 in '5')
!==	Disuguaglianza senza casting	<code>let r = 5!==='5'</code>	r è vero (non avviene casting di 5 in '5')
&&	AND	<code>let s = i&&j</code>	s è falso
	OR	<code>let t = i j</code>	t è vero
!	NOT	<code>let u = !j</code>	u è falso

JS: Strutture di controllo condizionali (1)

Blocco if

```
if (a==5)          if (a==5) {  
    istruzione_singola ;      istruzione_1;  
                                istruzione_2;  
                                ...  
if (a==5) {          } else {  
    istruzione_1;      istruzione_3;  
    istruzione_2;      istruzione_4;  
    ...  
}  
                                ...  
}
```

Operatore ternario

```
let x = (b==5 ? 'pippo' : 'paperino') ;
```

Blocco switch

```
switch (a) {  
case 'a':  
    istruzione_1; istruzione_2 ;  
    ... ;  
    break;  
case 'b':  
    istruzione_3; istruzione_4 ;  
    ... ;  
    break;  
...  
default:  
    istruzione_n; istruzione_npiu1 ;  
    ...;  
    break;  
}
```



JS: strutture di controllo
cicli (1)

Blocco for

```
for (let i=0; i<k; i++) {  
    istruzione_1;  
    istruzione_2;  
    alert(i); ← i è l'indice di controllo del loop. E' un intero  
    ....  
}
```

Blocco for ... in

```
for (j in obj) {  
    istruzione_1;  
    istruzione_2;  
    alert(obj[j]); ← j è l'indice di controllo del loop. E' una stringa  
    ....  
}
```

il blocco for... in è il foreach

JS: strutture di controllo
cicli (2)

Blocco while

```
while (k < 5) {  
    istruzione_1;  
    istruzione_2;  
    ...  
}
```

Blocco do ... while

```
do {  
    istruzione_1;  
    istruzione_2;  
    ...  
} while (k < 5)
```

JS: strutture di controllo
eccezioni

Blocco try ... catch

```
let x=prompt("Enter a number between 0 and 9:","");
try {
    let el = document.getElementById("menu"+x)
    let address = el.attributes["href"].value
    return address ;
} catch(er) {
    return "input value out of bounds" ;
}
```

JS: funzioni

- Le funzioni in Javascript sono blocchi di istruzioni dotati di un nome e facoltativamente di parametri.
- Possono ma non sono obbligate a restituire un valore di ritorno. Le funzioni non sono tipate, i valori di ritorno sì (come al solito).

```
function double(n) {
    let m = n+n;
    return m ;
}
let a = double(4) ;
let b = double('test') ;
```

a vale 8
b vale 'testtest'

- Se manca un parametro, non restituisce errore ma assume che il parametro sia undefined.

```
function double(n) {
    if (typeof n!= undefined) {
        let m = n+n;
    } else {
        let m = 0 ;
    }
    return m ;
}
let c = double() ;
```

function double(n) {
 return n? n+n : 0;
}

c vale 0

in questa riga di codice nella prima parte (`n?`) è richiesto un booleano, javascript per fare il cast di un valore ad un booleano ritorna true se il valore è definito, false se è undefined.

JS: Tipi di dati strutturati

- Javascript ha un unico tipo di dato strutturato, chiamato object. Anche gli array sono un tipo speciale di object.
- Gli object sono liste non ordinate di proprietà,

coppie nome-valore.

```
let persona = {  
    nome: 'Giuseppe',  
    cognome: 'Rossi',  
    altezza: 180,  
    nascita: new Date(1995,3,12)  
}
```

JS: Tipi di dati strutturati (2)

- Il valore di una proprietà può essere esso stesso un object. let persona = {
 nome: 'Giuseppe',
 cognome: 'Rossi',
 altezza: 180,
 nascita: new Date(1995,3,12),
 indirizzo: {
 via: {
 strada: 'Via Indipendenza',
 numero: '15'
 },
 citta: 'Bologna',
 nazione: 'Italia'
 }
}

• E' un (frequente) errore la virgola dopo l'ultimo elemento di un blocco. Alcuni interpreti la ignorano, altri no.

JS: Tipi di dati strutturati (3)

Ci sono due sintassi per accedere alle proprietà di un object:

- Dot syntax (ispirazione dai linguaggi Object Oriented)
alert(persona.nome + ' ' + persona.cognome)
- Square bracket syntax (ispirazione dagli array associativi)
alert(persona['nome'] + ' ' + persona['cognome'])

il nome della proprietà in questo caso è una normalissima stringa:

```
let n1 = 'nome';  
alert(persona[n1]);
```

posso anche fare elaborazioni sulle stringhe:

```
let n2 = 'cog'+ n1;  
alert(persona[n1] +' '+ persona[n2]);
```

JS: Tipi di dati strutturati (4)

Uso entrambe le sintassi per leggere e per scrivere le proprietà dell'object:

```
persona.cognome = 'Verdi';  
persona['nome'] = 'Antonio';  
let n1 = 'nome';  
persona[n1] = 'Andrea';
```

Moltiplico i punti o aggiungo square bracket per proprietà in oggetti annidati:

```
persona.indirizzo.via.numero = '36';  
persona['indirizzo']['via']['numero'] = '15/a';
```

JS: Array (1)

Un array è un object in cui le chiavi sono numeri interi assegnati automaticamente. Per distinguerlo da un oggetto normale usa la parentesi quadra invece che la graffa.

```
let nomi = ['Andrea', 'Beatrice', 'Carlo'] ;
```

La dot syntax non può essere usata, ma solo quella bracket. Il primo numero è lo zero.

```
alert(nomi[0] + ' ' + nomi[1]) ;
```

Posso normalmente leggere e scrivere elementi dell'array:

```
nomi[0] = 'Adriano';
```

JS: Array (2)

Un array è un object con alcuni metodi e proprietà molto utili:

- length: lunghezza dell'array

```
let n = nomi.length;
```

- indexOf(item): la posizione di item nell'array

```
let k = nomi.indexOf('Beatrice')
```

- pop(): toglie un valore in fondo all'array e lo restituisce

```
let c = nomi.pop()
```

- push(item): aggiunge item in fondo all'array

```
nomi.push('Davide')
```

- shift(): toglie un valore in cima all'array e lo restituisce

```
let a = nomi.shift()
```

- unshift(item): aggiunge item in cima all'array e restituisce la nuova lunghezza

```
let d = nomi.unshift('Antonio')
```

```
n == 3
```

```
k == 1
```

```
c == 'Carlo', nomi == ['Andrea', 'Beatrice']
```

```
a == 'Andrea', nomi == ['Beatrice', 'Davide']
```

```
nomi == [Andrea, 'Beatrice', 'Davide']
```

```
let nomi = ['Andrea', 'Beatrice', 'Carlo']
```

```
nomi == ['Antonio', 'Beatrice', 'Davide']
```

JS: Array (3)

Altri metodi utili:

- slice(start,end): restituisce un array da start a end (escluso):

```
let b = nomi.slice(1,2);
```

- splice(pos,rimuovi,inserisci): inserisce e rimuove elementi

```
let pers = ["Andrea", "Barbara", "Carlo", "Elena"];
```

```
pers.splice(2, 1, "Claudio");
```

- join(sep): crea una stringa usando sep come separatore.

```
let p = pers.join(", ");
```

- Rimosso 1 elemento dalla posizione 2 ('Carlo')

- Aggiunto un nuovo item

```
persone == ["Andrea", "Barbara", "Claudio", "Elena"]
```

```
p == "Andrea, Barbara, Claudio, Elena"
```

```
nomi == ['Antonio', 'Beatrice', 'Davide']
```

```
b == ['Beatrice']
```

JS: Array (4)

Oggetti e array possono contenersi liberamente. Attenzione ad usare parentesi quadre per gli array e graffe per gli oggetti.

```

let persona = {
  nome: ['Giuseppe', 'Andrea', 'Federico'],
  cognome: 'Rossi',
  altezza: 180,
  nascita: new Date(1995,3,12),
  indirizzo: {
    via: {
      strada: 'Via Indipendenza',
      numero: '15'
    },
    citta: 'Bologna',
    nazione: 'Italia'
  },
  telefono: [
    { tipo: 'casa', numero: '051 123456'},
    { tipo: 'cell', numero: '335 987654'}
  ]
}

```

JS: Oggetti Predefiniti

- Javascript predefinisce alcuni oggetti utili per raccogliere insieme i metodi più appropriati per certi tipi di dati.
- Oggetti multipli
 - Object
 - Array
 - String
 - Date
 - Number
 - RegExp
 - ...
- Oggetti singoletto
 - Math
 - JSON
 - ...

JS: Stringhe

L'oggetto String contiene metodi disponibili per tutti i valori di tipo stringa:

```

let str = 'Precipitevolissimevolmente';
– length: lunghezza della stringa
let s = str.length;
– indexOf(sub): la posizione di sub nella stringa
let t = str.indexOf('ss')
– substring(start, end): restituisce la sottostringa da start ad end
let x = str.substring(3,8)
– substr(start, length): restituisce la sottostringa da start per length caratteri
let y = str.substr(3,8)
– split(sep): separa una stringa in array di utilizzando sep come separatore.
let w = "130.136.1.110" ;
let z = w.split(".");

```

```

s == 26
t == 13
x ==
'cipit'
z == ['130', '136', '1', '110']
y ==
"cipitevo"
JS: JSON (1)
JSON (JavaScript Object Notation) è un formato dati derivato dalla notazione
usata da JS per gli oggetti.
{
  "nome": ["Giuseppe",
    ,
    "Andrea",
    ,
    "Federico"],
  "cognome": "Rossi",
  ,
  "altezza": 180,
  "nascita": "1995-04-11T22:00:00.000Z",
  "indirizzo": {
    "via": {
      "strada": "Via Indipendenza",
      ,
      "numero": "15"
    },
    "citta": "Bologna",
    ,
    "nazione": "Italia"
  },
  "telefono": [
    { "tipo": "casa", "numero": "051 123456"},
    { "tipo": "cell", "numero": "335 987654"}
  ]
}
JS: JSON (2)

```

- Rispetto alla notazione usata nei programmi,
bisogna ricordare solo:
 - Solo valori string, number, boolean, array o object
 - Anche i nomi delle proprietà sono tra virgolette
 - Si usano solo le virgolette doppie e non le semplici
 - Non si possono inserire commenti di nessun tipo
- Tutti i linguaggi di programmazione più importanti
oggi accettano e scrivono dati in JSON

JS: JSON (3)

JSON è un singoletto in JavaScript che supporta due soli
metodi.

`let j = {`

```

nome: 'Fabio',
voto:10
};
let s = JSON.stringify(j);
let t = JSON.stringify(j, null, 2);
let k = '{\n"cognome": "Rossi",\n"amici": ["Andrea", "Lucia"]\n}';
let u = JSON.parse(k);
s vale la stringa: '{ "nome": "Fabio", "voto":10}'
t vale la stringa '{\n"nome": "Fabio",\n"voto": 10
}'
u vale l'oggetto {\n"cognome": "Rossi",\n"amici": ["Andrea", "Lucia"]\n}
JS: Date

```

Una data è un oggetto che esprime un giorno e un orario rappresentandolo come il numero di millisecondi trascorsi dalla mezzanotte del 1 gennaio 1970 e la data in questione. Poiché in realtà è un numero, questo permette di fare operazioni aritmetiche e confronti numerici.

– Costruttore:

```

let d = new Date();
let Capodanno = new Date(2022,0,1);
– getDay(): Il giorno della settimana della data (0-Domenica fino a 6-Sabato)
let w = Capodanno.getDay()
– toDateString(): converte il numero in una stringa leggibile
let a = Capodanno.toLocaleDateString()
let b = Capodanno.toDateString()

```

– Operazioni

```

let msInADay = 1000*60*60*24;
let msSinceCapodanno = d - Capodanno ;
let daysSinceCapodanno = Math.round(msSinceCapodanno/msInADay)

```

– Confronti

```

let In2022 = d > Capodanno
d contiene la data e l'ora di adesso

```

I mesi sono contati da 0

```

w vale 6 (Sabato)
"Sat Jan 01 2022"
"01/01/2022"

```

In2022 è vero

daysSinceCapodanno == 79

Altri oggetti

- Math
- Singoletto che raccoglie funzioni e costanti matematiche utili
- Math.PI, Math.abs(), Math.sin() Math.cos(),

Math.round(), Math.sqrt(), Math.random, ecc.

- RegExp

– Classe delle espressioni regolari, da usare per fare match su stringhe. Le RegExp usano '/' come delimitatore.

```
let str = 'Precipitevolissimevolmente' ;
```

```
let re = /pi(.)/ ;
```

```
let x = str.match(re) ;
```

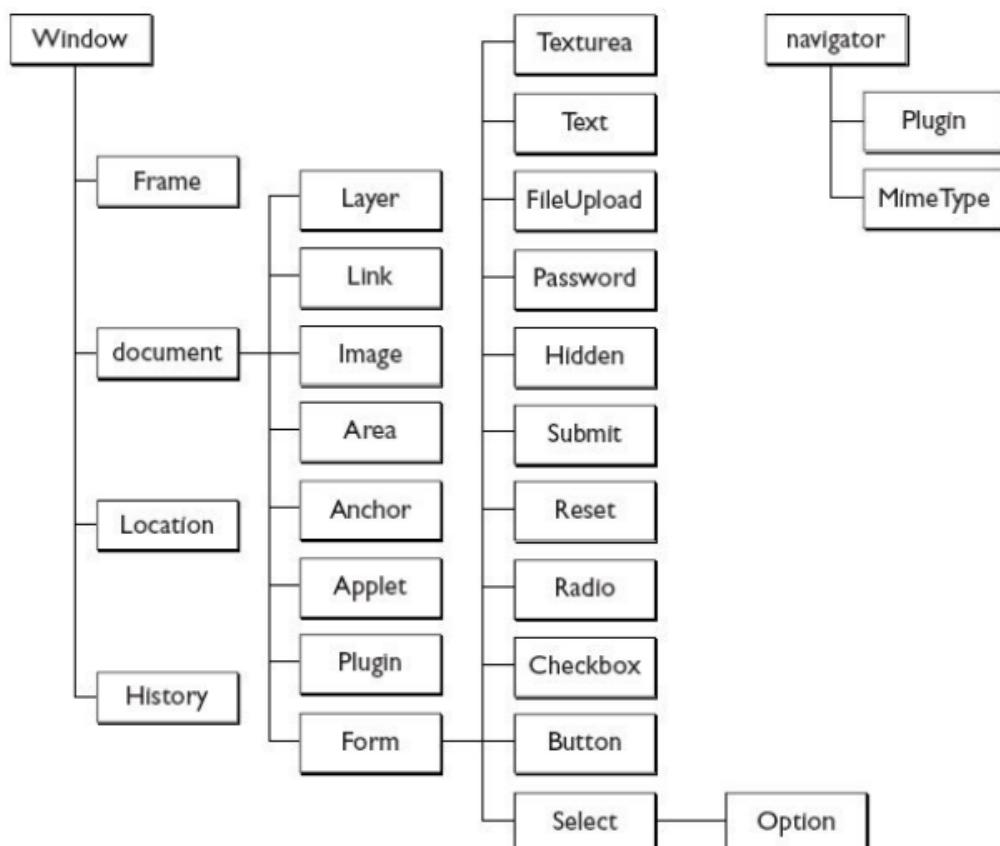
```
let y = re.exec(str) ;
```

sia x, sia y sono array

con due match: ['pit' e 't']

Javascript client side
ovvero lato browser

Gli oggetti predefiniti del browser



javascript client side si basa su due oggetti principali: window e navigator, di questi due navigator è poco utile, mentre window è essenziale.

window ha vari attributi:

- location: indica la pagina che si sta visualizzando (il suo uri).
- cose... copiaaaa

Gli oggetti principali:

window e navigator

- window: è l'oggetto top-level con le proprietà e i metodi della finestra principale:

- posizione: moveBy(x,y), moveTo(x,y), etc.
- dimensioni: resizeBy(x,y), resizeTo(x,y), etc.

– altre finestre: open("URLname", "Windowname", ["opt"])

- navigator: è l'oggetto con le proprietà del client come nome, numero di versione, plug-in installati, supporto per i cookie, etc.

Gli oggetti principali:

location e history

- location: l'URL del documento corrente. Modificando questa proprietà il client accede a un nuovo URL (redirect):

- window.location = "http://www.cs.unibo.it/";
- window.location.href = "http://www.cs.unibo.it/";

- history: l'array degli URL acceduti durante la navigazione.

Possibile creare applicazioni client-side dinamiche che 'navigano la cronologia':

- Proprietà: length, current, next
- Metodi: back(), forward(), go(int)

Gli oggetti principali:

document

document rappresenta il contenuto del documento, ed ha proprietà e metodi per accedere ad ogni elemento nella gerarchia:

- document.title: titolo del documento
 - document.forms[0]: il primo form
 - document.forms[0].checkbox[0]: la prima checkbox del primo form
 - document.forms[0].check1: l'oggetto con nome "check1" nel primo form (non per forza una checkbox!)
 - document.myform: l'oggetto "myform"
 - document.images[0]: la prima immagine
- Inoltre document rappresenta l'oggetto DOMDocument del DOM del documento visualizzato

Modello di documento

Ogni oggetto nella gerarchia è caratterizzato da un insieme di proprietà, metodi ed eventi che permettono di accedervi, controllarlo, modificarlo.

Javascript nasce per controllare i valori di un form:

```

function Verify() {
    if (document.forms[0].elements[0].value == ""){
        alert("Il nome è obbligatorio!")
        document.forms[0].elements[0].focus();
        return false;
    }
    return true;
}

<form action= “...” onSubmit="Verify()">
<p>Name: <input type="text" name="userNmae"> </p>

```

Javascript e DOM

Javascript implementa i metodi standard per accedere al DOM del documento.

```

var c = document.getElementById('c35');

c.setAttribute('class','prova1');
c.removeAttribute('align') ;

var newP = document.createElement('p');

var text = document.createTextNode('Ciao Mamma.');

newP.appendChild(text);

c.appendChild(newP);

```

```

// Creazione elementi singoli
olist = document.createElement("ol");
voce1 = document.createElement("li");
voce2 = document.createElement("li");
testo1 = document.createTextNode("un po' di testo");
testo2 = document.createTextNode("altro testo - item 2");

// Creazione lista completa
voce1.appendChild(testo1);
voce2.appendChild(testo2);
olist.appendChild(voce1);
olist.appendChild(voce2);

// Inserimenti lista in una data posizione
div = document.getElementById("lista");
body = document.getElementsByTagName("body").item(0);
body.insertBefore(olist,div);

```

Javascript ed eventi DOM

Javascript permette di associare callback di eventi ad oggetti (dichiarazione locale o globale)

```
<script language="JavaScript">
  window.onkeypress= pressed;
  window.document.onclick = clicked;

  function pressed(e) { alert("Key pressed: " + e.which);}
  function clicked() { alert("Mouse click! "); }
</script>
...
<body>
  <p id="mypara">Puoi
    <a href="test.htm" onClick="alert('Link!');">
      cliccare qui
    </a>
    oppure qui.
  </p>
</body>
```



innerHTML e outerHTML

Javascript (non DOM!) permette di leggere/scrivere interi elementi, trattandoli come stringhe:

- innerHTML: legge/scrive il contenuto di un sottoalbero (escluso il tag dell'elemento radice)
- outerHTML: legge/scrive il contenuto di un elemento (incluso il tag dell'elemento radice)

```
// HTML: <div id="d"><p>Paragrafo!</p></div>

d = document.getElementById("d");

alert(d.innerHTML);           // <p>Paragrafo!</p>
alert(d.outerHTML);          // <div id="d"><p>Paragrafo!</p></div>

d.innerHTML = "<ul><li>Lista!</li></ul>";
```

Selettori in DOM

I metodi standard in DOM per accedere ai nodi di un documento sono essenzialmente:

- getElementById: solo ovviamente se l'elemento ha un id
- getElementsByName: se l'elemento ha un attributo name
- getElementsByTagName: tutti gli elementi con nome specificato

Il successo di JQuery ha portato nel tempo a proporre ed implementare in DOM HTML anche due nuovi selettori:

- getElementsByClassName: cerca tutti gli elementi di classe specificata
- querySelector: accetta un qualunque selettore CSS e restituisce il primo elemento trovato - del tutto equivalente a \$()[0] in JQuery
- querySelectorAll: accetta un qualunque selettore CSS e restituisce tutti gli elementi trovati - del tutto equivalente a \$() in JQuery

Ajax

AJAX: Introduzione

AJAX (Asynchronous JavaScript And XML) è una tecnica per la creazione di applicazioni Web interattive.

inventato da garret ?? nel ??.

Permette l'aggiornamento asincrono di porzioni di pagine HTML

Utilizzato per incrementare:

- l'interattività
- la velocità
- l'usabilità

AJAX: Discussione

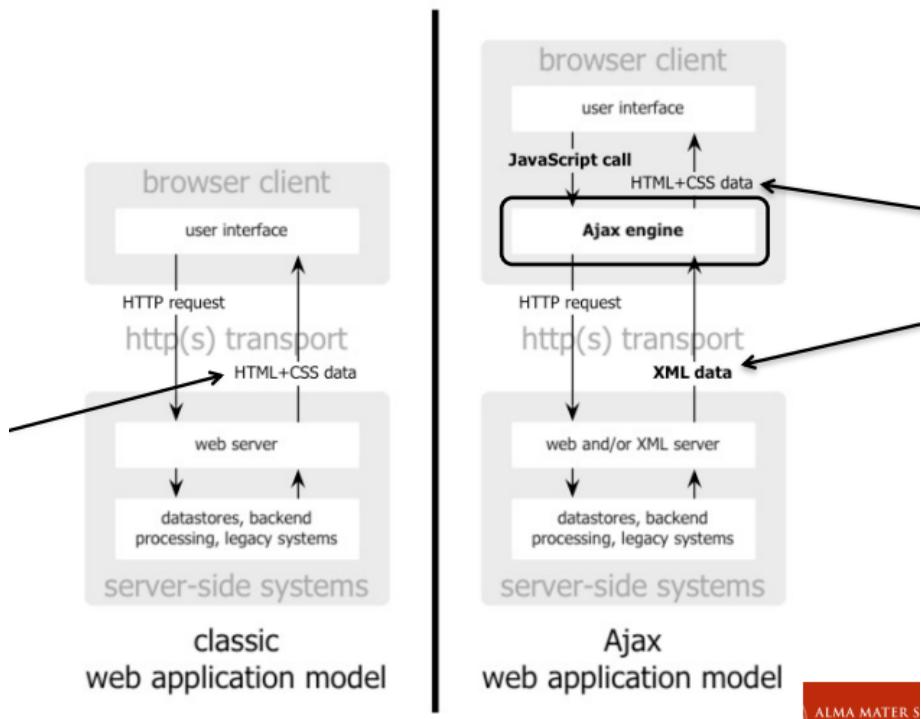
Non è un linguaggio di programmazione o una tecnologia specifica

E' un termine che indica l'utilizzo di una combinazione di tecnologie comunemente utilizzate sul Web:

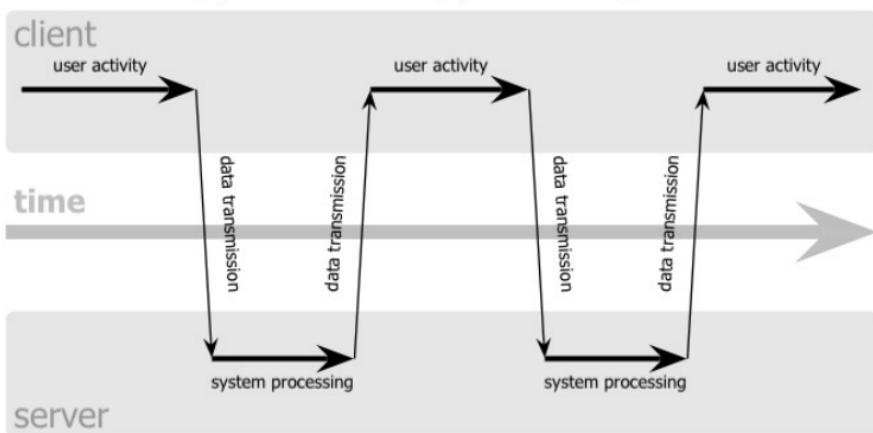
- XHTML e CSS
- DOM modificato attraverso JavaScript per la manipolazione dinamica dei contenuti e dell'aspetto
- XMLHttpRequest (XHR) per lo scambio di messaggi asincroni fra browser e web server
- XML o JSON come meta-linguaggi dei dati scambiati (json viene introdotto dopo ed attualmente è preferito)

invio di una richiesta e ricezione di una risposta sono operazioni separate, in modo da non tenere bloccato il browser mentre si attende la risposta del server.

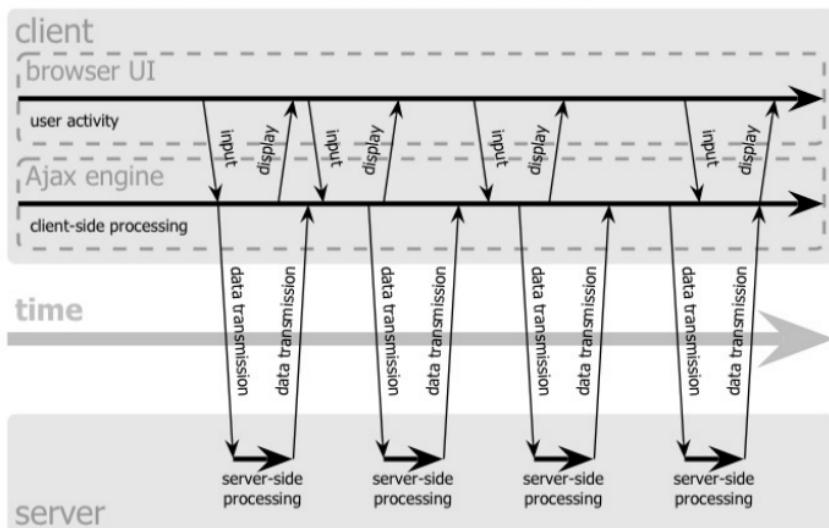
AJAX: Architettura (1)



classic web application model (synchronous)



Ajax web application model (asynchronous)



AJAX: Un po' di storia

Inizialmente sviluppato da Microsoft (XMLHttpRequest) come oggetto ActiveX
In seguito implementato in tutti i principali browser ad iniziare da Mozilla 1.0 sebbene con alcune differenze
Il termine Ajax è comparso per la prima volta nel 2005 in un articolo di Jesse James Garrett

Ajax: Pregi

- Usabilità
 - Interattività (Improve user experience)
 - Non costringe l'utente all'attesa di fronte ad una pagina bianca durante la richiesta e l'elaborazione delle pagine (non più click and-wait)
- Velocità
 - Minore quantità di dati scambiati (non è necessario richiedere intere pagine)
 - Una parte della computazione è spostata sul client
- Portabilità
 - Supportato dai maggiori browser
 - Se correttamente utilizzato è platform-independent
 - Non richiede plug-in

Ajax: Difetti

- Usabilità
 - Non c'è navigazione: il pulsante "back" non funziona
 - Non c'è navigazione: l'inserimento di segnalibri non funziona
 - Poiché i contenuti sono dinamici non sono correttamente indicizzati dai motori di ricerca
- Accessibilità
 - Non supportato da browser non-visuali
 - Richiede meccanismi di accesso alternativi
- Configurazione:
 - È necessario aver abilitato Javascript
 - in Internet Explorer è necessario anche aver abilitato gli oggetti ActiveX
- Compatibilità:
 - È necessario un test sistematico sui diversi browser per evitare problemi dovuti alle differenze fra i vari browser
 - Richiede funzionalità alternative per i browser che non supportano Javascript

Creare un'applicazione AJAX

Un'applicazione AJAX è divisa in alcuni momenti chiave:

1. Creazione e configurazione delle richieste per il server

- Usando XMLHttpRequest
- Usando funzioni di libreria che nascondono XMLHttpRequest
- Usando fetch()

2. Attivazione della richiesta HTTP...
3. ... passa del tempo...(ma il browser non è bloccato)
4. ... ricezione della risposta HTTP e analisi dei dati (o errore)
5. Modifiche al DOM della pagina

Creazione dell'oggetto

XMLHttpRequest

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // Internet Explorer
    try {
        http_request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
        try {
            http_request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) {
            ...
        }
    }
}
```

Inizializzazione della richiesta

(sincrona, asincrona)

La funzione open prepara la connessione HTTP (non viene ancora attivata). Due tipi di connessioni: sincrona e bloccante, oppure asincrona. Nel caso di funzione asincrona, prima di attivare la richiesta è necessario specificare la funzione che si occuperà di gestire la risposta e aprire la connessione con il server

I parametri della 'open' specificano:

- il metodo HTTP della richiesta,
- l'URL a cui inviare la richiesta,
- un booleano che indica se la richiesta è asincrona,
- due parametri opzionali che specificano nome utente e password

`http_request.onreadystatechange = nameOfTheFunction;`

`http_request.open('GET','http://www.example.org/some.file',true);`

`http_request.open('GET','http://www.example.org/some.file',false);`

Invio della richiesta

La richiesta viene inviata per mezzo di una 'send':

`http_request.send(null);`

Il parametro della 'send' contiene il body della risorsa da inviare al server:

- per una POST ha la forma di una query-string
`name=value&anothername=othervalue&so=on`
- per un GET ha valore "null" (in questo caso i parametri sono passati tramite l'URL indicato della precedente "open")
- può anche essere un qualsiasi altro tipo di dati; in questo caso è necessario specificare il tipo MIME dei dati inviati:
`http_request.setRequestHeader('Content-Type', 'mime/type');`

Gestione della risposta (1)

La funzione asincrona incaricata di gestire la risposta deve controllare lo stato della richiesta:

```
function nameOfTheFunction() {  
if (http_request.readyState == 4) {  
// risposta ricevuta  
} else {  
// risposta non ricevuta ancora  
}  
...  
}
```

I valori per 'readyState' possono essere:

```
0 = uninitialized  
1 = loading  
2 = loaded  
3 = interactive  
4 = complete
```

Gestione della risposta (2)

E' poi necessario controllare lo status code della risposta HTTP:

```
if (http_request.status == 200) {  
// perfetto!  
}  
else {  
// c'è stato un problema con la richiesta,  
// per esempio un 404 (File Not Found)  
// oppure 500 (Internal Server Error)  
}
```

Infine è possibile leggere la risposta inviata dal server utilizzando:

- http_request.responseText che restituisce la risposta come testo semplice
- http_request.responseXML che restituisce la risposta come XMLDocument

```
function getData(){  
// load the Ajax data  
myXMLHttpRequest = new XMLHttpRequest();  
myXMLHttpRequest.open("GET", "names.json", false);  
myXMLHttpRequest.send(null);  
//legge la risposta  
let d = JSON.parse(myXMLHttpRequest.responseText);  
// prepara i dati  
var fragment = prepareData(d)  
// modifica il documento corrente  
document.getElementById("area1").appendChild(fragment);  
}  
Leggere e visualizzare dati  
(modalità sincrona)
```

```

Browser bloccato e non accetta interazione con l'utente
function getData(){
// load the Ajax data
myXMLHttpRequest = new XMLHttpRequest();
myXMLHttpRequest.onreadystatechange = prepareData;
myXMLHttpRequest.open("GET", "names.json", true);
myXMLHttpRequest.send(null);
}
function showData() {
if (myXMLHttpRequest.readyState == 4) {
if (myXMLHttpRequest.status == 200) {
//legge la risposta
let d = JSON.parse(myXMLHttpRequest.responseText);
// prepara i dati
var fragment = prepareData(d)
// modifica il documento corrente
document.getElementById("area1").appendChild(fragment); }
}
}

```

Leggere e visualizzare dati

(modalità asincrona)

Browser immediatamente libero e accetta interazioni con l'utente

Semplificando...

- La complessità di XMLHttpRequest e le differenze di implementazione tra browser e browser hanno portato a suggerire molte alternative:
 - jQuery è stato introdotto ed è diventato famoso anche perché forniva un meccanismo per fare connessioni Ajax molto più semplice anche se ancora basato su callback.
 - Sia React sia Angular introducono librerie interne per fare connessioni Ajax totalmente integrate nel loro framework
 - Con il passaggio da W3C a WhatWG, e con l'introduzione delle promesse, è stata proposta e standardizzata una specifica API nativa, chiamata Fetch, per realizzare connessioni Ajax con una libreria nuova NON basata su XMLHttpRequest.
- Ne parleremo via via...

I framework Ajax

Sono librerie Javascript che semplificano la vita nella creazione di applicazioni Ajax anche complesse.

Hanno tre scopi fondamentali

- Astrazione: gestiscono le differenze tra un browser e l'altro e forniscono un modello di programmazione unico (o quasi) che funziona MOLTO PROBABILMENTE su tutti o molti browser.
- Struttura dell'applicazione: forniscono un modello di progetto dell'applicazione omogeneo, indicando con esattezza come e dove fornire le caratteristiche individuali dell'applicazione
- Libreria di widget: forniscono una (più o meno) ricca collezione di

elementi di interfaccia liberamente assemblabili per creare velocemente interfacce sofisticate e modulari

Categorie di framework

- Modello applicativo
 - Framework interni
 - Sono frameworks che vengono usati direttamente dentro alla pagina HTML con programmi scritti in Javascript
 - Framework esterni
 - Sono frameworks usati all'interno di un processo di sviluppo client e server e sono disponibili in un linguaggio di programmazione indipendente da Javascript (ad esempio in Java).
 - Ricchezza funzionale
 - Librerie di supporto JavaScript
 - Prototype, jQuery e MooTools sono semplicemente livelli di astrazione crossbrowser per task di basso livello DOM-oriented, ad esempio per arricchire graficamente un sito web tradizionale.
 - Framework RIA (Rich Internet Application)
 - Ext, GWT, YUI, Dojo e qooxdoo sono framework ricchi per la creazione di applicazioni complete, e includono una ricca collezione di widget, modelli di comunicazione client e server, funzionalità grafiche e interattive, e spesso anche strumenti di sviluppo.

Alcuni framework rilevanti

Librerie Javascript

- Prototype (<http://www.prototypejs.org/>)
- jQuery (<http://jquery.com/>)

Rich Internet Application Framework

- Angular (<http://angular.io/>)
- React (<http://reactjs.org/>)
- Vue (<http://vuejs.org/>)

Javascript avanzato

(in che modo è diverso dagli altri linguaggi)

javascript è un linguaggio object oriented ma non è basato sulle classi.

Peculiarità di Javascript

- Valori falsy e truthy (sono i booleani, in italiano significano verino e falsino, ovvero sono concettualmente booleani ma diversi nell'implementazione da quelli classici)
- Funzioni come entità di prima classe
- Classi e prototipi
- Closure e IIFE
- Altre peculiarità di ECMA 2015
- Gestione dell'asincronicità

JS: Falsy e truthy (1/4)

- Javascript definisce come falsy quei valori che in caso di casting a Booleano diventano falsi:

- false
- 0
- null
- undefined
- ""
- NaN
- Ogni altro tipo di valore è *truthy* (ovvero cast a true), inclusi:
 - "pippo", 3.14, Infinity
 - {}
 - []
 - "0", "undefined", "null"

JS: Falsy e *truthy* (2/4)

Qualunque programmatore di derivazione C o Java, per vedere se un valore è falso o nullo o indefinito, scriverà una cosa tipo:

```
if (value != null && value.length >0) {
// ok agisci
}
```

magari allungando pure la lista dei controlli.

In Javascript c'è il casting automatico, che permette una scrittura molto più semplice e veloce:

```
if (value) {
// ok agisci
}
```

JS: Falsy e *truthy* (3/4)

Se vi arrivano parametri da fuori, bisogna inizializzarli ad un valore decente se sono vuoti o falsi o non definiti. Supponiamo che param sia spesso un numero, ma non sempre:

```
if (param== null || param==0) {
misura = "12px" ;
} else {
misura = param + 'px'
}
```

o se ve ne ricordate, magari potreste usare l'operatore ternario:

```
misura = (param? param:'12')+'px' ;
```

Ma Javascript ha un modo ancora più semplice:

```
misura = (param || '12')+'px' ;
```

JS: Falsy e *truthy* (4/4)

In pratica, moltissimi programmatori lo usano come verifica della presenza e istanziazione di una variabile o la disponibilità di una libreria o un servizio.

```
if (window.XMLHttpRequest) {
...
} else if (window.ActiveXObject) {
...
}
```

Oppure per verificare l'esistenza di un parametro opzionale, invece di:

```
function connect(hostname, port, method) {
if (hostname === undefined) hostname = "localhost";
if (port === undefined) port = 80;
```

```

if (method === undefined) method = "GET";
...
}
posso usare:
function connect(hostname, port, method) {
hostname = hostname || "localhost";
port = port || 80;
method = method || "GET" ;
...
}

```

Funzioni in Javascript

JS: funzioni come entità di classe (1/6)

una funzione è un frammento di codice a cui viene associato un nome e che può essere riutilizzato più volte, ad una funzione viene associata una chiamata e dei parametri.

In Javascript, le funzioni sono oggetti quasi come tutti gli altri:

- Possono essere assegnate a variabili
- Possono essere passate come parametri di funzione
- Possono essere restituite da una funzione
- Possono essere elementi di un object
- Possono essere elementi di un array

in più:

- Possono essere invocate con l'operatore () (e quindi eseguite)

Una funzione

- Si può assegnarle ad una variabile:

```
var potenza = function(a,b) {
return Math.pow(a,b);
}
```

```
var c = potenza(5,3)
```

- Si può assegnare una funzione come metodo di un oggetto:

```
var e = {p:3,q:5,r:7}
e.sum = function() {
return this.p+this.q+this.r
}
```

- Il nome semplice è una variabile, se si aggiungono le parentesi diventa un'invocazione e la funzione viene eseguita:

```
if (e.sum) { // controllo l'esistenza: variabile
var c = e.sum() // invoco ed eseguo: funzione
}
```

```
function potenza(a,b) {
return Math.pow(a,b);
}
```

```
var c = potenza(5,3)
```

Function statement

Function expression

JS: funzioni come entità di classe (3/6)

Si può assegnarle come proprietà di un oggetto o di un array:

```

var persona = {
  nome: 'Giuseppe',
  cognome: 'Rossi',
  altezza: 180,
  nascita: new Date(1995,3,12),
  saluta: function(name, id) {
    var saluto = "Ciao "+name
    if (id) {
      document.getElementById(id).innerHTML = saluto;
    } else {
      alert(saluto);
    }
  }
}

```

JS: funzioni come entità di I classe (4/6)

Si possono restituire funzioni come risultato di altre funzioni:

```

var expGenerator = function(e) {
  return function(b) {
    return Math.pow(b,e)
  }
}

```

Posso creare delle funzioni in serie chiamando il generatore:

```

var quadrato = expGenerator(2);
var cubo = expGenerator(3);

```

E queste sono vere funzioni:

```

var c = quadrato(5); // c vale 25
var d = cubo(5); // d vale 125

```

Questa tecnica viene usata spessissimo!

JS: funzioni come entità di I classe (5/6)

Posso passare funzioni anonime come parametri di funzione:

```

var msg = document.getElementById('msg');
msg.innerHTML = '<p>via!</p>';
window.setTimeout(function() {
  msg.innerHTML += '<p>1 secondo è passato</p>';
}, 1000);
window.setTimeout(function() {
  msg.innerHTML += '<p>2 secondi sono passati</p>';
}, 2000);
window.setTimeout(function() {
  msg.innerHTML += '<p>3 secondi sono passati</p>';
}, 3000);

```

Nota: setTimeout(f,n) esegue la funzione f dopo n millisecondi dalla invocazione.

JS: funzioni come entità di I classe (6/6)

La funzione bind(obj,args) permette di associare parametri a funzioni anonime o chiamate indirettamente:

```

var msg = document.getElementById('msg');

```

```

msg.innerHTML = '<p>via!</p>' ;
for (var i=1; i<=3; i++) {
window.setTimeout(
function(n) {
this.innerHTML += '<p>' +n+' secondi sono passati';
}.bind(msg,i),
i*1000
);
}

```

Nella chiamata bind(obj,args), obj rappresenta l'oggetto a cui verrà associata la funzione (cosa trovo dentro alla variabile predefinita this), mentre args sono gli argomenti che voglio passare alla funzione.

Funzioni filtro su array

Gli array di Javascript hanno tantissimi metodi che accettano una funzione come parametro. Permettono di fare specifiche operazioni sugli elementi dell'array in maniera veloce e sistematica. Ad esempio:

```

let salespeople = [
{ name: 'Alice', cognome: 'Bruni' , sales: 78500 },
{ name: 'Bruno', cognome: 'Verdi' , sales: 135000 },
{ name: 'Carla', cognome: 'Rossi' , sales: 251200 },
{ name: 'Dario', cognome: 'Bianchi', sales: 7500 }
]

```

```

let byCognome = function (i,j) {return i.cognome > j.cognome ? 1 : -1 }
let largerthan100 = function(i) { return i.sales >= 100000 }

```

```

let best = function(i) { i.best = true }

```

```

let sorted = salespeople.sort(byCognome) ;

```

```

salespeople.filter(largerthan100).forEach(best)

```

sorted contiene gli elementi di

salespeople ordinati per cognome

Ho selezionato solo gli elementi di salespeople con vendite ≥ 100000 , poi ho assegnato loro il campo best a true. Ora salespeople è:

```

[ { name: 'Alice', cognome: 'Bruni' , sales: 78500 },
{ name: 'Bruno', cognome: 'Verdi' , sales: 135000, best: true },
{ name: 'Carla', cognome: 'Rossi' , sales: 251200, best: true },
{ name: 'Dario', cognome: 'Bianchi', sales: 7500 }
]

```

Funzioni filtro su array (2)

- array.sort(f)

- restituisce un array ordinato sulla base della funzione f, che deve avere due parametri e restituire un valore 1 (stesso ordine) o -1 (inverte l'ordine)

- array.filter(f)

- restituisce un secondo array che contiene solo gli elementi che soddisfano la funzione booleana f.

- array.some(f), array.every(f)

- restituisce vero se almeno un elemento (some()) o tutti gli elementi (every()) soddisfano la funzione booleana f.

- array.find(f)

- restituisce il primo elemento che soddisfa la funzione booleana f
- `array.forEach(f)`
- esegue sull'array la funzione f permettendo di modificare l'array direttamente.
- `array.map(f)`
- crea un nuovo array in cui ogni elemento viene modificato dalla funzione f
- `array.reduce(f)`
- esegue su ogni elemento dell'array la funzione f passando il risultato dell'esecuzione precedente. Ottimo per fare totali.

Object orientedness

in Javascript

JavaScript: oggetti e classi

- JavaScript è un linguaggio object-oriented anche se non è tipato come Java.
- In un linguaggio object oriented tradizionale, la classe è un template sulla base del quale vengono istanziati gli oggetti del programma, specificando i membri (stati dell'oggetto, valori) e i metodi (comportamenti dell'oggetto, funzioni).
- JavaScript ha oggetti, ma non sono basati sul concetto di classe, ma quello di prototipo.
- Poiché le funzioni sono oggetti di primo livello, ogni oggetto può contenere al suo interno delle funzioni, senza ricorrere alla classe.
- E' possibile istanziare oggetti semplicemente dichiarandone il contenuto, oppure tramite un costruttore.

Classi in Javascript

- Le classi non sono entità di primo livello. Al loro posto si usano degli oggetti che provengono dallo stesso costruttore.

- Un costruttore è banalmente una funzione che restituisce un oggetto. Si usa `new` per usarlo come costruttore dell'oggetto.

```
function Persona(nome,altezza,nascita){ // Costruttore
```

```
this.nome = nome
```

```
this.altezza = altezza
```

```
this.nascita = nascita
```

```
this.saluta = function() { return 'ciao!' }
```

```
}
```

```
var mario = new Persona("Mario", 175, new
```

```
Date(1993,6,14)); // Creazione di un oggetto
```

```
var alice = new Persona("Alice", 168); // manca un
```

parametro: non è un problema. `alice.nascita` è `undefined`

Parentesi pedantina

I linguaggi object-oriented sono divisi in:

- Class-based (es. SmallTalk, C++, Java, C#, etc.),
- la classe esiste come concetto esplicito e primario: le classi formano una gerarchia di tipi, l'ereditarietà avviene tra classi, gli oggetti sono istanze pure delle classi (non hanno metodi propri).
- Il design delle interfacce precede ed è strumentale alla creazione degli algoritmi per la esecuzione dei compiti dell'applicazione. Questo facilita

la compilazione e fornisce una base "contrattuale" tra creatori ed utenti degli oggetti per la garanzia del buon funzionamento del programma.

- Prototype-based (es. ECMAScript, Javascript, etc.),
- non esiste il concetto di classe, ma quello di prototipo, una istanza primaria, astratta, sempre accessibile e modificabile, di cui le singole istanze clonano (e, se serve, modificano) sia membri sia metodi.
- Il design delle interfacce è contemporaneo e indipendente dalla creazione degli algoritmi, e può essere modificato in qualunque momento, anche a run-time. Non c'è contratto, ma massima flessibilità.

JavaScript: Prototype

- Ogni oggetto in Javascript è autonomo e si possono aggiungere tutti i metodi/proprietà che si vuole senza modificare gli altri.
- Per aggiungere proprietà/metodi condivisi da molti oggetti debbo usare l'oggetto prototype.
- Si usa per creare o riusare librerie di oggetti e metodi:
 - estendere le proprietà di un oggetto built-in nel linguaggio
 - estendere le proprietà di oggetti creati in precedenza
- Ogni oggetto javascript ha una proprietà .prototype a cui si può aggiungere un membro ed associare una funzione
- La modifica del prototipo può avvenire in qualunque momento nell'esecuzione del programma.

Esempio di prototype

```
Persona.prototype.welcome = function(){  
    alert("Benvenuto, "+this.name+"!");  
} // Aggiunta di una funzione al prototipo Persona  
mario.welcome(); // Utilizzo della funzione
```

JS: Usare prototype (1/3)

Supponiamo che facciate spesso gli stessi controlli, anche molto semplici, ad esempio che una certa stringa finisca con una certa sottostringa o un elemento abbia l'attributo id:

```
if (a.substr(-1*b.length) == b) {  
    // ok agisci  
}  
if (c.substr(-1*d.length) == d) {  
    // ok agisci  
}  
if (n.attributes && n.attributes['id']) {  
    // ok, agisci  
}  
if (m.attributes && m.attributes['id']) {  
    // ok, agisci  
}
```

JS: Usare prototype (2/3)

Ovviamente possiamo definire funzioni globali:

```
function endsWith(string,value) {  
    return string.substr(-1*value.length)==value  
}
```

```

function checkId(node) {
    return (node.attributes && node.attributes['id'])
}
if (endsWith(a,b)) {
// ok agisci
}
if (checkId(n)) {
// ok agisci
}

```

Ma questo viene considerato discutibile perché riempie lo spazio dei nomi globali di funzioni molto specifiche

JS: Usare prototype (2/3)

Un approccio molto comune è modificare il prototype della classe builtin relativa:

```

String.prototype.endsWith = function(value) {
    return this.substr(-1*value.length)==value
}

```

```

Node.prototype.checkId = function(node) {

```

```

    if (this.attributes && this.attributes['id'])

```

```

        return true

```

```

        return false
    }

```

```

}

```

```

if (a.endsWith(b)) {

```

```

// ok agisci
}

```

```

}

```

```

if (b.checkId()) {

```

```

// ok agisci
}

```

```

}

```

Closure e IIFE

La closure (1/2)

Javascript non ha protezione dei membri privati di un oggetto, ma sono tutti accessibili e manipolabili.

Ad esempio, definiamo una classe Counter con uno stato privato accessibile attraverso l'interfaccia data dalle funzioni inc() e dec():

```

Counter = function() {
    this.state = 0; // privata
    this.inc = function() { return this.state++ };
    this.dec = function() { return this.state-- };
}

```

```

var c = new Counter();

```

c.inc(); var d = c.inc() // d vale 2, corretto.

c.state = 7; var e = c.inc() // possibile!!! Inoltre

e vale 8!

Questo è molto grave: significa che l'interfaccia di inc() e dec() è solo apparente, non posso impedire l'accesso alle variabili private.

Closure (2/2)

- Abbiamo detto che in Javascript ci sono solo due

scope: quello globale e quello della funzione. Non è vero.

- C'è un terzo scope, detto closure, che è lo scope della funzione all'interno della quale viene definita la funzione.

- Ad esempio, una funzione che restituisce una funzione ha uno scope che è sempre accessibile alla funzione interna, ma non dal mondo esterno.

Closure (2/2)

Ottengo allora delle variabili interne veramente private:

```
Counter = function() {  
    var state = 0; // privata  
    return {  
        inc: function() { return ++state },  
        dec: function() { return --state }  
    }  
}  
  
var c = new Counter();  
c.inc(); var d = c.inc() // d vale 2, corretto.  
c.state = 7; var e = c.inc() // e vale 3, c.state  
è lecita ma inutile.
```

IIFE

Immediately Invoked Function Expression

- Una function expression immediatamente invocata (IIFE) è una funzione anonima creata ed immediatamente invocata.

- Serve sostanzialmente per fare singleton (oggetti non ripetibili) dotati di closure (e quindi di stato interno privato).

- L'oggetto JQuery è il risultato di un IIFE, e così MOLTISSIME librerie Javascript usano IIFE:

IIFE

Immediately Invoked Function Expression

```
var people = (function() {  
    var persone = [];  
    return {  
        add: function(p) { persone.push(p) },  
        lista: function() { return persone.join(',') }  
    }  
})()
```

- L'oggetto people, in questo caso, è un singleton con un array come stato interno e due metodi per accedere e modificare i valori.

- Per merito della closure, la variabile persone è accedibile da add e lista, ma NON dall'esterno.

- La coppia di parentesi alla fine invoca la funzione immediatamente e senza side effect.

www.unibo.it

Fabio Vitali
Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna
Fabio.vitali@unibo.it

Accessibilità

Che cos'è l'accessibilità?

- Per accessibilità si intende la possibilità che un sistema informatico (pagina web, applicazione, etc) possa essere utilizzato anche da persone con disabilità mediante tecnologie assistive. Si tratta di una pratica inclusiva di rimozione di tutte le “barriere” che impediscono l'utilizzo di un sistema mediante tali tecnologie.

Tipi di disabilità

- Visive: disabilità visive tra cui cecità, ipovisione e problemi nella distinzione dei colori.
- Motorie: difficoltà o impossibilità di usare le mani, inclusi tremori, lentezza muscolare, non completa padronanza della mobilità fine, ecc.
- Uditive: sordità e/o disturbi dell'udito.
- Convulsioni: attacchi foto-epilettici causati da effetti stroboscopici e/o lampeggianti.
- Cognitive e intellettive: disabilità dello sviluppo, difficoltà di apprendimento (dislessia, discalculia, ecc.) e disabilità cognitive (sindrome di Down, Alzheimer) di varie origini, che causano effetti negativi su memoria, attenzione, sviluppo, capacità logiche e di problem solving, etc.

Ma l'accessibilità è molto di più!

- Ma l'accessibilità non è vantaggiosa solo per le persone elencate nella diapositiva precedente, ma porta benefici a chiunque stia vivendo una disabilità permanente, temporanea o situazionale.
- Disabilità temporanea: un polso rotto rende impossibile il controllo del mouse.
- Per disabilità derivante dal contesto s'intende una situazione momentanea di disagio derivante da fattori esterni (ad esempio vista offuscata a causa dell'illuminazione solare, utilizzo di una sola mano quando si accompagna un bambino).

Perchè dovresti preoccupartene?

- motivi sociali ed etici;
- ragioni legali
- motivi economici
- passare questo esame ♦

Implicazioni sociali

- L'accessibilità è un diritto civile, riconosciuto dalla Convenzione delle Nazioni Unite sui diritti delle persone con disabilità (CRPD).
- Qualsiasi sito Web, applicazione o sistema non accessibile

può essere considerato una discriminazione, poiché impedisce a gruppi di persone di utilizzarlo.

- I sistemi inaccessibili incidono negativamente sulla dignità, l'autonomia, la piena ed efficace partecipazione, le pari opportunità e molto altro su un alta percentuale di popolazione.

Ragioni legali

- Sono in vigore leggi e regolamenti sull'accessibilità per imporre la creazione di contenuti accessibili in tutto il mondo (Australia, Stati Uniti, Canada, Unione Europea, Italia ed altri).
- Le aziende con siti Web e / o applicazioni inaccessibili possono essere (e vengono) citate in giudizio per questo (Stati Uniti, Australia, Canada, UE coming soon, etc).
- In Italia, i siti Web e le applicazioni mobili sviluppati per conto delle pubbliche amministrazioni o delle società che forniscono servizi per loro conto devono essere accessibili. Gli enti pubblici (comprese scuole, musei, università, ecc.) non possono acquistare soluzioni IT inaccessibili.

Motivi economici

- È stato stimato che il reddito disponibile al netto delle imposte per le persone in età lavorativa con disabilità negli Stati Uniti è di circa 490 miliardi di dollari, dello stesso ordine di grandezza di altri importanti segmenti di mercato, come gli afroamericani (\$ 501 miliardi) e gli ispanici (\$ 582 miliardi di euro). In poche parole, proprio a causa della loro inaccessibilità, i sistemi inaccessibili perdono un'importante fetta di mercato.
- Le persone con disabilità non sono un mercato isolato; poiché sono circondati da familiari e amici che riconoscono anche il valore di prodotti e servizi che soddisfano i bisogni di tutti.
- Essere citati in giudizio per motivi legati all'inaccessibilità è oneroso e in ogni caso, oltre alle spese legali, bisognerà investire per rendere il sistema accessibile, oltre a considerare il danno d'immagine.

Passare questo esame

- Anche l'accessibilità (sia teoria che pratica) sarà un argomento d'esame.
- L'accessibilità potrebbe essere anche parte del progetto, con modalità e requisiti ancora in fase di definizione.
- Nel mese di giugno faremo un esperimento di progettazione di siti accessibili, e cerchiamo volontari.
- Inoltre sono disponibili tesi di laurea su questo argomento per gli studenti interessati ad approfondirli

L'accessibilità nel World Wide Web

Verso linee guida ufficiali

Al fine di garantire che un sistema possa essere utilizzato da chiunque, indipendentemente dalle tecnologie assistive di cui abbia bisogno, è necessario introdurre linee guida più complesse.

Data la varietà di tecnologie disponibili al giorno d'oggi per implementare nuovi sistemi, tali linee guida dovrebbero essere definite in modo sufficientemente astratto da essere valide per ciascuna di esse, ma comunque abbastanza concreto da rendere possibile l'implementazione effettiva di tali raccomandazioni.

WCAG 2.1

(la 2.2 verrà rilasciata a breve)

Web Content Accessibility Guidelines (WCAG) 2.1 è una raccomandazione del W3C che contiene una serie di linee guida che ogni sistema deve soddisfare per essere considerato accessibile.

Tali linee guida sono organizzate attorno a 4 principi fondamentali. Per ciascuna linea guida, vengono forniti i cosiddetti success criteria (dichiarazioni verificabili), che specificano cosa testare e i risultati attesi, in modo indipendente da una specifica tecnologia.

La conformità a WCAG 2.1 può essere classificata in tre diversi livelli (A, AA, AAA) a seconda dei criteri di successo che il sistema soddisfa.

Si noti che vengono forniti documenti di supporto aggiuntivi (“techniques for WCAG 2.1”) per offrire esempi pratici su come soddisfare i criteri di successo utilizzando specifiche tecnologie (implementazioni concrete).

WCAG 2.1 principles

questi principi vengono raggruppati sotto l'acronimo POUR.

The guidelines and Success Criteria are organized around the following four principles, which lay the foundation necessary for anyone to access and use Web content. Anyone who wants to use the Web must have content that is:

- Perceivable. Information and user interface components must be presentable to users in ways they can perceive. This means that users must be able to perceive the information being presented (it can't be invisible to all of their senses)
- Operable. User interface components and navigation must be operable. This means that users must be able to operate the interface (the interface cannot require interaction that a user cannot perform).

WCAG 2.1 principles II

Anyone who wants to use the Web must have content that is:

- Understandable. Information and the operation of user interface must be understandable. This means that users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding).

– Robust. Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies. This means that users must be able to access the content as technologies advance (as technologies and user agents evolve, the content should remain accessible).

- [Source: Introduction to understanding WCAG 2.1]

Esempi pratici (1/2)

- La codifica di un sito utilizzando HTML semanticamente valido, fornendo rappresentazioni testuali equivalenti per immagini e di testi dei link significativi aiutano gli utenti non vedenti che utilizzano screen reader e display Braille.
- Testo e immagini di grandi dimensioni e / o ingrandibili semplificano la lettura e la comprensione del contenuto da parte degli utenti con problemi di vista.
- Utilizzare link colorati e sottolineati o altrimenti differenziati assicura che gli utenti daltonici possano notarli.
- Link e aree cliccabili di dimensioni più ampie aiutano gli utenti che non possono controllare con precisione un mouse (o utilizzano il sito Web con un dispositivo touchscreen).

Esempi pratici (2/2)

- Scrivere codice che non ostacoli la navigazione mediante la sola tastiera o il controllo interruttori consente agli utenti che non possono utilizzare un mouse di poter usufruire comunque della pagina Web.
- Fornendo i sottotitoli, una trascrizione e / o la versione in lingua dei segni del sonoro di un video, gli utenti non udenti e ipovedenti potranno comprenderne i contenuti.
- Quando gli effetti stroboscopici e lampeggianti vengono evitati o resi opzionali, si diminuiscono i rischi per gli utenti che soffrono di convulsioni e/o attacchi foto-epilettici causate da questi effetti.
- Scrivendo contenuti didattici in un linguaggio semplice e illustrandoli con diagrammi e animazioni, essi risulteranno maggiormente comprensibili per gli utenti con difficoltà di apprendimento.

Un po' di esempi in HTML

Lingua del documento

- In HTML, l'attributo «lang» specifica la lingua in cui è scritto il documento. Specificare un valore corretto per tale attributo è importante per l'accessibilità: gli screen reader, per esempio, possono usare l'informazione per attivare dizionari di pronunce specifiche per quella lingua.
- L'attributo lang ammette come valori un «language tag» secondo lo standard BCP47 («en» per inglese, «it» per italiano, ad esempio), o la stringa vuota; questo secondo caso è da evitare! Può essere usato su ogni elemento HTML; se un elemento non lo specifica, la lingua usata è quella dell'elemento padre (se esiste).

Esempio

L'esempio mostra una pagina web in italiano che contiene al suo interno una citazione in inglese.

```
<!doctype html>
<html lang="it">
[...]
<p>William Faulkner ha detto:</p>
<blockquote lang="en">
The past is never dead. It's not even past.
</blockquote>
[...]
</html>
```

Descrivere le immagini

Esistono immagini decorative (utilizzate unicamente a fine estetico) ed immagini non decorative (con valore funzionale)

- Ad ogni immagine «non decorativa» (ovvero ogni immagine che non svolga esclusivamente funzioni estetiche nella pagina), deve essere associata una breve descrizione che consenta di comprenderne a pieno il contenuto anche senza vederla. Oltre che dai motori di ricerca, questa informazione è indispensabile per rendere la pagina accessibile: gli screen reader, per esempio, la utilizzano per permettere alle persone non vedenti di comprendere cosa è mostrato nell'immagine.

Descrivere le immagini in HTML

- Esistono vari modi, ma si consiglia l'utilizzo dell'attributo «alt» al cui interno si può inserire una stringa di testo.
``
- Scrivere una «descrizione alternativa» per un'immagine non è semplice!

Immagini decorative

- Se l'immagine è decorativa e viene caricata mediante il tag «img», allora occorre specificare un attributo «alt» il cui valore sia la stringa vuota;
- Questo indica alle tecnologie assistive che quell'immagine va trattata, appunto, come immagine decorativa.
- Non specificare alcun valore per l'attributo alt è sempre un errore!

Sì: ``

No: ``

Intestazioni

- Per rendere una pagina Web accessibile le intestazioni

sono fondamentali! In HTML ne esistono 6 livelli (ordinati gerarchicamente dal più importante al meno importante), che sono rappresentati dai tag «h1», «h2», «h3», «h4», «h5», e «h6».

Le intestazioni vanno annidate gerarchicamente in modo corretto: è un errore, per esempio, utilizzare «h3» se in precedenza non è stato utilizzato il tag «h2» per l'intestazione di livello superiore.

Esempio corretto di annidamento delle intestazioni

[...]

```
<h1>1. Lezione di tecnologie Web sull'accessibilità</h1>
<p>Testo, anche più di un paragrafo.</p>
<h2>1.1. Introduzione</h2>
<p>Testo, anche più paragrafi</p>
<h2>1.2. Argomenti</h2>
<p>Testo opzionale, anche più paragrafi.</p>
<h3>1.2.1. Lingua del documento</h3>
<p>Testo, anche più paragrafi</p>
<h3>1.2.2. Descrivere le immagini</h3>
<p>Testo, anche più paragrafi</p>
<h2>1.3. I form</h2>
<p>Testo, anche più paragrafi</p>
[...]
```

Regole empiriche per le intestazioni

- Immaginare di dover attribuire delle numerazioni gerarchiche rende intuitivo capire quali livelli di intestazione usare (ed evitare di saltarne qualcuno!).
- Se l'aspetto grafico del livello di intestazione corretto non è adatto possiamo modificarlo tramite CSS.
- Ogni pagina deve sempre contenere al massimo un livello di intestazione 1 (il titolo della pagina!).
- Se un frammento di testo viene rappresentato come un sottotitolo (centrato, con colore diverso, font più grande, etc.) per metterlo in evidenza dal resto del testo, probabilmente usare un livello di intestazione può essere una buona idea.

Tabelle

- Non tutte le celle di una tabella sono uguali. Alcune di esse, infatti, fungono da intestazioni per righe e colonne; in un certo senso, queste celle «descrivono» la funzione dei dati a cui si riferiscono.
- Non è sufficiente modificarne l'aspetto grafico in modo da metterle in risalto: affinché le tabelle siano accessibili queste relazioni devono essere «esposte» semanticamente usando i tag appropriati.

Esempio di tabella

```
[...]
<table>
<caption>Orari di apertura dei locali</caption>
<thead>
<tr>
<th>Locale</th><th>Lunedì</th><th>Martedì</th>
<th>Mercoledì</th><th>Giovedì</th><th>Venerdì</th><th>Sabato</th><th>Domenica</th>
</tr>
</thead>
<tbody>
<tr>
<th>Pizza pazza</th><td>Chiuso</td><td>12:30 - 23:30</td><td>12:30 - 23:30</td><td>12:30 - 23:30</td><td>Chiuso</td><td>Chiuso</td> </tr>
<tr>
<td>La bistecca</td><td>Chiuso</td><td>11:30 - 15:30, 19:30 - 22:30</td><td>11:30 - 15:30, 19:30 - 22:30</td><td>11:30 - 15:30, 19:30 - 22:30</td><td>11:30 - 15:30, 19:30 - 22:30</td><td>12:00 - 15:00, 19:00 - 23:00</td>
</tr>
</tbody>
</table>
[...]
```

Etichettare i campi di un form

- Una stringa di testo situata immediatamente a sinistra (o subito sopra) di un campo di un form è la sua etichetta.
- Questa relazione è ovvia, ma affinché la pagina sia accessibile deve essere «esposta» anche semanticamente con appositi tag e attributi HTML.

Esempio

```
[...]
<form>
<label>Nome:</label>
<input type="text">
</label>
<label for="lastname">Cognome:</label>
<input id="lastname" type="text" required>
<label for="genere">Genere:</label>
<select name="genere" id="genere">
<option value="NA">-- Scegliere un opzione --</option>
<option value="m">Maschile</option>
<option value="f">Femminile</option>
<option value="other">Altro, o preferisco non dirlo</option>
</select>
<input type="text" name="subject" title="Oggetto del messaggio"
```

```

placeholder="Inserisci l'oggetto del messaggio">
<label for="txt">Testo del messaggio:</label>
<textarea id="text" cols="30" rows="10" required></textarea>
<input type="checkbox" id="check">
<label for="check">Accetto i termini di servizio</label>
<button type="submit">Invia modulo!</button>
<input type="reset" value="Pulisci i campi">
</form>
[...]

```

Osservazioni sui campi dei form

- Esistono più modi (equivalenti tra loro a livello di accessibilità) per associare un etichetta ad un campo;
- Ogni etichetta deve essere associata ad un campo; ogni campo deve essere associato ad un'etichetta;
- Per indicare che la compilazione di un campo sia obbligatoria possiamo usare l'attributo «required»;
- I pulsanti (tag button) non richiedono etichetta; le tecnologie assistive usano il testo al loro interno (o l'attributo alt di una eventuale immagine);
- Alcuni tag input (di tipo «submit» o «reset», per esempio) usano come accessibility name il valore dell'attributo «value».

Fieldset

- E' possibile esprimere semanticamente un «raggruppamento» tra campi di un form, per esempio se essi sono riconducibili allo stesso macroargomento (il gruppo «dati personali» potrebbe comprendere i campi per nome, cognome e data di nascita, per esempio).
- Il loro utilizzo è opzionale, almeno che il form non sia molto complesso (perché ne facilitano la navigazione), o si debbano usare i radio button!

Esempio di radiogroup

```

[...]
<fieldset>
<legend>Cottura della bistecca:</legend>
<input id="radio1" type="radio">
<label for="radio1">Al sangue</label>
<input id="radio2" type="radio">
<label for="radio2">Cottura media</label>
<input id="radio3" type="radio">
<label for="radio3">Ben cotta</label>
</fieldset>
[...]

```

Struttura della pagina

- In HTML 5 sono stati introdotti diversi tag che permettono di attribuire un valore semantico a diverse «aree» della

pagina Web. Se usate correttamente, queste «annotazioni» possono contribuire a rendere la pagina più accessibile.

- In gergo a volte si usa il termine «landmark», giacché essi possono costituire dei punti di riferimento quando si naviga la pagina mediante tecnologie assistive.

Esempio di struttura

- <!DOCTYPE html>
- <html lang="en">
- <head>
- <meta charset="utf-8">
- <title>native landmarks examples</title>
- </head>
- <body>
- <header class="header">
-
- <p>Welcome to Saharian's official website, a place where you can find a lot of lorem ipsum to test accessibility landmarks!
- </p>
- </header>
- <nav class="nav">
- <h2>Fictional menu</h2>
-
-
- Home
-
-
- About Saharian (page does not exist)
-
-
- About us (page does not exist)
-
-
- </nav>
- <div class="search" role="search" aria-labelledby="searchblocktitle">
- <h2 id="searchblocktitle">Search this website</h2>
- <form action="" method="get">
- <label for="query">Search for:</label>
- <input type="search" id="query" name="q" aria-describedby="searchblockhelp">
- <p id="searchblockhelp">Enter the terms you wish to search for... Just be ware that this is a fictional web page
- to test landmark roles, therefore searchig does not work at all.</p>
- <input type="submit" value="Search">
- </form>
- </div>
- <main class="main">
- <h1>HomePage</h1>

- <p>Hello! Welcome to the HomePage of Saharian, a pretty useless website if you exclude the fact that this page can
- be used to test different aria landmarks.</p>
- <p>In deed, this has been created for the sole purpose of containing as many landmarks as the specification can
- provide, really. Not for any other purpose.</p>
- <h2>Some facts</h2>
- <p>Here we could put anything. For example:</p>
-
- a list
- of non sense elements.
-
- <p>And why not wrapping <code>a simple sentence</code> as source code? Don't do that!, let me emphasize

Sugli esempi visti finora

- Gli esempi visti finora sono particolarmente semplici, ma rendono l'idea del fatto che le raccomandazioni teoriche delle linee guida sull'accessibilità hanno riscontri estremamente pratici;
- In rete esistono svariate risorse estremamente utili per comprendere come rendere accessibile qualsiasi elemento di una pagina Web, la più importante delle quali è sicuramente «Techniques for WCAG 2.1».

Techniques for WCAG 2.1

E' una raccolta di tutorial, esempi e guide che illustrano come soddisfare le WCAG 2.1 creando interfacce e contenuti accessibili con specifici linguaggi (HTML, CSS, JavaScript, ETC.). Si dividono in vari gruppi, di cui i più importanti sono:

- client-side script techniques, che spiegano come rispettare le WCAG 2.1 ;
- CSS Techniques, che illustrano come rispettare le WCAG 2.1 quando si usano I fogli di stile CSS;
- common failures, che illustrano errori comuni e forniscono le alternative "corrette";
- general techniques, che illustrano come rendere accessibile un'applicazione in fase di design;
- HTML techniques, che illustrano aspetti specifici relativi a HTML/XHTML;
- PDF techniques, che spiegano come "rendere accessibili" documenti PDF (complicato!);
- Server-Side script techniques, che spiegano come migliorare l'accessibilità mediante I linguaggi server-side.

Creare un sistema
accessibile

Un processo in più passaggi

Per definizione, è chiaro che nello sviluppo di un sistema accessibile sono coinvolte più fasi del suo ciclo di vita:

- Design: decisioni importanti per il progetto devono essere

prese ancor prima di scrivere la prima riga di codice; è molto più semplice rendere accessibile un'interfaccia facile da usare piuttosto che una complessa.

- Sviluppo: se non effettuata correttamente, l'implementazione del design può introdurre problemi di accessibilità.
- Creazione dei contenuti: anche le informazioni inserite nel sistema devono essere accessibili, altrimenti tutti gli sforzi fatti nelle fasi precedenti sono vani!

Design

Ogni decisione di progettazione ha il potenziale per includere o escludere dei clienti. Il **design inclusivo** è una filosofia di design che massimizza il contributo fornito dalla comprensione della diversità degli utenti per prendere tali decisioni, consentendo così di eliminare gli ostacoli che ne impediscono l'utilizzo per quante più persone possibili.

La diversità degli utenti riguarda le differenti capacità, esigenze, aspirazioni, etc.

Esempio

- Un'interfaccia viene concepita per supportare solo ed esclusivamente l'interazione mediante il drag-and-drop degli elementi con il mouse.
- SBAGLIATO: occorre progettare anche «esperienze utente» alternative che prevedano l'utilizzo di comandi da tastiera (o meccanismi sostitutivi per l'interazione).

Sviluppo

- Anche la fase di implementazione può introdurre ostacoli. Il progetto deve essere implementato sfruttando le tecnologie esistenti note per essere accessibili o adottando tutti i meccanismi necessari per renderle tali.
- Sono disponibili documenti di supporto e standard tecnici che spiegano come rispettare WCAG 2.1 in scenari specifici, ad es. Tecniche per WCAG 2.1

Contenuti

- Anche gli sforzi fatti per creare il sistema più accessibile svaniscono quando i suoi contenuti non vengono creati per essere accessibili. Gli esempi includono:
 - allegare documenti inaccessibili (file PDF scansionati senza OCR),
 - screenshot privi di descrizioni alternative che consentano di comprenderne il contenuto,
 - scrivere testi che non possono essere compresi da tutti (ad es. Usare informazioni che possono essere correlate solo a un senso),
 - non fornire abbastanza contesto per la comprensione del contenuto in caso di disabilità.

Livelli di conformità WCAG 2.1

Esistono tre livelli di conformità al WCAG:

A:

- livello più basso di conformità;
- rimuove le principali barriere per cecità, sordità e disabilità motorie.

AA:

- livello successivo di conformità (include A);
- rimuove le principali barriere per gli utenti ipovedenti;
- offre un piccolo aiuto per le disabilità cognitive.

AAA:

- massimo livello di conformità (include A e AA);
- non è consigliabile che sia richiesto come politica generale per interi siti perché non è possibile soddisfare tutti i requisiti di livello AAA per alcuni contenuti.

Come testiamo la conformità?

- Le verifiche di conformità alle linee guida sull'accessibilità possono essere automatizzate solo in parte, perciò è necessario comunque eseguire test manuali per valutarla in modo esaustivo ed attendibile.
- Consideriamo un semplice criterio di successo: tutte le immagini non decorative dovrebbero avere un testo alternativo descrittivo.

Controllare che a un'immagine sia associato un testo alternativo è banale, ma assicurarsi che sia descrittivo per quell'immagine non lo è (ancora). Anche distinguere quali immagini sono decorative e quali no può essere complicato, (lo è per l'uomo, figurarsi per un sistema automatizzato).

Una nota sugli strumenti di test automatizzati dell'accessibilità

Si noti che i primi strumenti di test per l'accessibilità automatizzati di prima generazione considerano solo il markup originale della pagina Web, quindi non possono rilevare tonnellate di errori e non sono adatti per le applicazioni AJAX.

D'altro canto, gli strumenti più recenti testano il DOM effettivo di una pagina Web, offrendo così risultati più accurati.

Miti e leggende I

Mito: un sito Web accessibile è brutto e noioso.

Fatto: è possibile implementare siti Web sofisticati e ben realizzati, ma accessibili! Un design accessibile è più “usabile”, ma questa è un'altra storia!

Mito: l'accessibilità è costosa!

Fatto: sì, ma solo se viene considerata a posteriori. La riparazione di progetti inaccessibili richiede molti più sforzi, tempo e conoscenza (quindi denaro) rispetto alla creazione del suo equivalente accessibile. In ogni caso, il risultato finale potrebbe non essere così buono come

sarebbe potuto essere considerando l'accessibilità fin dal principio.

Miti e leggende II

Mito: l'accessibilità avvantaggia troppo poche persone.

Fatto: si stima che circa il 10% della popolazione mondiale abbia una disabilità che influisce sull'uso di Internet. Circa 700 milioni di persone sono troppo poche? A queste occorre aggiungere le persone colpite da disabilità temporanee e derivanti dal contesto! E piaccia o no, con l'età diminuiscono l'udito, la vista e la destrezza, cambiando la nostra capacità di usare Internet.

Mito: i siti Web accessibili sono statici.

Fatto: siti Web altamente dinamici e sofisticati (anche applicazioni desktop-like) possono essere resi accessibili, è necessaria solo maggiore attenzione. Benvenuti in WAI-ARIA!

WAI-ARIA

wai aria nasce in seguito alla diffusione di ajax, con la quale le applicazioni web diventano più complicate e l'aggiornamento di aree dello schermo non coincide più necessariamente con la pagina che viene ricaricata.

La specifica WAI-ARIA

- Accessible Rich Internet Applications (WAI-ARIA) 1.1 è una raccomandazione del W3C che fornisce un'ontologia di ruoli, stati e proprietà che definiscono gli elementi dell'interfaccia utente e possono essere utilizzati per migliorare l'accessibilità e l'interoperabilità dei contenuti e delle applicazioni Web. Progettato per consentire a un autore di comunicare in modo appropriato comportamenti dell'interfaccia utente e informazioni strutturali alle tecnologie assistive attraverso il markup.
- È uno strumento fondamentale per rendere accessibili le applicazioni Web desktop-like, in quanto vi sono (molti) widget avanzati (barre dei menu, visualizzazioni ad albero, barre degli strumenti, ecc.) ancora non disponibili in HTML.

Ruoli, proprietà e stati

È possibile usare WAI-ARIA sfruttando attributi specifici da applicare su qualsiasi elemento HTML:

- l'attributo role, che specifica il ruolo (semantica) dell'elemento (button, checkbox, tree, tablist, tab, ecc);
 - proprietà (aria-label, aria-labelledby, aria-valuenow, ecc.), attributi essenziali per la natura di un dato oggetto o che rappresentano un valore di dati associati ad esso. Una modifica di una proprietà può avere un impatto rilevante sul significato o sulla presentazione di un oggetto;
 - stati (aria-checked, aria-selected, ecc.), proprietà dinamiche che esprimono le caratteristiche di un oggetto che possono cambiare in risposta ad azioni intraprese dall'utente o a processi automatizzati.
- Gli stati non influenzano la natura essenziale dell'oggetto, ma rappresentano i dati associati all'oggetto o alle possibilità di interazione dell'utente con esso.

Tipi di ruoli

I ruoli facenti parte della specifica WAI-ARIA possono essere raggruppati in:

- landmark, che consentono di «annotare» punti di riferimento nella pagina. Spesso esistono tag HTML equivalenti;
- widget per cui esistono analoghi elementi in HTML: è questo il caso di «checkbox», «radio», «radiogroup» o «table», per esempio;
- semantica di widget non rappresentati da alcun elemento in HTML; è questo per esempio il caso di «tab», «tablist», «tabpanel», «tree», «treeitem», etc. In questo caso la specifica WAI-ARIA diventa l'unico modo disponibile per creare una rappresentazione accessibile di quel widget.

Regole di ARIA

è difficile definire un insieme di regole che se seguite portano certamente ad un buon utilizzo di aria, però una serie di linee guida possono essere:

- Non usare ARIA a meno che non sia necessario; se esiste un elemento HTML o un attributo nativo con la semantica ed il comportamento necessari, usa quello. Eccezioni:
 - se la funzionalità è disponibile in HTML ma non è implementata o la sua implementazione non fornisce supporto di accessibilità;
 - Se i vincoli di progettazione visiva escludono l'uso di un particolare elemento nativo, in quanto non può essere “stilizzato” come necessario.
- Non modificare la semantica nativa, a meno che non sia necessario. Si noti che se un elemento non interattivo (ad es. span) viene utilizzato come interattivo (ad es. pulsante), è compito dello sviluppatore implementare il comportamento appropriato utilizzando JavaScript.
- tutti i controlli interattivi ARIA devono essere utilizzabili con la tastiera. Il supporto deve essere implementato dallo sviluppatore.
- Non utilizzare role = "presentation" o aria-hidden = "true" su un elemento focalizzabile, altrimenti il focus potrebbe essere posizionato in punti su cui non potrebbe esserlo, causando effetti non determinati.
- Tutti gli elementi interattivi devono avere un accessible name.

Un ruolo è una promessa

Quando si usa un ruolo della specifica ARIA si sta «facendo una promessa» agli utenti di tecnologie assistive, dichiarando che quel'elemento si comporta secondo la semantica di ciò che rappresenta il ruolo che gli è stato attribuito. Tali comportamenti, però, devono essere implementati dal programmatore (spesso tramite JavaScript), seguendo la documentazione disponibile (e.g. WAI-ARIA Authoring Practices).

Miglioriamo un form con ARIA

- Gli attributi della specifica WAI-ARIA possono essere

utilizzati insieme ad elementi HTML nativi per migliorarne l'accessibilità. Riprendiamo il form visto in precedenza, ed associamo ad alcuni campi anche un testo di aiuto che spieghi all'utente come compilarli.

Campi con descrizioni

```
[...]
<form>
<label>Nome:
<input type="text">
</label>
<label for="lastname">Cognome:</label>
<input id="lastname" type="text" required
aria-describedby="lastname-help">
<p id="lastname-help">Questa è la descrizione aggiuntiva di un campo obbligatorio, da utilizzare per un testo d'aiuto all'utente.</p>
<label for="genere">Genere:</label>
<select name="genere" id="genere">
<option value="NA">-- Scegliere un opzione --</option>
<option value="m">Maschile</option>
<option value="f">Femminile</option>
<option value="other">Altro, o preferisco non dirlo</option>
</select>
<input type="text" name="subject" title="Oggetto del messaggio"
placeholder="Inserisci l'oggetto del messaggio">
<label for="txt">Testo del messaggio:</label>
<textarea id="text" cols="30" rows="10" required></textarea>
<input type="checkbox" id="check">
<label for="check">Accetto i termini di servizio</label>
<button type="submit">Invia modulo!</button>
<input type="reset" value="Pulisci i campi">
</form>
```

[...]

Le live region

- Sono un aspetto particolarmente importante della specifica WAI-ARIA. Si tratta di un meccanismo che consente di «informare» le tecnologie assistive (specialmente screen readers) del fatto che alcune porzioni della pagina cambiano dinamicamente, e che gli aggiornamenti devono essere «comunicati» all'utente prima possibile.

- Si pensi per esempio ad un timer: il testo che indica il tempo residuo deve essere comunicato costantemente affinché il timer sia accessibile!

Attributi per le live region

- Si usano gli attributi aria-live, che può assumere i valori «assertive»(se lo screen reader sta leggendo altro lo interrompe e comunica subito il cambiamento), «polite» (se lo screen reader sta leggendo altro attende che abbia finito)

oppure «off», e «aria-atomic» (che può assumere valore «true» o «false») su un qualsiasi elemento

HTML con role «region».

- In alternativa esistono diversi ruoli appositamente predisposti per scopi specifici: «status», «timer», «log».
- Se mediante JavaScript viene aggiornato il DOM di una live region, le tecnologie assistive comunicheranno l'aggiornamento in modo appropriato secondo gli attributi usati.

Landmark con WAI-ARIA I

- Come detto, può capitare che alcuni attributi della specifica WAI-ARIA siano del tutto equivalenti ad altri elementi e attributi già presenti in HTML. Vediamo la stessa struttura per una pagina Web analizzata in precedenza, ma stavolta implementata solo con attributi della specifica WAI-ARIA.

Landmark con WAI-ARIA II

- <!DOCTYPE html>
- <html lang="en">
- <head>
- <meta charset="utf-8">
- <title>native landmarks examples</title>
- </head>
- <body>
- <div role="banner" class="header">
-
- <p>Welcome to Saharian's official website, a place where you can find a lot of lorem ipsum to test aria landmarks!
- </p>
- </div>
- <div role="navigation" class="nav">
- <h2>Fictional menu</h2>
-
-
- Home
-
-
- About Saharian (page does not exist)
-
-
- About us (page does not exist)
-
-
- </nav>
- <div class="search" role="search" aria-labelledby="searchblocktitle">
- <h2 id="searchblocktitle">Search this website</h2>
- <form action="" method="get">
- <label for="query">Search for:</label>
- <input type="search" id="query" name="q" aria-describedby="searchblockhelp">

- <p id="searchblockhelp">Enter the terms you wish to search for... Just be ware that this is a fictional web page
 - to test landmark roles, therefore searchig does not work at all.</p>
 - <input type="submit" value="Search">
 - </form>
 - </div>
 - <div role="main" class="main">
 - <h1>HomePage</h1>
 - <p>Hello! Welcome to the HomePage of Saharian, a pretty useless website if you exclude the fact that this page can
 - be used to test different aria landmarks.</p>
 - <p>In deed, this has been created for the sole purpose of containing as many landmarks as the specification can
 - provide, really. Not for any other purpose.</p>
 - <h2>Some facts</h2>
 - <p>Here we could put anything. For example:</p>
 -
 - a list
 - of non sense elements.
 -
 - <p>And why not wrapping <code>a simple sentence</code> as source code? Don't do that!, let me emphasize
 - that. <i>Don't do that!</i>, maybe italics works better for you.</p>
 - <h2>Some more facts</h2>
- Un ruolo è una promessa...
- ...E ogni promessa è debito. Ma cosa vuol dire? Vediamolo con un esempio pratico: implementiamo un radiogroup usando la specifica WAI-ARIA anziché gli elementi nativi forniti da HTML. Questo dovrebbe convincervi del fatto che conviene usare la specifica ARIA solo quando (e se) è strettamente indispensabile!
- Leggiamo la specifica di radiogroup
- Ci viene in aiuto il documento «WAI-ARIA 1.1 authoring practices», che illustra nel dettaglio cosa un utente di tecnologie assistive si aspetti da ogni implementazione di questo design pattern. Per brevità concentriamoci sui comandi da tastiera:
- Pressing «tab» moves focus to the checked radio button in the radiogroup. If a radio button is not checked, focus moves to the first radio button in the group.
 - If the radio button with focus is not checked, pressing “spacebar” changes the state to checked.
 - Pressing “Down or Right arrow” moves focus to and checks the next radio button in the group. If focus is on the last radio button, moves focus to the first radio button. Always changes the sate of the previously checked radio button to “unchecked”.
 - Pressing “Up or Left arrow” does the opposite.
- Markup di Radiogroup con aria
- [...]

```
<div id="rg1" class="radiogroup" role="radiogroup" aria-labelledby="radiogroup1-title">
<h3 id="radiogroup1-title">Steak doneness:</h3>
<span class="radio" role="radio" aria-checked="true"
tabindex="0">rare</span>
<span class="radio" role="radio" aria-checked="false"
tabindex="-1">medium-rare</span>
<span class="radio" role="radio" aria-checked="false"
tabindex="-1">medium</span>
<span class="radio" role="radio" aria-checked="false"
tabindex="-1">medium-well</span>
<span class="radio" role="radio" aria-checked="false"
tabindex="-1">well-done</span>
</div>
[...]
```

Ma cos'è tabindex?

- Quando si naviga una pagina Web mediante la tastiera esiste il cosiddetto «tabbing order», ovvero l'ordine in cui viene posizionato il focus sugli elementi quando si preme «tab» (o «shift+tab» per l'ordine inverso).
- Tale ordine viene determinato da vari fattori tra cui la natura degli elementi, la loro posizione nel dom, nonché il valore dell'eventuale attributo “tabindex”.

Valori di tabindex

- L'attributo tabindex può avere tre valori:
 - «-1», che indica che l'elemento può ricevere il focus esclusivamente mediante apposite istruzioni JavaScript; i comandi da tastiera ignorano l'elemento (in gergo, è «fuori dal tabbing order»);
 - Valore «0», il valore di default: l'elemento può ricevere il focus sia mediante i comandi da tastiera che attraverso istruzioni JavaScript. La sua posizione nel «tabbing order» deriva dalla sua posizione nel DOM della pagina;
 - Valore positivo: l'elemento può ricevere il focus sia mediante comandi da tastiera che tramite istruzioni JavaScript. La sua posizione nel «tabbing order» è influenzata dal valore di tabindex;
- Si noti che tabindex influenza la propagazione di eventi JavaScript relativi a comandi da tastiera (e.g. 'keydown') verso l'elemento padre;
- Di fatto, a causa di differenze implementative nei browser è sconsigliatissimo utilizzare valori di tabindex positivi maggiori di 0.

Perché usiamo tabindex?

La corretta implementazione di molti design pattern che sfruttano la specifica WAI-ARIA necessita la scrittura di codice JavaScript per gestire appositi comandi da tastiera (che sono specifici per ogni widget) in modo adeguato. Sfruttare le proprietà di «tabindex» è la chiave per implementarli attraverso due tecniche:

- «rowing tabindex», in cui il «padre» gestisce il tabindex di tutti i suoi figli; in ogni momento, solo uno di essi avrà tabindex uguale a 0.
- «aria-activedescendant»: sfruttando un particolare attributo della

specifica WAI-ARIA, solo il «padre» avrà tabindex uguale a 0, mentre i figli avranno sempre «tabindex» negativo; l'attributo «aria-activedescendant» conterrà sempre un ID univoco che si riferisce al figlio «selezionato», qualunque cosa significhi per il widget da implementare (radiogroup, menubar, tablist, ETC. hanno definizioni diverse di «selezione» che derivano dalla loro semantica).

ATAG

ATAG 2.0

Uno strumento di authoring è un qualunque strumento che permetta di produrre del contenuto (editor di testo, compilatori, software di produzione musica, ecc).

Le linee guida per l'accessibilità degli strumenti di authoring (ATAG) 2.0 sono una raccomandazione del W3C creata appositamente per garantire l'accessibilità degli strumenti di authoring come:

- strumenti per la creazione di pagine Web (ovvero editor HTML WYSIWYG);
- software per la generazione di siti Web (ovvero sistemi CMS);
- software che convertono i contenuti in tecnologie web;
- strumenti di authoring multimediale;
- siti Web i cui utenti possono aggiungere contenuti (ad es. social network).

ATAG 2.0 II

- La specifica è divisa in due parti principali:
 - parte a, che consiste nel rendere accessibili gli strumenti di creazione in modo che le persone con disabilità possano utilizzarli;
 - parte b, che consiste nell'aiutare gli autori a produrre contenuti accessibili, ovvero contenuti conformi a WCAG 2.1.
- Come in WCAG 2.1, in ATAG troviamo linee guida organizzate attorno a principi chiave, la cui soddisfazione può essere valutata sulla base della conformità ai criteri di successo sugli stessi 3 livelli (A, AA, AAA).

ATAG 2.0 principles

Part A principles:

A1. The authoring tool user interface follows applicable accessibility guidelines.

A2. Editing-views are perceivable.

A3. Editing-views are operable

A4. Editing-views are understandable

Part B principles:

B1. Fully automatic processes produce accessible content.

B2. Authors are supported in producing accessible content

B3. Authors are supported in improving the accessibility of existing content

B4. Authoring tools promote and integrate their accessibility features.

Contenuti multimediali

- Anche contenuti aggiuntivi come file audio e video possono presentare problemi di accessibilità, ed esistono

accorgimenti per evitare tli problemi.

- Accompagnare video con opportune audiodescrizioni e sottotitoli può far sì che essi siano maggiormente fruibili sia per i non vedenti che per i non/ipo udenti.
- Associare al file audio di un podcast la sua trascrizione lo renderà fruibile anche da parte di utenti non udenti.

Post social e documenti PDF

- Col passare del tempo sta diventando possibile la creazione di contenuti accessibili anche sui social network; le principali piattaforme, infatti, oramai supportano funzioni quali l'inserimento di testo alternativo per le immagini o la sottitolazione dei video.

- Per i documenti PDF, invece, la situazione resta complessa.

Nonostante esistano opportuni standard di riferimento (PDF/UA), gli strumenti che consentono di produrre documenti PDF realmente accessibili sono pochissimi, sono molto complessi e richiedono quantità di lavoro aggiuntivo decisamente elevate.

Accessibility essentials

Tip: su cosa concentrarsi?

- Correttezza e conformità agli standard e alle linee guida per l'accessibilità del markup;
- navigazione e operabilità mediante tastiera;
- Gestione del focus;
- Accessibilità dei contenuti (descrizione per le immagini, testi di link ed etichette, etc.);

Risorse (molto) utili

- Web Content Accessibility Guidelines (WCAG) 2.1 | W3C
- understanding WCAG 2.1 | W3C
- Techniques for WCAG 2.1
- Accessible Rich Internet applications (WAI-ARIA 1.1)
- Design patterns | WAI-ARIA 1.1 authoring practices
- Articles | WebAIM
- Accessibility resources and code examples (by Deque Systems)

Domande?

www.unibo.it

Vincenzo Rubano

Dipartimento di Informatica – Scienze e Ingegneria

Alma mater – Università di Bologna

vincenzo.rubano@unibo.it

Extra

Suggerimenti e trucchi

- Ecco alcuni suggerimenti che possono essere utili per l'implementazione del progetto e più in generale creare applicazioni web accessibili.

Suggerimento I: mistrust the authority

- Se si utilizza un framework di componenti per l'interfaccia utente (bootstrap, Angular-material, element-ui, ecc.), non dare per scontato che tali componenti siano accessibili.

Verificare sempre la loro accessibilità ed eventualmente aggirare i problemi riscontrati. Scegliere un framework diverso, se necessario.

Suggerimento II: test con un lettore di schermo

- Esistono prove evidenti del fatto che la navigazione di un'applicazione Web e l'interazione con essa tramite uno screen reader offrono la valutazione più completa della sua accessibilità.

- Essa non copre tutti gli aspetti da verificare, ma è comunque un buon punto di partenza!

Suggerimento III: scegli uno screen reader facile da usare per i test

- Se decidi di testare la tua applicazione web con uno screen reader, assicurati di sapere come usarlo (funzionalità principali, scorciatoie da tastiera, ecc.).

- Sembra ovvio ma, assolutamente, assicurati di sapere come disabilitarlo: gli screen reader cambiano spesso le modalità di interazione con il computer, quindi possono essere considerati invasivi; assicurati di sapere come gestirli.

- Da questo punto di vista Chrome Vox è un'ottima opzione, in quanto offre un ottimo tutorial interattivo introduttivo ed è un'estensione del browser.

Suggerimento III: test automatizzati

- Esamina sempre i problemi segnalati da strumenti automatici, poiché in alcuni casi potrebbero non essere errori reali.

- Distinguere tra problemi segnalati come errori e problemi segnalati come potenziali errori (potrebbero esserlo o no, ma gli strumenti automatizzati non sono in grado di dedurre una risposta).

- In caso di incertezza meglio concentrarsi sugli errori, che vanno risolti.

Cheatsheet di design accessible

- Come si dovrebbe progettare l'interfaccia del progetto, e in generale qualsiasi interfaccia web, per massimizzare le possibilità che essa sia accessibile? Scopriamolo insieme.

Distinguere widget e design pattern

- Identificare i design pattern richiesti per visualizzare i dati e i widget per rappresentarli, inserirli o interagire con essi in altro modo.

- Cercare di comporre l'interfaccia con il minor numero possibile di widget, sii coerente.

- Chiamando Element ogni widget o design pattern

individuato, allora...

Elementi HTML

- Element è disponibile in HTML?
- Bene, usalo ... Non provare a emularlo, a meno che tu non abbia una ragione molto, molto buona per farlo (probabilmente non ce l'hai). Sia e_html tale elemento HTML.
- L'elemento e_html richiede l'utilizzo di attributi particolari per renderlo accessibile? Dovresti essere in grado di rispondere a questa domanda con una buona comprensione dei principi e delle linee guida WCAG 2.1, ma puoi anche scoprirla guardando "Techniques for WCAG 2.1".
- Regola empirica: se si tratta di un'immagine o di un elemento interattivo (ad es. un campo di un modulo), è così. Riempire tali attributi con valori significativi.

E_html è un'immagine?

- Se e_html è un'immagine statica (ovvero l'interazione con essa non avvia alcuna azione che altera lo stato dell'applicazione), chiediti "e_html è un'immagine decorativa?"
- In tal caso, assicurati che abbia un attributo alt vuoto (alt = "") per trasmettere tale informazione alle tecnologie assistive o visualizzala tramite CSS.
- Altrimenti, assicurati che abbia una descrizione significativa usando l'attributo "alt". Test: disattiva la visualizzazione delle immagini nel browser. Riesci a dare un senso a ciò che viene mostrato in quelle immagini semplicemente leggendo la loro descrizione?

Widget e schemi ARIA

- Element è un widget o design pattern che richiede l'utilizzo della specifica WAI-ARIA per essere reso accessibile? È molto probabile che sia documentato in "Esempi di modelli di progettazione ARIA", parte delle pratiche di authoring WAI-ARIA. Assicurati che si comporti esattamente come documentato nella guida. Sentiti libero di essere ispirato dal codice di quelle pagine per implementarli se necessario (dopo tutto è stato scritto proprio per questo scopo ;)

Gestione del focus

- Più dinamica e sofisticata è la tua applicazione, maggiore è l'importanza della gestione del focus! Ogni volta che si verificano cambiamenti dello stato dell'interfaccia, chiediti:
 - Dov'è il focus?
 - Dove dovrebbe essere?
- La gestione del focus errato o mancante causa il disorientamento degli utenti di tecnologie assistive (una finestra di dialogo modale viene chiusa, ma il focus non viene riportato sull'elemento che ne ha causato l'apertura) e lo stato dell'interfaccia utente cambia inosservato (ad esempio viene visualizzata una finestra di dialogo modale, ma il focus

non viene posizionato sul suo primo figlio focalizzabile).

- Spostare il focus su un elemento potrebbe far sì che gli utenti di tecnologie assistive ignorino gli elementi che lo precedono, quindi scegli saggiamente quando farlo.
- Regola empirica: nessun autofocus di una pagina / vista / schermo, a meno che non sia richiesto dal design pattern che la implementa.

Informazioni sui framework esterni

- Realisticamente, è probabile che Element sia un componente fornito da un framework esterno o un elemento HTML migliorato da questo. Occorre sempre assicurarsi che sia accessibile o risolvere i suoi problemi di accessibilità nel tuo codice. Scegli un framework noto per essere un buon punto di partenza in questo senso:

Bootstrap (ultima versione 4.x), angular-material, elements-ui per citarne alcuni. In ogni caso, aspettati di fare un po' di lavoro su questo fronte. Ricorda, sei sempre responsabile per qualsiasi cosa tu consegni.

A proposito di styling

- Sentiti libero di “stilizzare” i tuoi elementi come desideri, ma tieni a mente i principi di accessibilità.
- Presta particolare attenzione al contrasto cromatico, alle dimensioni dei caratteri e rendi il tuo layout il più responsive possibile per rispondere alle modifiche alle dimensioni dei caratteri, allo zoom, ecc.
- Scegli i caratteri che rendono il contenuto più leggibile (tenendo anche presente il contesto).

Test automatizzati

- Gli strumenti di test di accessibilità automatizzati possono aiutarti a identificare i problemi di accessibilità, usali! Fai attenzione a scegliere quelli affidabili.
- Ax per Google Chrome, Ax per Mozilla Firefox da Deque, Systems e / o il Strumento di valutazione Wave per Chrome e Firefox da WebAIM sono raccomandati.
- Assicurati di eseguire tali strumenti per ogni variazione della tua interfaccia (ad es. quando un trigger di modulo si guasta o no, viene presentato o meno un modale, un menu viene espanso o compresso, ecc.).

Test manuale I

- Il test di accessibilità manuale è essenziale.
- È possibile interagire con ogni parte dell’interfaccia esclusivamente utilizzando la tastiera? Verificalo!
- Prova a utilizzare uno screen reader per interagire con la tua applicazione: ha senso il “navigation flow”? La tua interfaccia è utilizzabile, comprensibile e percepibile? Lo stato cambia anche manualmente o automaticamente?
- L’uso di uno screen reader nativo è piuttosto complesso, ma il Chrome Vox è consigliata l’estensione per Google Chrome; ha un ottimo

tutorial introduttivo che spiega come usarlo, segui!

Test manuale II

- Presta attenzione a identificare le informazioni trasmesse solo dai colori, anche se si spera a questo punto non dovresti averne. Se trovate, ripetere questo processo su quel caso particolare per trovare una rappresentazione accessibile.
- Prova ad ingrandire le dimensioni dei caratteri (almeno da 2x a 4x). La tua interfaccia si adatta bene a questo cambiamento?

Massime priorità

- Supporto tastiera
- Etichette per moduli
- Gestione del focus
- Alternative testuali
- Contrasto cromatico e dimensioni dei caratteri

Non dimenticare i contenuti

- Come abbiamo detto, la progettazione e lo sviluppo sono solo due dei tre principali componenti coinvolti nel rendere accessibile un sistema.
- Ricorda, una delle storie del tuo progetto deve essere accessibile, cioè giocabile anche da persone non vedenti. Ricorda che le persone con disabilità possono usare la tecnologia, ma con adattamenti (tecnologie assistive).
- Non richiedere azioni che una persona disabile non può eseguire (ad esempio, cercare qualcosa che si trova in una posizione elevata per una persona con sedia a rotelle, distinguere tra i colori per i non vedenti, ecc.).
- Ricorda i principi WCAG 2.1

Esempio: Un campo di un form I

```
<span>CVV: </span>
<input type = "text">
<p>Inserisci il codice di tre o quattro cifre della tua
carta di credito, generalmente situato sul retro. </p>
```

Questo controllo è accessibile?

Un campo di un form II

- NOO! Non esiste alcuna relazione tra il campo del modulo e la sua etichetta, così come tra il campo e la sua descrizione. Ecco una versione equivalente e accessibile di esso.

```
<label for="cvv"> CVV: </label>
<input type="text" id="cvv" aria-describedby =
"cvvdescr">
<p id="cvvdescr"> Inserisci il codice di tre o quattro cifre
della tua carta di credito, generalmente situato sul retro.
</p>
```

Domande?

www.unibo.it

Vincenzo Rubano
Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna
vincenzo.rubano@unibo.it