Tecnologie Web

Email: fabio.vitali@unibo.it
Pagine web: http://vitali.web.cs.unibo.it/

Introduzione
Un *ipertesto* è un insieme di documenti messi in relazione tra loro per mezzo di parole chiave. Può essere visto come una rete; i documenti ne costituiscono i nodi.
Lo *Standard Generalized Markup Language* (SGML), è un metalinguaggio avente lo scopo di definire linguaggi da utilizzare per la stesura di testi destinati ad essere trasmessi ed archiviati con strumenti informatici, ossia per la stesura di documenti in forma leggibile da computer.
Il *World Wide Web Consortium* (W3C), è un'organizzazione non governativa internazionale che ha come scopo quello di sviluppare tutte le potenzialità del World Wide Web. La principale attività svolta dal W3C consiste nello stabilire standard tecnici per il World Wide Web inerenti sia i linguaggi di markup che i protocolli di comunicazione.

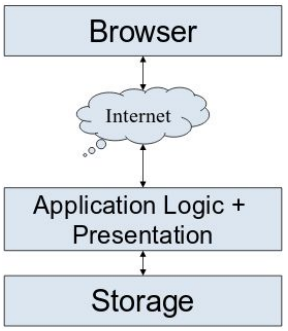
World Wide Web
Sistema Client Server per la visualizzazione di documenti ipertestuali con interfaccia generica. Composto da Client e Server. Il modello HTTP definisce chi può essere chiamato Client e chi Server. Il server è un'interfaccia tra file system e le richieste, serve il file a chi lo chiede, in generale, è un ambiente di servizio file che trasmette informazioni al client.
Il protocollo URI è uno standard per identificare in maniera generale risorse di rete e poter specificare all'interno di documenti ipertestuali. Il protocollo HTTP è uno standard di comunicazione stateless e client-server per l'accesso a risorse ipertestuali via rete.HTML è utilizzato per la realizzazione di documenti ipertestuali.
Le evoluzioni del WWW comprendono: l'inclusione di oggetti (es. immagini in-line), il client scripting (es. Javascript), l'introduzione di stili tipografici (es. CSS), la gestione delle transazioni (es. cookies, AJAX), la dinamicità dei siti web (es. PHP, DBs), la struttura dei documenti (es. XML), i framework di sviluppo (es. Django, Rails, GWT).

LAMP (Linux, Apache, MySQL, PHP)
Stack di linguaggi e protocolli per la realizzazione di siti web dinamici e interattivi, con memorizzazione delle informazioni su un database relazione, separazione di memorizzazione, logica e distribuzione.

REST (Representational State Transfer)
Modello di interazione tra client e server proposto da HTTP.
E' un insieme di regole che regola come strutturare un progetto (es. MVC)
Una corretta applicazione di REST garantisce interoperabilità, scalabilità, e uso delle catatteristiche HTTP.

AJAX (Asynchronous Javascript And XML)
Meccanismo per generare applicazioni web client-side e server-side fortemente interattive e in grado di minimizzare il traffico di rete.

MEAN (MongoDB, ExpressJS, AngularJS, NodeJS)
Un nuovo stack di linguaggi e tecnologie.



Siti statici: semplici, ogni pagina è a se stante.
Siti dinamici a 3 livelli: Browser (JS), Application Logic + Presentation (PHP), Storage (MySQL). C'è permanenza dei dati.
Siti dinamici a 4 livelli: Browser (JS), Application Logic, Presentation (PHP), Storage (MySQL)
Siti dinamici modello rich client: una parte più o meno complessa della rappresentazione complessiva viene trasferita al client, il client diventa quindi un ambiente complesso che inizia a sostituire il server (Ajax, web 2.0).

Ajax
Tramite un'apposita chiamata di libreria, XMLHttpRequest, si permette ad applicazioni client di effettuare connessioni al server, in modo indipendente da ciò che viene visualizzato nella pagina (in modo asincrono). Il file HTML testuale viene parsato dal browser che ne crea un albero in base ai tag, questo albero è il DOM. Un'applicazione Javascript andrà quindi a modificare il DOM in base ai vari eventi innescati dall'utente.

Caratteri
Elementi fondamentali della scrittura, per trovare un adeguato modo di rappresentarli.
I protocolli comunicano in US-ASCII e questo non è un problema, i problemi nascono per i contenuti dei messaggi scambiati dai protocolli di comunicazione. Per rappresentare un testo, è necessario fornire una rappresentazione digitale (cioè numerica in quanto va ridotto tutto a bit) degli elementi fondanti della scrittura.

Il glifo è la forma di una lettera, può assumere suoni diversi in lingue diverse (es. P in italiano è "p" in cirillico e greco si legge "r").

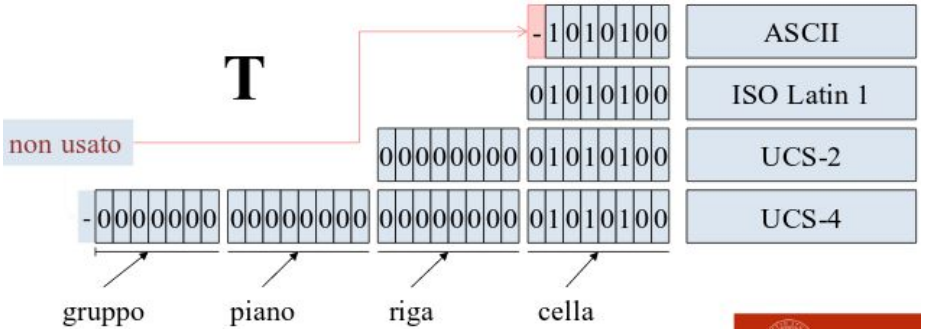
Ordine: i valori numerici seguono l'ordine alfabetico culturalmente riconosciuto.
Contiguità: ogni valore numerico compreso tra il più basso e il più alto è associato ad un carattere.
Raggruppamento: l'appartenenza ad un gruppo logico è facilmente riconoscibile da considerazioni numeriche.
Shift: un codice riservato che cambia mappa da adesso in poi. Lo stesso shift o un secondo carattere di shift, può poi far tornare alla mappa originaria
Codici liberi: codici non associati a nessun carattere. La loro presenza in un flusso di dati indica probabilmente un errore di trasmissione.
Codici di controllo: codici associati alla trasmissione e non al messaggio.

ASCII (American Standard Code for Information Interchange): Secondo lo standard ANSI (X3.4 - 1968) che definisce valori per 128 caratteri, ovvero 7 bit su 8. Nello standard originale il primo bit non è significativo ed è pensato come bit di parità.

EBCDIC (Extended Binary Characters for Digital Interchange Code): codifica proprietaria (IBM, 1965) a 8 bit, viene usata nei suoi mainframe. Contemporaneo dell'ASCII.
ISO 646-1991: codifica ISO per permettere l'uso di caratteri nazionali europei in un contesto sostanzialmente ASCII. ISO 646 lascia 12 codici liberi per le versioni nazionali dei vari linguaggi europei. Ogni tabella nazionale la usa per i propri fini.
Code page ASCII: quando iniziò ad essere disponibile dell'hardware affidabile vennero proposte delle estensioni di ASCII per usare i rimanenti 128 caratteri (128-255). IBM (e, separatamente, Microsoft) proposero un meccanismo di estensioni multiple ed indipendenti di ASCII per le necessità di script, alfabeti ed usi diversi, chiamate code page. In generale i caratteri da 0 a 127 sono quelli di ASCII.
Codifiche CJK: il cinese, il Giapponese e il Koreano (CJK) condividono molti caratteri e per cui tradizionalmente sono stati codificati insieme. Poiché i caratteri sono molti di più di 256, non è possibile usare una codifica a 8 bit, ma sono stati usate codifiche a 16 bit (56k combinazioni) e anche più larghe (Unicode al momento codifica 70.000 caratteri di CJK).
ISO 8859/1 (ISO Latin 1): l'unica estensione standard e comprende un certo numero di caratteri degli alfabeti europei come accenti, ecc.

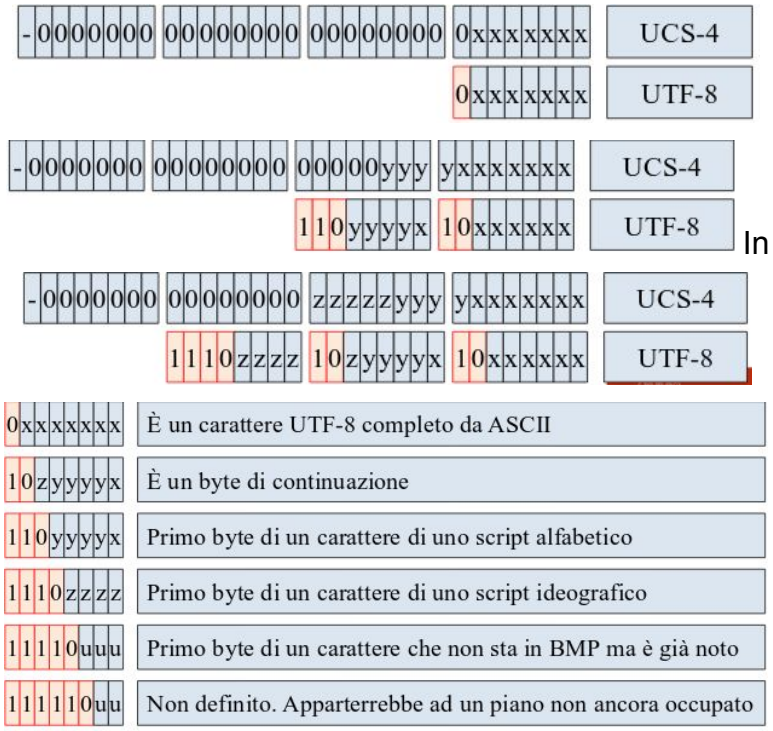
È quindi necessaria una codifica universale per tutti i tipi di caratteri. Il compito di creare uno standard unico è stato affrontato indipendentemente da due commissioni di standard, Unicode e ISO/IEC 10646. Le due commissioni, una industriale, l'altra espressione governativa, hanno lavorato indipendentemente per le prime versioni, salvo poi convergere.

ISO 10646 è composto di due schemi di codifica:
- UCS-2 è uno schema a due byte. È un'estensione di ISO Latin 1.
- UCS-4 è uno schema a 31 bit in 4 byte, estensione di UCS-2. È diviso in gruppi, piani, righe e celle.



In UCS-4 esistono dunque 32768 piani di 65536 caratteri ciascuno. Il primo piano, o piano 0, è noto come BMP (Basic Multilingual Plane) ed è ovviamente equivalente a UCS-2.

In questo modo però per mandare semplici messaggi in inglese si invieranno una marea di byte che non saranno mai utilizzati, nasce quindi UTF (Unicode Transformation Format o UCS Transformation Format) che è un sistema a lunghezza variabile che permette di accedere a tutti i caratteri di UCS in maniera semplificata e più efficiente.
UTF-8 permette di accedere a tutti i caratteri definiti di UCS-4, ma utilizza un numero compreso tra 1 e 4 byte per farlo. Se il primo bit è 0, si tratta di un carattere ASCII di un byte; se i primi due bit sono 11, si tratta di un carattere appartenente ad un alfabeto non ideografico; se i primi tre bit sono 111, si tratta di un carattere appartenente ad un alfabeto ideografico.



In

Unicode specifica un codice, FFFE, come segnalatore di ordinamento del flusso. FEFF è il carattere Zero-Width No-Break Space (ZWNBSP), un carattere che può essere usato in qualunque contesto di whitespace (cioè ovunque tranne in mezzo alle parole) senza modificare il significato dei testi. La sua forma corrispondente in little-endian, FFFE, è un carattere proibito in Unicode. Unicode suggerisce allora di utilizzare un carattere ZWNBSP all'inizio di ogni flusso UTF-16 e UCS-2. Se il processore riceve FEFF deduce che il sistema sorgente è big-endian, altrimenti che è little-endian, e decide di riconvertire il flusso su questa base.

Content encoding

Escaping: il carattere proibito viene preceduto o sostituito da una sequenza di caratteri speciali. Es. “this is a \”string\”.”

Encoding: il carattere proibito viene rappresentato numericamente con il suo codice naturale secondo una sintassi speciale. Es. felicità

I problemi nascono dal protocollo SMTP, che impone certi limiti, come la lunghezza massima del messaggio è di 1 Mb, i caratteri accettati sono solo ASCII a 7 bit, ogni messaggio deve contenere una sequenza CRLF ogni 1000 caratteri o meno. Questi limiti impediscono la trasmissione di documenti binari. MIME (Multipurpose Internet Mail Extensions) vuole risolvere questi problemi. Il messaggio non compatibile con SMTP viene trasformato in uno o più messaggi SMTP da un preprocessore al server SMTP.

I limiti SMTP su MIME sono: la codifica caratteri, le sequenze CRLF e la lunghezza dei messaggi. I servizi di MIME sono: la dichiarazione di tipo e i messaggi multi-tipo.

Special: spazio web del dipartimento

http://www.informatica.unibo.it/it/servizi-e-strutture

http://www.informatica.unibo.it/it/servizi-informatici/accesso-da-remoto

tecnici@cs.unibo.it

FTP:
Host: annina.cs.unibo.it
Username: *usernameunibo*
Password: “password”
Port: 22
SSH
ssh *usernameunibo*@annina.cs.unibo.it
Online
/home/web/*usernameunibo*/html/
http://*usernameunibo*.tw.cs.unibo.it/
Interprete node
/usr/local/node/bin/node

HTTP (HyperText Transfer Protocol)
È un protocollo client-server generico (ovvero indipendente dal formato dei dati con cui vengono trasmesse le risorse) e stateless (il server non è tenuto a mantenere informazioni che persistano tra una connessione di un client, e la successiva).
> *User agent*: è un particolare client che inizia una richiesta HTTP (un browser, un bot, …)
> *Origin server*: il server che effettivamente possiede fisicamente la risorsa richiesta.
> *Proxy*: applicazione intermedia che agisce sia da client che da server.
> *Gateway*: applicazione che agisce da intermediario per qualche altro server.
> *Tunnel*: programma intermediario che agisce da trasmettitore passivo di una richiesta HTTP.

HTTP/1.1
Le *connessioni persistenti* permettono meno connessioni TCP, permettono di ridurre l’attesa della visualizzazione e di gestire in modo migliore gli errori. Il *pipelining* è la trasmissione di più richieste senza attendere l’arrivo della risposta alle richieste precedenti.

HTTP/2
Con il *multiplexing* nella stessa connessioni è possibile avere richieste e risposte multiple “ricostruite” nell’host di destinazione; molte più operazioni in parallelo,
quindi molta meno latenza. Vi è anche una compressione degli headers.

I metodi della richiesta
Un metodo è sicuro se non genera cambiamenti allo stato interno del server.
Un metodo è idempotente se l’effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta.

GET: recupera (una copia di) una rappresentazione di una risorsa. [sicuro e idempotente]
HEAD: recupera (una copia di) una rappresentazione delle cose che un server sa su una risorsa. [sicuro e idempotente]
POST: associa delle informazioni ad una risorsa. [non sicuro e non idempotente]
PUT: inserisce una nuova risorsa. [sicuro ma non idempotente]
DELETE: rimuove una risorsa e ogni informazione presso di essa. [non sicuro ma idempotente]

OPTIONS: fornisce l'elenco delle opzioni attive del server web (ma serve anche per testare la raggiungibilità del sito).

Gli *headers* specificano quattro tipi di informazione che caratterizzano: la trasmissione, l'entità trasmessa, la richiesta effettuata e la risposta generata.

Status Codes
1xx: **Informational**. Una risposta temporanea alla richiesta, durante il suo svolgimento.
2xx: **Successful**. Il server ha ricevuto, capito e accettato la richiesta.
3xx: **Redirection**. Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta.
4xx: **Client error**. La richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata).
5xx: **Server error**. La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno (suo o di applicazioni CGI).

Cookies
Il termine cookie in informatica indica un blocco di dati opaco (i.e.: non interpretabile) lasciato in consegna ad un richiedente per poter ristabilire in seguito il suo diritto alla risorsa richiesta.

Cross-site vulnerability
Supponiamo che in una pagina appartenente ad un dominio protetto (es. in una sessione autenticata) si riesca ad intrufolare un pezzo di codice malizioso, che raccoglie e trasmette informazioni ad un repository accessibile ai malintenzionati. Poiché è difficile pensare che il codice malizioso venga dallo stesso dominio della pagina protetta, si parla di vulnerabilità tra domini o cross-domain. Una politica (molto conservatrice) dei browser è quella di rifiutare qualunque connessione Javascript ad un dominio diverso da quello della pagina ospitante (dominio diverso = schema o dominio o porta diversa). Solo gli script che originano nello stesso dominio della pagina HTML verranno eseguiti. Due eccezioni/soluzioni: JSON-P (lo vedremo nella lezione Javascript); CORS.

Cross-Origin Resource Sharing (CORS)
Una tecnica HTTP introdotta dal W3C (W3C Rec 29/1/2013) e priva dei limiti e dei difetti di JSON-P. Si attiva solo per le connessioni Ajax (non si usa per connessioni HTTP normali) e prevede l'uso di due nuovi header:
> nella richiesta: Origin, per specificare il dominio su cui si trova il contenuto
> nella risposta: Access-Control-Allow-Origin per indicare gli altri domini da cui è permesso caricare contenuti.

REST (REpresentional State Transfer)
È un modello architetturale che sta dietro al World Wide Web e in generale dietro alle applicazioni web "ben fatte" secondo i progettisti di HTTP. Applicazioni non REST si basano sulla generazione di un'API specifica alle funzioni messe a disposizione dell'applicazione, e alla creazione di un'interfaccia indipendente dal protocollo di trasporto e ad essa completamente nascosta. Viceversa, 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.

Web Service
Scopo dei Web Service è realizzare un interfaccia HTTP che permetta ad applicazioni remote di utilizzare i servizi della mia applicazione.

URI (Uniform Resource Identifier)
Il web è tutto ciò che è caratterizzato e descritto da un URI. Quest'ultimo identifica una risorsa, può essere un esempio di URI anche un codice a barre su un prodotto al supermercato, così come un documento su un server. Questi identificatori sono la vera e propria invenzione di Tim Berners Lee, ovvero una sintassi sotto forma di stringa per identificare file, programmi, applicazioni, siti web, concetti,…

- Concetto di *locazione* (Uniform Resource Locator - URL): insieme di istruzioni immediatamente eseguibili da un processo automatico per accedere alla risorsa di interesse. È possibile che con il tempo non sia però più valido (es. error 404).
- Concetto di *nome* (Uniform Resource Names - URN): un meccanismo permanente e non ripudiabile per associare un id alla risorsa stessa. Ci si aspetta che questo sia permanente, sopravvivendo ad eventuali modifiche tecniche architetturali alle strutture di accesso (dato un nome, restituisce un locatore utilizzabile per accedere alla risorsa).
Es. il numero di telefono è un URI per raggiungere l'utente che lo possiede.

La struttura URI:
`schema: [//authority] path [? query] [# fragment]`

schema: stringa registrata presso IANA usata come prefisso (negli URL è il protocollo).
authority: organizzazione gerarchica dello spazio dei nomi a cui sono delegati i nomi, questa è suddivisa in: `[userinfo @] host [: port]`
> **userinfo**: è utilizzato se lo schema prevede identificazione personale (però passa i dati in chiaro).
> **host**: o un nome di dominio, o un indirizzo IP.
> **port**: porta di accesso, omessa se la porta è nota (es. 80 per http).
path: parte identificativa della risorsa all'interno dello spazio dei nomi indicato dallo schema e (se esiste) dall'authority.
query: fornisce un meccanismo di passaggio di parametri ad una risorsa (verosimilmente un processo server-side non statico) nella forma: `nome1=valore1&nome2=piu+parole`.
fragment: identifica uno specifico frammento della risorsa su cui ci si vuole focalizzare.

Negli URI a volte è necessario effettuare un'escape dei caratteri nella forma %XXX (con XXX forma esadecimale del carattere).

URI reference (URI ref)
Stringhe di caratteri con caratteristiche simili agli URI utilizzabili in specifici contesti per riferirsi alle risorse dinamicamente. Ad esempio se all'interno di http://www.sito.com/dir1/dir2/pluto.html

si chiama pippo.html, si fa riferimento in realtà a http://www.sito.com/dir1/dir2/pippo.html, esiste il concetto quindi di URI base.
> *risoluzione*: operazione che applicata ad un URI ref restituisce l'URI (assoluto).
> *deferenziazione*: l'operazione di fornitura della risorsa (viene effettuata dal browser).

Schema FILE
Da accesso ai file di un file system locale (cioè del computer su cui gira il browser), è nella forma: `file://host/path [#fragment]`

Schema DATA
Schema non gerarchico, in cui URI e risorsa coincidono, un modo per inserire inline una risorsa multimediale, nella forma: `data:[<media type>][;base64],<data>`
Es. `data:text/plain;charset=UTF-8;some%20text%20for%20you`

Schema FTP
Nella forma: `ftp://[user[:password]@]host[:port]/path`

Errori con gli URI
Fare assunzioni sulla natura della risorsa in base all'URI, ad esempio se l'URI termina con .html assumere che sia un file HTML, sarebbe un errore. URI è un identificatore non un descrittore.

URL permanenti
Nomi che utilizzano la stessi sintassi dei locatori, ma garantiscono persistenza nel tempo della risorsa puntata.È possibile assegnare ad un URL qualunque, anche in aggiunta un URL permanente, in modo che anche se cambio dominio, o nome del server, di raggiungere comunque la risorsa. Questo tramite un URL che lavora in modo indipendente e separato.

URL rewriting
Quando è possibile avere un'organizzazione interna molto complicata, ma si espone all'esterno un URL più semplice e immediato, es. http://example.com/wiki/index.php?title=Argomento diventa http://example.com/Argomento.

URI shortener
Servizi che abbreviano URI, e che funzionano tramite redirezione, es. bit.ly, goo.gl, …

Gli URI tuttavia possono contenere solo caratteri ASCII, quindi non alfabeti diversi (es. cinese, giapponese, ecc). Nel 2005 nasce l'**IRI** (Internationalized Resource Identifier) che permette di inserire all'interno di un URI tutti i caratteri di UCS-4. Uno dei rischi di IDNA è utilizzare caratteri di altri alfabeti al posto di quelli latini, che risulterebbero diversi a livello tecnico, ma simili all'occhio, questi potrebbero essere usati per spoofing e phishing. Per questo non sono più ammessi mix di caratteri di script diversi.

CURIE (Compact URI)
Sintassi per abbreviare URI lunghi, utilizzando un prefisso invece che un URI vero e proprio.

Ad esempio:

```
<html xmlns:dom="http://www.dominio.com/">
     <p>Si veda <a href="[dom:pagina1]">
          la prima pagina</a></p>
</html>
```

È utilizzabile [dom:pagina1] al posto di http://www.dominio.com/pagina1

Linked Data

Modello concettuale per la caratterizzazione di tutte le risorse di dati che possono essere messi a disposizione ed incrociati da applicazioni ed umani.

HTML

HTML non è più uno standard W3C, viene considerato dai produttori di browser un insieme complesso di tecnologie, che comprende anche un linguaggio di markup. La differenza tra HTML 5.1 a HTML Living Standard è la modalità in cui l'evoluzione del linguaggio avviene, il primo è una raccomandazione del W3C, la seconda del WHATWG, quest'ultima ha aggiornamenti molto più frequenti e indipendenti.

SGML

Standard ISO (modello più alto degli standard, nella gerarchia degli standard) di linguaggio di markup. SGML è un metalinguaggio di markup, ovvero un linguaggio con cui definire linguaggi di markup. Solo con la versione 4.0 HTML diventa un linguaggio di markup basato su SGML, prima ne violava le regole.

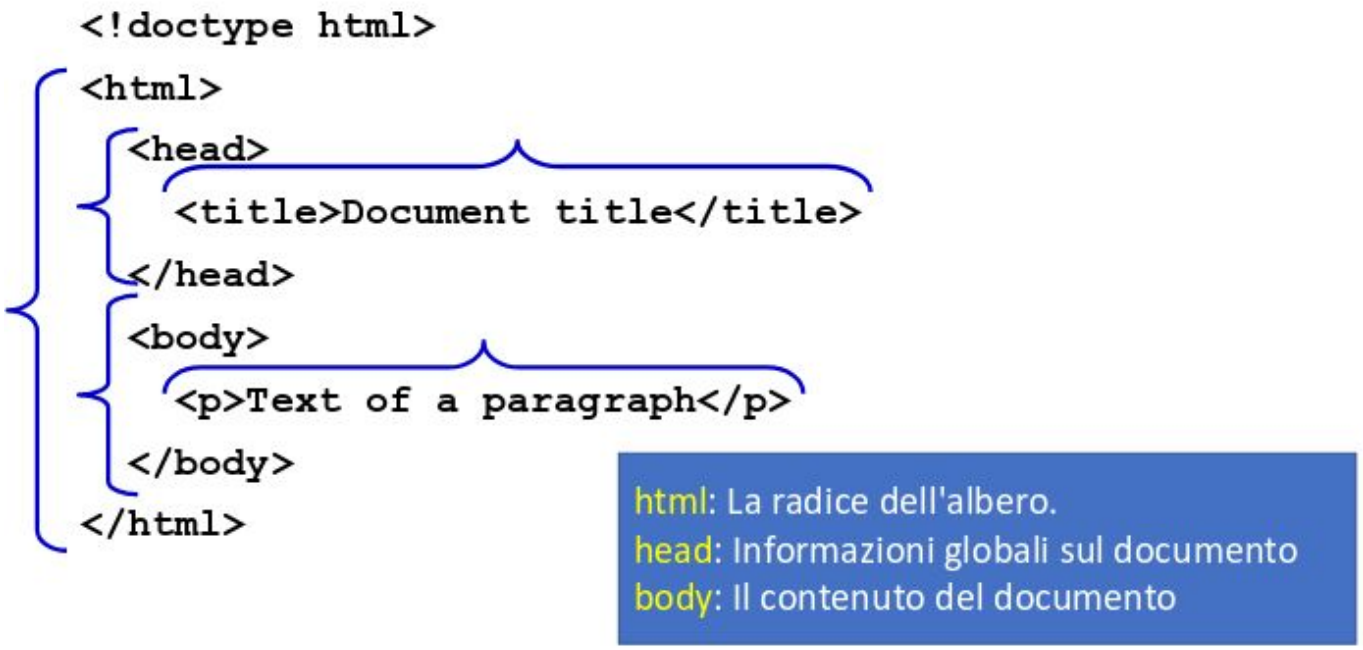
Un documento SGML è un documento leggibile (di testo), generico (può riguardare qualsiasi ambito), strutturale (è definito da quali componenti è creato e come si relazionano tra loro) e gerarchico (queste strutture hanno una struttura gerarchica, annidata).

La versione finale di HTML nel '99 è HTML 4, e per un po' W3C non fornisce aggiornamenti. Tuttavia i browser hanno un approccio flessibile, fornendo una visualizzazione anche a documenti HTML che non seguono lo standard ultimo. Si è voluto quindi prevedere due modelli di HTML e CSS: *quirks mode* e *strict mode*, il primo è molto permissivo e accetta errori noti nei documenti, il secondo è compatibile solo con le specifiche ufficiali.

A W3C questi approcci non piacciono, crea quindi la Recommendation XHTML 1.0 nel 2000, ovvero una riformulazione di HTML 4 come applicazione XML 1.0. Un vocabolario di markup di html sulla base delle regole di xml e non più di html con qualche lieve differenza, contenente una sintassi molto più rigorosa e rigida.

Vi sono varie incongruenze tra i produttori di browser, molto permissivi, e gli standard prodotti dal W3C, molto stringenti. Nasce WG che produce standard frequenti, però estremamente flessibili.

HTML 4



Una struttura gerarchica composta da elementi e tag (in xml considerati diversi, in html intesi come la stessa cosa). Far convivere nello stesso contenitore, struttura e contenuto, il fatto che alcuni elementi contengano al loro interno sia altri elementi che testo.

Elemento di tipo inline: non rompe l'organizzazione naturale in paragrafi.
Elemento di tipo blocco: organizzazione verticale del testo.
Elemento generico: un elemento vuoto, privo di caratteristiche semantiche o presentazionali, si definirà un elemento "class" all'interno per dargli significato (es. div e span).

Tabelle
Le tabelle sono un'organizzazione bidimensionale di dati omogenei per tipo. Tre aree principali, "thead" "tbody" e "tfoot". Tuttavia vi sono altre tabelle, che sono utilizzate per fare tabelle di layout.

Form
Un form è costituito da n campi di una pagina, modificabili dall'utente, e un elemento submit che permette di creare una connessione HTTP con il server, a cui far arrivare i dati.
La codifica è *application/x-www-form-urlencoded*, un'estensione della codifica degli URI prevista nello standard della sintassi dell'URI.

Document Object Model (DOM)
Insieme di classi, metodi e membri di classi, disponibili dei linguaggi di programmazione (es javascript) attraverso i quali è possibile modificare il documento HTML.
Il Document Object Model è un interfaccia di programmazione (API) per documenti sia HTML sia XML. Definisce la struttura logica dei documenti ed il modo in cui si accede e si manipola un documento.

DOMDocument : il documento di cui si sta parlando
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.

Presentazione PROGETTI A.A. 2017/2018
È necessario che il server giri su una macchina del dipartimento (GNU/Linux).
NO Ruby, Java, Pearl, PHP, (NI) Python, SI Node.

Distinguere i video ufficiali da lyrics, cover, ecc.
Prendere informazioni da youtube (artista, album, ecc) e approfondirle su wikipedia.
Indicare alla discussione le parti facoltative fatte.
Il recommender *search* può restituire meno di 10 elementi (se il risultato comprende meno di 10 risultati). Salvare i dati di utilizzo dell'utente tramite la feature "webstorage" dei browsers.
Gestire in maniera corretta i pulsanti back e forward sui video.

Cascading Style Sheets
http://www.fabiovitali.it/TW/2018/slides/07-CSS.pdf

Preprocessori CSS
LESS e SASS sono i due processori più noti, ovvero dei linguaggi che vengono integrati in un'applicazione per estendere le funzionalità di CSS. Permettono ad esempio di creare variabili, utilizzare aritmetica base e mix-in ovvero regole parametrizzabili, simili a funzioni.

Framework CSS
Librerie già pronte con regole associate ad elementi e classi, pronte ad essere utilizzate. Il grande vantaggio è che l'utilizzo di un framework garantisce l'adattamento a browser e sistemi operativi diversi. Esempio più famoso Twitter Bootstrap, una specifica CSS più una specifica libreria JS che crea un mondo di regole CSS che possono essere attivate in modo semplice all'interno della propria pagina web. Bootstrap fornisce un modello di scatole basato su divisori espressi in 12esimi.

Esercitazione: esercizitw@gmail.com
Subject: **TW2018: esercizio HTML n [x] di [nome cognome] ([matricola])**
L'ultimo ha subject: **Completato TW2018: esercizio HTML n [x] di [nome cognome] ([matricola])**
Consegna massima: lunedì 23 aprile entro le 13:00

Javascript

Vi sono due oggetti principali predefiniti del browser: *window* che rappresenta la finestra principale e *navigator* che rappresenta il browser. Questi due oggetti sono separati, in modo che

una window non possa interrogare il navigator per sapere quali altre finestre sono aperte, ad esempio.

window: è l'oggetto top-level con le proprietà e i metodi della finestra principale. *Una variabile globale è per costruzione un membro dell'oggetto window* (sul browser infatti la variabile globale x e window.x sono la stessa cosa).
navigator: è l'oggetto con le proprietà del client come nome, numero di versione, plug-in installati, supporto per i cookie, etc.
location: l'URL del documento corrente. Modificando questa proprietà il client accede a un nuovo URL (redirect). Tale oggetto non è accessibile dal DOM, ma solo tramite *window.location*, sia in lettura che scrittura.
history: l'array degli URL acceduti durante la navigazione. È possibile creare applicazioni client-side dinamiche che 'navigano la cronologia'. I pulsanti back e forward sono gestiti qui, sta allo sviluppatore sovrascriverli e impedire che la navigazione torni alla pagina web precedente, invece che mostrare l'azione precedente.
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).

Il **DOM** è la rappresentazione per javascript del documento (come HTML, XML, …), fornendo meccanismi per navigare e modificare l'albero dei nodi. La sovraclasse di tutto p node, da cui scendono document, element (tutti gli elementi del documento, al cui interno possono esservi attribute e textnode), i.e. `document.getElementById('c35');`
`elem.setAttribute('class', 'prova1'); document.createElement('p');`
`document.createTextNode('Ciao Mamma.'); insertBefore(Olist, div);`

Oggi oggetto nel DOM ha un metodo per ogni evento a cui sa rispondere, solitamente iniziano con "*on*" e hanno il nome dell'evento dopo.

Bubbling degli eventi, meccanismo standard dei sistemi a finestra, a cui si può associare il meccanismo di reazione agli eventi ad un qualunque elemento nella gerarchia degli elementi che man mano si risalirà al parent di ogni elemento fino a che non si trova una reazione all'evento verificatosi.

InnerHTML è la rappresentazione in stringa del DOM dell'elemento che su cui viene chiamato.

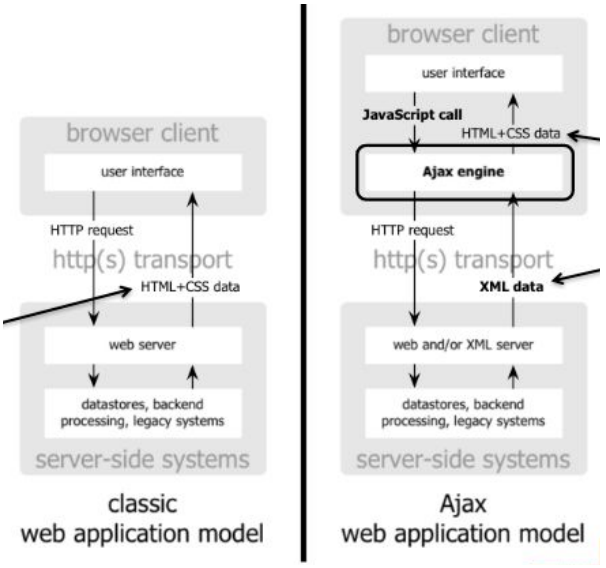
<u>Selettori nel DOM</u>
Base:

getElementById: solo ovviamente se l'elemento ha un id
getElementsByName: se l'elemento ha un attributo name
getElementsByTagName: tutti gli elementi con nome specificato

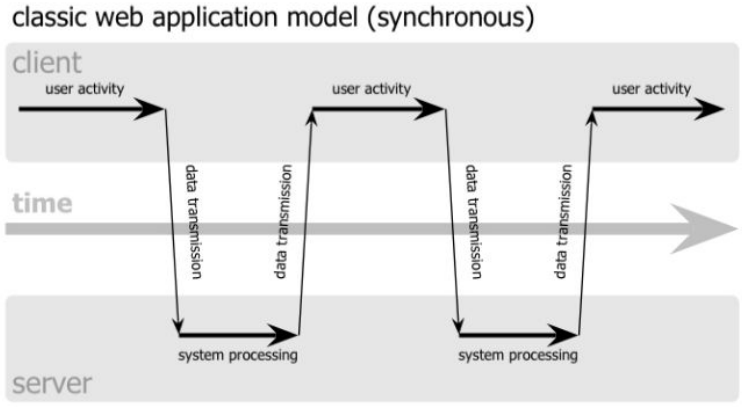
Dopo l’avvento di JQuery:

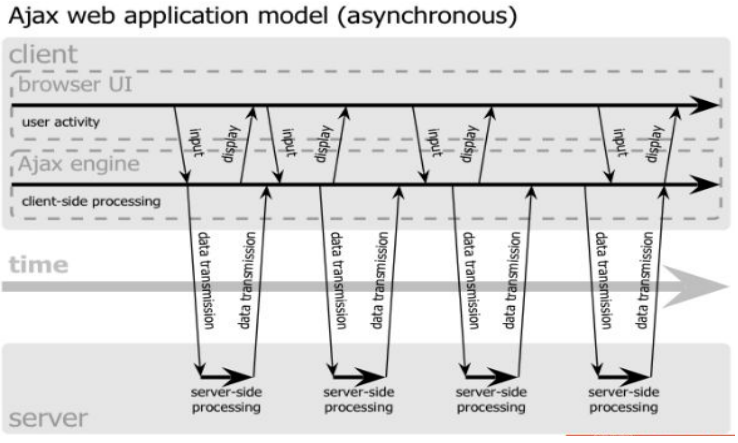
getElementsByClassName; cerca tutti gli elementi di classe specificata
querySelector: accetta un qualunque selettore CSS e restituisce il primo elemento trovato - del tutto equivalente a $()[0] in JQuery
querySelectorAll: accetta un qualunque selettore CSS e restituisce tutti gli elementi trovati - del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

AJAX (Asynchronous JavaScript And XML)
Stile di progettazione per applicazioni web, utilizzato per la creazione di applicazioni Web interattive. Permette l’aggiornamento asincrono di porzioni di pagine HTML. Viene utilizzato per incrementare: l’interattività, la velocità e l’usabilità.
Non è un linguaggio di programmazione o una tecnologia specifica, è un termine che indica l’utilizzo di una combinazione di tecnologie comunemente utilizzate sul Web: XHTML e CSS, il DOM modificato attraverso JavaScript per la manipolazione dinamica dei contenuti e dell’aspetto, XMLHttpRequest (XHR) per lo scambio di messaggi asincroni tra browser e web server e XML o JSON come meta-linguaggi dei dati scambiati.



Architettura





I pregi sono tra gli altri, la velocità, data dalla minore quantità di dati scambiati, infatti non è richiesto l'invio di intere pagine, ma solo dei dati da aggiornare; infatti una parte della computazione è spostata sul client.

Inizializzazione della richiesta

```
http_request.onreadystatechange = nameOfTheFunction;
http_request.open('GET','http://www.example.org/some.file', true);
```

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.

Invio della richiesta

```
http_request.send(null);
```

Il parametro della 'send' contiene il body della risorsa da inviare al server.

Gestione risposta

```
function nameOfTheFunction() {
      if(http_request.readyState == 4)
            // risposta ricevuta
      else
            // risposta non ricevuta ancora
}
```

I valori per 'readyState' possono essere:
0 = uninitialize
1 = loading
2 = loaded
3 = interactive
4 = complete

È necessario controllare anche 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)
}
```

È possibile anche leggere la risposta inviata dal server con:
`http_request.responseText`: che restituisce la risposta come testo semplice
`http_request.responseXML`: che restituisce la risposta come XMLDocument

Leggere e visualizzare i dati in modalità asincrona:

```
function getData(){

  // load the xml file
  myXMLHTTPRequest = new XMLHttpRequest();
  myXMLHTTPRequest.onreadystatechange = showData;
  myXMLHTTPRequest.open("GET", "example1.xml", true);
  myXMLHTTPRequest.send(null);
}

  function showData() {
    if (myXMLHTTPRequest.readyState == 4) {
      if (myXMLHTTPRequest.status == 200) {
          //legge la risposta
          xmlDoc = myXMLHTTPRequest.responseXML;
          //recupera il frammento
          fragment = xmlDoc.getElementById("content").item(0);
          // modifica il documento corrente
          document.getElementById("area1").appendChild(fragment);
        }
     }
  }
```

Javascript (avanzato)

Valori falsy e truthy
Un valore falsy è un valore che nel momento in cui viene castato a booleano è restituito false, equivalente per truthy. Esempi del primo sono: false, 0, null, undefined, “”, NaN, esempi del secondo sono “Mickey”, 3.14, Infinity, {}, [], “0”, “undefined”, “null”. In JS il casting a boolean è effettuato in modo automatico ovunque richiesto. Una tecnica interessante è utilizzare controlli binari nel seguente modo `var misura = (var || ‘12’) + “px”;` In questo modo se param è truthy viene restituito quello, altrimenti viene restituito ‘12’, grazie alla lazy evaluation.

Le funzioni in JS sono oggetti di primo ordine, ovvero sono oggetti di tipo Function, ed è possibile assegnare alle variabili delle funzioni, passare funzioni come parametri, restituire una funzione come risultato di un’altra funzione e così via, sono normali oggetti. Il seguente operatore “()”, operatore di invocazione, è utilizzato per eseguire oggetti di tipo Function, se appartengono a questa categoria.

Un function statement appartiene al linguaggio di programmazione, un function expression appartiene allo stato di runtime del programma. È possibile inoltre assegnare una funzione come metodo di un oggetto. La creazione di una funzione avviene tramite: `var a.meth = function(a, b) { ... }`, ovvero tramite la parola chiave function e indicando il numero di parametri in input. Verrà invocata con `a.meth(x, y);`

Javascript ha oggetto, ma non sono basati sul concetto di classe, ma quello di prototipo. Dato che le funzioni sono oggetti di primo livello, ogni oggetto può contenere al suo interno delle funzioni, senza ricorrere alla classe. È possibile istanziare oggetti semplicemente dichiarandone il contenuto, oppure tramite un costruttore. Non è necessario che le funzioni sia collegati ad una certa classe, ma è possibile ad ogni oggetto assegnare una certa funzione diversa in base alle

necessità. Per far sì invece che una funzione sia associata ad una classe, ovvero reperibile una volta istanziato un oggetto, si utilizzano i **Prototipi**, che fungono da “template” per gli oggetti.

Un costruttore è banalmente una funzione che resituisce un oggetto, Si usa new per utilizzarlo come costruttore dell’oggetto, i.e.

```
function Persona(nome, cognome) {
      this.nome = nome
      this.cognome = cognome
      this.saluta = function() { return “ciao” }
}
```

Un **prototipo** è un’istanza di un oggetto fittizio che comprende solo metodi che tutti gli oggetti creati dallo stesso costruttore condividono. Nel momento in cui viene cambiato il prototipo di un oggetto, viene automaticamente cambiato anche per tutti gli altri oggetti che hanno utilizzato quel prototipo, questo è possibile anche grazie al fatto che Javascript sia un linguaggio interpretato.

```
Persona.prototype.welcome = function() {
      alert(“Benvenuto, “ + this.name + “!”);
}
```

La seguente funzione:

```
String.prototype.endsWith = function(value) {
      return this.substr(-1*value.length) == value
}
```

Viene utilizzata per evitare *namespace pollution*, ovvero inquinare lo spazio dei nomi con un’insieme di funzioni inutilmente.

Questa tecnica ha senso utilizzarla se si sta creando una funzione che può essere utilizzata in più occasioni, e non è legata all’applicazione che si sta sviluppando, in questo modo si evitano possibili conflitti se si utilizzano librerie che ne fanno uso.

La funzione `bind(obj, args)` permette di associare parametri a funzioni anonime o chiamate indirettamente.

La superclasse è un modello puramente concettuale di classe che serve per dare ordine e omogeneità a tutte le sottoclassi, che sono tutte quelle istanziabili, i.e. non posso istanziare un felino, ma posso istanziare una tigre e indicare che è di tipo felino.

I linguaggi tradizionali hanno due scope, uno globale e uno più ristretto, in cui viene definita l avariabile. In JS è presente un terzo scope, dato che in questo linguaggio non è presente la protezione dei membri privati di un oggetto, ma sono tutti public, si utilizza questo terzo scopo per fornire una visibilità ridotta. Questo terzo scope, detto *closure*, 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.

```
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)
Uno dei pattern di programmazione JS più frequenti. 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.

```
var people = (function() {
  var persone = [] ;
  return {
    add:   function(p) { persone.push(p)},
    lista: function(){ return persone.join(', ') }
  }
})()
```

Un IIFE è un costruttore che viene chiamato solo una volta (definito e chiamato insieme) e restituisce un oggetto dotato di uno stato interno privato.

Programmazione asincrona
La caratteristica più peculiare e tipica di Javascript, evidente immediatamente, è la asincronicità come filosofia di design.
1. Una richiesta Ajax viene eseguita asincronicamente rispetto alla navigazione della pagina HTML
2. La gestione dei dati ricevuti via Ajax viene eseguita asincronicamente rispetto alla emissione della richiesta
3. La gestione degli eventi dell'utente viene eseguita asincronicamente rispetto alla specifica della funzione callback
4. setTimeout() posticipa di n millisecondi l'esecuzione di una funzione.

Non vi è concorrenza in quanto JS è single-threaded, una volta lanciato un comando che va a modificare il thread principale altri che sopraggiungono dovranno attendere il loro turno.

NOTA: prima viene sempre concluso il thread principale e poi ci si occupa dei vari thread asincroni, quindi in caso di una richiesta ajax in mezzo prima verrà letto il file fino alla fine, e poi gestiti i thread asincroni.

Soluzione 0: rinunciare all'asincronicità, settando il parametro async di ogni richiesta a false.
Soluzione 1: usare un linguaggio multi-threaded server-side, e fare un'unica richiesta al server, che si occupi di tutti i dettagli (approccio Ruby on Rails). Ho comunque un'attesa, e non ho nessun particolare vantaggio da Ajax.
Soluzione 2: Posso passare una funzione callback come argomento di chiamata a funzione, che viene eseguita alla conclusione del servizio (soluzione tradizionale Ajax). Tuttavia le callback: non possono restituire valori alla funzione chiamante, ma solo eseguire azioni coi dati ottenuti. Sono funzioni indipendenti, e vengono eseguite alla fine dell'esecuzione della funzione che le chiama. Non hanno accesso alle variabili locali della funzione chiamante, ma solo a variabili globali e closure.
Soluzione 3: una promessa è un oggetto che, si promette, tra un po' conterrà un valore. La promessa viene creata dalla funzione chiamante e mantenuta dalla funzione chiamata. Quando riceve asincronamente il risultato, chiama la funzione *resolve* della promessa. La funzione *then* viene eseguita in conclusione alla promessa precedente (con i dati della resolve come parametro). Un'altra libreria, Q, ha introdotto il concetto di generator e yield. Partiamo dal concetto di return, in generale, prima mette su uno stack il valore di ritorno, e successivamente conclude la funzione. Se si separasse il momento di fornitura sullo stack di un valore di ritorno, dalla terminazione della funzione, si potrebbero avere elementi che forniscono sullo stack valori di ritorno e altri che terminano la funzione. Tornando a generator/yield, Il generatore è una meta-funzione (una funzione che restituisce una funzione che può essere chiamata ripetutamente ed interrotta fino a che ne hai nuovamente bisogno). Ha una sintassi particolare con * dopo function. Il comando yield mette in attesa la assegnazione di valore fino a che non si chiude l'esecuzione della funzione chiamata. La funzione next() fa proseguire l'esecuzione della funzione fino al prossimo yield. In sostanza con yield si può creare un algoritmo che progressivamente si popola di oggetti risultato di un'esecuzione asincrona che si blocca fino al valore di ritorno, ma che non blocca l'esecuzione dell'algoritmo.

```
function *main(query) {
  var products = yield productDB.search(query) ;
  var opinions = yield opinionDB.search(products);
  var tweets   = yield twitter.search(opinions);
  var output   = prepareDOM(products, opinions, tweets);
  document.getElementById("display").appendChild(output);
}
var m = main();
m.next();
```

EcmaScript 2015

Funzioni freccia

Sintassi tradizionale
```
var power = function(a,b) {
  return Math.pow(a,b);
}
var c = power(5,3)
```

Nuova sintassi
```
var power = (a,b) => {
   return Math.pow(a,b);
}
var c = power(5,3)
```

Sintassi tradizionale
```
var arr = [1, 2, 3];
var squares = arr.map(function (x) { return x * x });
```

Nuova sintassi
```
var arr = [1, 2, 3];
var squares = arr.map(x => x * x);
```

Template literals

New way
```
var firstName = 'Jane';
var x = `Hello ${firstName}!
How are you
today?`;
```

```
x vale "Hello Jane!
How are you
today?"
```

Definizione classi

Sintassi tradizionale

```
var Shape = function(id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move=function(x,
y) {
    this.x = x;
       this.y = y;
};
```

Nuova sintassi

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

I Framework JavaScript

Librerie JS che aiutano nella creazione di applicazioni Ajax. Le tre macro-categorie di servizi messi a disposizione sono:
- *Astrazione*: gestiscono le differenze tra i browsers e forniscono un modello di programmazione (quasi) unico, che funziona su molti browsers.
- *Struttura dell'applicazione*: forniscono un modello di progetto omogeneo dell'app, un ambiente che guida e fornisce semilavorati utili per un gran numero di applicazioni web.
- *Libreria di widget*: forniscono una ricca collezione di elementi di interfaccia liberamente assemblabili per creare velocemente interfacce sofisticate e modulari.

Caratteristiche comuni
> Accesso al DOM
> Comunicazioni asincrone con il server (Ajax)
> Gestione eventi
> Librerie di widget

CDN - Content Delivery Network
Caratteristica delle rete di fornire servizi centralizzati di svariato tipo. Si decide tutti di utilizzare un unico accesso alle librerie condivise, in modo da permettere un miglior caching per l'utente (non è necessario ogni volta riscaricare vari KB di file jquery per ogni sito).

L'accesso a file protetti da https da fonti http genera un rallentamento dato dalla creazione della connessione https, mentre l'accesso a elementi non protetti da https da una fonte https genera un errore. Per questo gli uri sono solitamente indicati partendo con due barre (//) in modo da mantenere lo stesso tipo di connessione.

JQuery
Un framework che fornisce supporto per la navigazione e la modifica del DOM, gestione di eventi e l'uso di comunicazioni asincrone con il server (Ajax). Estendibile attraverso la creazione di plug-in. Non ha librerie di widget proprie, ma il progetto parallelo jQueryUI fornisce una prima libreria di widget (con l'aggiunta di altre librerie).

JQuery è un oggetto, chiamato indicato con: $. jQuery definisce allora una funzione $() che è un alias del costruttore jQuery (), e ha come parametro un selettore DOM. La variabile $ è un singleton, (oggetto creato una volta e mai più istanziato), che fornisce tutti i servizi quando chiamato, senza occupare lo spazio dei nomi.

Le parentesi dopo il dollaro accettano come argomento un selettore CSS e restituisce una sorta di lista di nodi del DOM. Successivamente si possono incatenare eventi, e specificarne eventualmente callback (i.e. `$("a").click(function() { // do something });`)

Il document.ready e il window.load sono due momenti specifici a partire dai quali si possono arrivare gli script. Solitamente gli script di setup dell'applicazione vengono fatti partire da document.ready, ovvero quando l'elemento DOM è stato completamente caricato. In window.load sono state caricate anche tutte le risorse esterne (es. video esterni).
Rispettivamente sono eseguiti con:

```
$(document).ready(function(){
      // codice da eseguire al ready
})
$(window).load(function(){
      // codice da eseguire al load
})
```

Selettori JQuery: oltre ai classici selettori id e nome della classe, si ha ad esempio, :selected - :visible - p[id^="basic"] che forniscono rispettivamente (tutti gli elementi selezionati, tutti quelli visibili e tutti i p il cui id inizia per "basic").

Alcune funzioni di modifica del DOM sono ad esempio: addClass(), removeClass(), before(), after(), css(), html(), text(), …
Tra gli effetti e le animazioni si ha: hide(), show(), fadeIn(), fadeOut(), slideDown(), slideUp(), …
Essendo possibile associare funzioni di callback ad eventi, sono disponibili tanti eventi quanti ne mette a disposizione il browser, associabili ad elementi del DOM, come mouseover(), keypress(), click(), … Nota, e.preventDefault() impedisce il comportamento di default previsto per l'evento e.

Se si vuole associare un comportamento ad eventi, sul model DOM che non esistono ancora, è necessario utilizzare la funzione "on()", specificando come primo parametro il particolare evento.

Una semplice chiamata Ajax con JQuery:

```
$.ajax({
   url: 'app.php',
   success: function(data) {
          $('.result').html(data);
          alert('Caricamento effettuato');
   },
   error: function(data) {
          alert('Caricamento impossibile');
   }
})
```

Altre funzioni equivalenti, utilizzate per maggior rapidità sono: $.get(), $.post() e $.put().

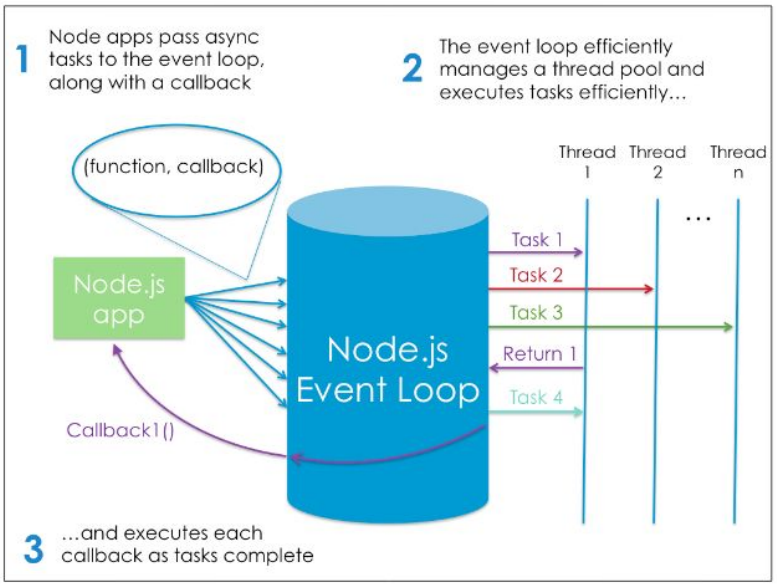
Una alternativa al call-back è usare le promesse. Il metodo $.ajax() è già una promessa, per cui si possono creare catene di funzioni **then()** con due parametri, la funzione da chiamare in caso di successo e quella in caso di insuccesso:

```
var good = function(data) {
    $('.result').html(data);
     alert('Caricamento effettuato');
};
var bad = function(data) {
     alert('Caricamento impossibile');
};
$.get('http://www.server.it/server').then(good, bad);
```

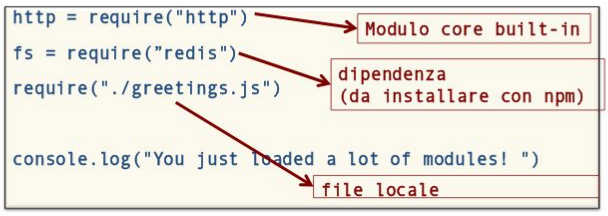
! la promessa è un oggetto che ha una funzione then() eseguita solamente se la prima non viene completata.

Node.js ed npm

Node è per design single-threaded, tutte le applicazioni eseguite da una singola istanza di Node, vengono messe in coda ed eseguite solo quando quella in esecuzione prima termina. NodeJS può porsi in ascolto su qualunque porta e implementare anche protocolli diversi da HTTP.



Node.js è quasi esclusivamente basato su funzioni asincrone e callback. La convenzione di node.js suggerisce di creare funzioni che accettano una funzione callback asincrona come ultimo parametro. La funzione callback per convenzione usa la sintassi error-first.
Node.js è estremamente modulare. E' possibile mettere codice indipendente in file javascript ed eseguirlo secondo necessità.



Per creare un modulo:

```
greetings = require("./greetings.js")
greetings.hello()
greetings.ciao()
```

greetings.js

```
hello = function() {
        console.log("\n Hello! \n")
}
ciao = function() {
        console.log("\n Ciao! \n")
}
module.exports.hello = hello;
module.exports.ciao = ciao;
```

Template

Una pagina priva di contenuti ma completa dal punto di vista grafico e tipografico, in cui vengono inseriti appositi elementi, detti placeholder, che verranno sostituiti con un valore calcolato dall'applicazione. I placeholder hanno una sintassi speciale, per distinguerli dal contenuto normale, e delle regole di composizione e arricchimento funzionale.

Interpolazione di variabili

0. Le virgolette sono caratteri ammessi in HTML, e sono caratteri fondamentali nel markup:

```
var a ="<p class='" + obj.class +"'>" + obj.testo + "</p>"
```

1. Molti linguaggi di programmazione hanno funzioni e operatori per fare formattazione controllata di valori in una stringa sulla base dell'ordine in cui i valori vengono specificati:

```
sprintf(v, "%d + %d = %d \n", a, b, a+b);
```

2. PHP nasce come motore di template (PHP significa Hypertext Preprocessor). Il meccanismo (poi esportato in altri linguaggi) di permettere embed di comandi eseguibili dentro ad una pagina HTML è un chiaro esempio di template:

```
<title><?php echo $page_title; ?></title>
```

3. Gli here document (heredoc) sono sintassi (dipendenti da linguaggio a linguaggio) per avere valori stringa lunghi più di una riga nel programma:

Python

```
V = """\
<html>
<body>
  <h1>Hello {name}!</h1>
</body>
</html>
""".format(name="John")
```

PHP

```
$V = <<<EOS
<html>
<body>
  <h1>Hello $name!</h1>
</body>
</html>
EOS
```

4. I template server-side servono per generare un documento pronto a partire da un ambiente di computazione e un documento HTML vuoto con variabili da interpolare (es. Pug per JS/Node).

HTML (sort of...)

```
doctype html
html(lang='en')
 head
  title #{title}
 body
   h1 #{title}
   div.container
    p #{message}
```

Javascript – Express.js

```
app.set('view engine', 'pug')
app.get('/', function (req, res){
  res.render('index.pug', {
    title: 'Ciao a tutti',
    message: Prima prova!'
  })
})
```

Client-side?
Javascript non aveva né un meccanismo di interpolazione di variabili, né una sintassi per stringhe lunghe (heredoc). Posso usare una stringa normale con i soliti problemi di lunghezza della riga e mancanza di interpolazione.

5. Idea: nascondere il testo del template nell'HTML stesso, ma inerte e inattivo, e poi quando serve prenderlo a inserirlo nel DOM in una parte visibile. Tuttavia debbo ricordarmi di proteggere il template HTML dalla sua visualizzazione, esecuzione, e/o indicizzazione prima del tempo o nel posto sbagliato.
6. Nell'HTML living standard è previsto un nuovo tag `<template>`. Il contenuto di `<template>` è inerte e non fa parte del documento: le immagini non vengono caricate, gli script non vengono eseguiti, il contenuto non occupa spazio nel layout, … Al momento utile, il template viene attivato importando il suo contenuto nel DOM come discendenti di un nodo del DOM esistente. In quel momento i contenuti si visualizzano, le immagini si caricano, gli script vengono eseguiti. Tuttavia non si hanno tecnologie molto pratiche per utilizzare i `<template>`.

```
var t = $('#tpl').content
$('img', t).src='fig1.gif'
var c= document.importNode(t)
$('#out').appendChild(c)
```

```
<template id="tpl">
  <p>Testo</p>
  <img src=""/>
</template>
```

7. Una nuova sintassi per definire stringhe multi-linea con interpolazione di variabili. Tre elementi fondamentali:
- Backticks come delimitatori: `
- I new line fanno parte della stringa
- Interpolatori: ${varName}

Purtroppo l'interpolazione avviene solo durante l'assegnazione a variabile, e non può essere controllata né posticipata.

```
var firstName = 'Jane';
var x = `Hello ${firstName}!
How are you
today?`;
```

x vale "Hello Jane!
How are you
today?"

8. Interpolation dei poveri

```
String.prototype.tpl = function(o) {
  var r = this ;
  for (var i in o) {
    r = r.replace(new RegExp("\\$"+i, 'g'),o[i])
  }
  return r
}
```

```
var t = "$0+$1=$2";
var a = [a, b, a+b];
var r = t.tpl(a)
```

9. Se uso <template>, e String.prototype.tpl(), posso ottenere risultati piacevoli in modo molto comodo:

```
<template id="tpl1">
  <p class="$c">$t</p>
</template>
<template id="tpl2">
  <span onclick='doX("$parameter")'>$testo</p>
</template>
...
$('#out').append($('#tpl1').html().tpl({c:"c1", t:"Prova"}));
$('#out').append($('#tpl2').html().tpl({
      parameter:"pippo.html",
      testo:"clicca qui"
}));
```

10. In alternativa, posso ricorrere a framework appositi. Negli ultimi anni ne sono nati tre di grande successo e con una ragionevole somiglianza tra loro:
- AngularJS (poi evoluto in una vera e propria piattaforma applicativa con poca attinenza con il templating)
– ReactJS
– VueJS

Template Javascript

> *Mustache.js*

Mustache è una semplice sintassi per template testuali con interpolazione di variabili. È logic-less, ovvero nessun flusso di controllo, nessuna istruzione per cicli o condizioni, separation of logic, non c'è nessun modo per fare embedding di logica applicativa nel template.

```
Ciao {{name}} {{surname}}!
Hai vinto {{value}} euro!
{{#taxed}}
  Beh, {{net}} euro, con le tasse.
{{/taxed}}
```

```
{
  "name": "Fabio",
  "surname": "Vitali",
  "value": 10000,
  "net": 10000*0.78,
  "taxed": true
}
```

```
Ciao Fabio Vitali!
Hai vinto 10000 euro!
Beh, 7800 euro, con le tasse.
```

> *Handlebar.js*

Handlebar è una semplice estensione di Mustache.js che permette di accedere agli oggetti annidati dentro alle variabili da interpolare e di ottenere l'effetto di cicli e istruzioni condizionali usando bocchi contestuali.

```
<div>
  <h1>{{title}}</h1>
  <h2>by
   {{#each authors}}
    <b>{{#if title}}{{title}}{{/if}}
       {{name}} {{surname}}
    </b>{{#unless @last}},{{/unless}}
   {{/each}}
  </h2>
  <div class="body">{{body}}</div>
</div>
```

```
{
  title: "My new post",
  authors: [{
    name: "John",
    surname: "Smith",
    title: "Dr."
  },{
    name: "Mandy",
    surname: "Brown"
  }],
  body: "This is my first post!"
}
```

My new post

by Dr. John Smith, Mandy Brown

This is my first post!

> *Angular.js*

È un framework per la semplificazione della realizzazione di Single Page Application, basato sui pattern Model-View-Controller (MVC) e Model-View-ViewModel (MVVM). Ogni parte di una pagina web può diventare la view di un model, e via Javascript si crea il controller che ne gestisce bi-direzionalmente la creazione e lo sviluppo.

Angular.js: **model**
Il model è una struttura dati (tipicamente espressa in JSON o come oggetto Javascript) che può esistere nella pagina web, essere caricata dinamicamente via Ajax, o essere calcolata a seguito di computazioni client-side.

```
{
  "users" : [{
    "name": "Fabio Vitali",
    "tel": "051 2094872",
   "studenti": 120
  },{
    "name": "Angelo Di Iorio",
    "tel": "051 2094889",
   "studenti": 60
  }]
}
```

Angular.js: **view**
La view è un frammento HTML nella pagina principale, che viene decorato con speciali attributi (che iniziano con `ng-`) e placeholder (con la sintassi `{{nome}}`).

```
<html>
  <body ng-app="simpleApp">
    <h1>Un semplice test Angular</h1>
    <table border="1" ng-controller="listCtrl">
      <tr>
       <th>Nome</th><th>Tel</th><th>studenti</th>
      </tr>
      <tr ng-repeat="person in list">
        <td>{{person.name}}</td>
        <td>{{person.tel}}</td>
        <td align='right'>{{person.studenti}}</td>
      </tr>
    </table>
  </body>
</html>
```

Angular.js: **controller**
Il module è il contenitore di tutte le parti che costituiscono un'applicazione AngularJS. Il controller lega la view al model. Il controller esplicitamente fa riferimento agli oggetti che vuole vedere. Questa si chiama dependency injection. In questo caso facciamo riferimento ai seguenti oggetti:
– $scope: lo scope di visibilità della view
– $http: l'oggetto che si occupa di comunicazioni via Ajax.

```
var simple = angular.module('simpleApp', []);
simple.controller('listCtrl', function($scope, $http){
     $http.get('list.json').success(function(data) {
        $scope.list = data.users;
     });
  });
```

Angular.js: **bi-directional binding**

Ovvero legame bidirezionale: se un elemento HTML è modificabile (es., un input di qualche tipo), allora cambiando il value dell'elemento si cambia anche il campo relativo nell'oggetto scope, e viceversa.

```
<tr ng-repeat="person in list">
  <td>{{person.name}}</td>
  <td>{{person.tel}}</td>
  <td align="right">{{person.studenti}}</td>
  <td><input type="number" ng-model="person.studenti"/>
    <button ng-click="add(person,10)">+10</button>
    <button ng-click="add(person,-10)">-10</button></td>
</tr>
...
var simple = angular.module('simpleApp', []);
simple.controller('simpleAppCtrl',function($scope, $http){
  $http.get('list.json').success(function(data) {
    $scope.list = data.users;
    $scope.add = function(person,n){person.studenti+=n};
  });
});
```

Angular.js: **dependency injection**
Per evitare effetti collaterali imprevisti, ogni controller lavora in un ambiente isolato privo di accesso all'ambiente esterno, in particolare le variabili globali. Per fornire accesso a questo o quell'oggetto esterno, è necessario passarglielo esplicitamente tra i parametri della funzione del controller (iniezione degli oggetti utilizzati, o dipendenze). Questo isola in un punto solo l'identificazione delle dipendenze dell'algoritmo, e permette di realizzare test sofisticati modificando il tipo e i permessi degli oggetti iniettati.

```
simple.controller('simpleCtrl',function($scope, $http){
  $http.get('list.json').success(function(data) {
    $scope.list = data.users;
    $scope.add=function(person,n){person.studenti+=n}
  });
});
```

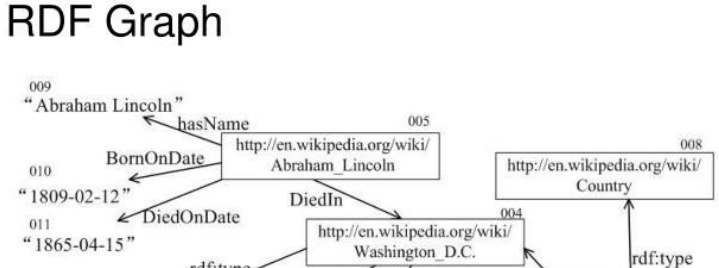
Semantic Web

URL: identifica cosa esiste nel web
URI: identifica, sul web, cosa esiste
IRI: identifica, sul web, per ogni lingua

Semantic web
Rendere i dati comprensibili anche alle macchine, non solo agli umani. Per far ciò è necessario che le informazioni siano in una forma leggibile dalle macchine. Il linguaggio quindi è consigliabile non sia HTML ma RDF, ad esempio. Si passa dal web dei documenti al web dei dati.

Linked data
Insieme di best practices per pubblicare dati sul web. Un modo è utilizzare URI come nomi per le cose, utilizzare gli standard RDF e SPARQL per fornire informazioni a qualcuno che cerca un certo URI.

RDF (Resource Description Framework)
Ha tre elementi: **soggetto** (risorsa identificabile con un URI), **predicato** (specificazione della relazione riutilizzata ed



identificata da un URI) e **oggetto** (una risorsa a cui il soggetto è collegato). Ogni insieme di asserzioni RDF può essere rappresentato e manipolato come un grafo orientato.
Possono essere presenti nel grafo dei “black nodes”, ovvero risorse non identificate. Questi nodi possono contenere comunque informazioni collegate ma il loro uso è sconsigliato. RDF Tutle è una sintassi per RDF più leggibile:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

Accessibilità
Specifica ARIA (Active Rich Internet Applications).
Screen reader: software in grado di intercettare ciò che compare sullo schermo e leggerlo, la lettura avviene tramite sintetizzatore vocale o attraverso un display Braille. La document mode dello screen reader è utile in caso di lettura di documenti complessi.

I requisiti formali sono: perceivable, operable, understandable e robust.

React.js
Nasce in Facebook. Modello generico di creazione di applicazioni che vengono implementate in base all’ambiente di sviluppo, es web diventano frame JS, su android Java, su iOS objective-C/Swift. Frammenti che vengono ottimizzati in base all’ambiente in cui girano. JSX prende il posto di Javascript, è un mix di javascript e XML/HTML. Il metodo render() è il momento in cui gli elementi vengono effettivamente posizionati nell’applicativo.

Vue.js
…

Search Engine Optimization
SERP (Search Engine Results Page), pagina di mostrata dal motore di ricerca in risposta ad una query. I motori di ricerca sono costituiti da tre processi, un crawler, un indicizzatore e un motore di query.
Il crawler, conosciuto anche come spider, è un bot internet che naviga nel WWW con lo scopo di indicizzare. Prende gli URL da crawl precedenti e tramite sitemaps XML prova a trovare pagine nuove o aggiornate. Elementi utili che interessano il web crawler sono: quanto impiega la pagina a caricare, quanto è importante un URL e se ci sono hyperlink ad altre pagine.

Robots exclusion standard
Il protocollo di esclusione robot indica, nel gergo di internet e più in generale del web, le regole indicate dai gestori di un sito web ai crawler che lo visitano, chiedendo di applicare restrizioni di analisi sulle pagine del sito. Esse sono contenute nel file robots.txt.

Indexers

Un indexer fornisce tre servizi, canonicalizzazione (trova l'URL canonico), WRS (effettua il render di una pagina web) e Page ranker.

Domande progetto

- Fabio chiede le tecnologie da cui dipende il progetto (l'unica obbligatoria è Node.js).
- Esiste API Javascript per fare query a Wikipedia (usarla), oltre a SPARQL.
- Usare moduli già pronti per usare le API Youtube (Fabio vuole che copiamo il più possibile da internet).
- Se non ci sono modi semplici per capire se due video sono troppo simili, chissenefrega, amen, l'importante è provarci e dare una giusta spiegazione (video ufficiale e video live sono diversi, video ufficiale e video messo da Paolino sono uguali).
- Quando si accede per la seconda volta al sito, ci si trova l'ultimo video visto, se ci si trova per la prima volta è presente una lista di video di partenza (si può mettere come modale o altro, verrà mostrato solo la prima volta che si accede al sito).
- Non arrivare con il database delle preferenze vuoto alla discussione, simulare un utilizzo per la presentazione.
- Giocare con le promises in caso di esecuzione asincrona server-side.