Dott. Antonio Giovanni Lezzi

NOTE IMPORTANTI

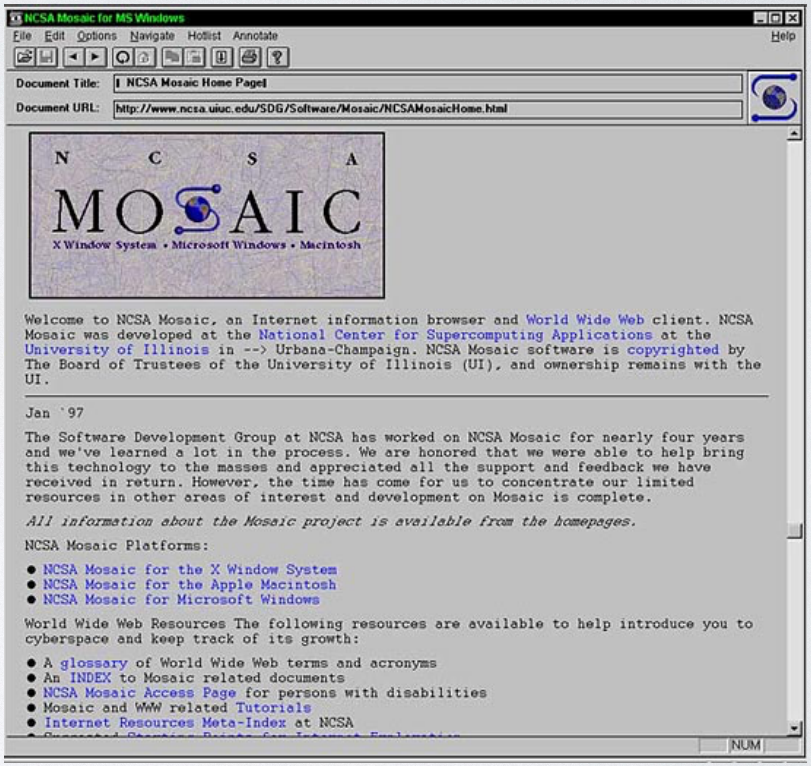
- C'è una sola via da percorrere per imparare a programmare, dovete: ascoltare la trattazione teorica, leggere codice, scrivere codice, porsi delle domande costruttive ed imparare dagli errori
- Per questo dovrete sempre scrivere il codice degli esempi ed eseguirlo per poi vedere che cosa succede: giocate con il codice, modificatelo come volete, la cosa peggiore che può capitarvi è che il programma non funzioni
- Provate a fare esercizi sempre nuovi e porvi dei target oltrepassando i limiti
- Mai rimandare a domani quello che puoi far oggi: gli argomenti si accumulano

PROGRAMMA DELLA GIORNATA

- Storia
- Standard
- Web ai giorni nostri
- Pagine web
- Strumenti di sviluppo
- Browser
- Anatomia di una pagina
- Progettazione

STORIA

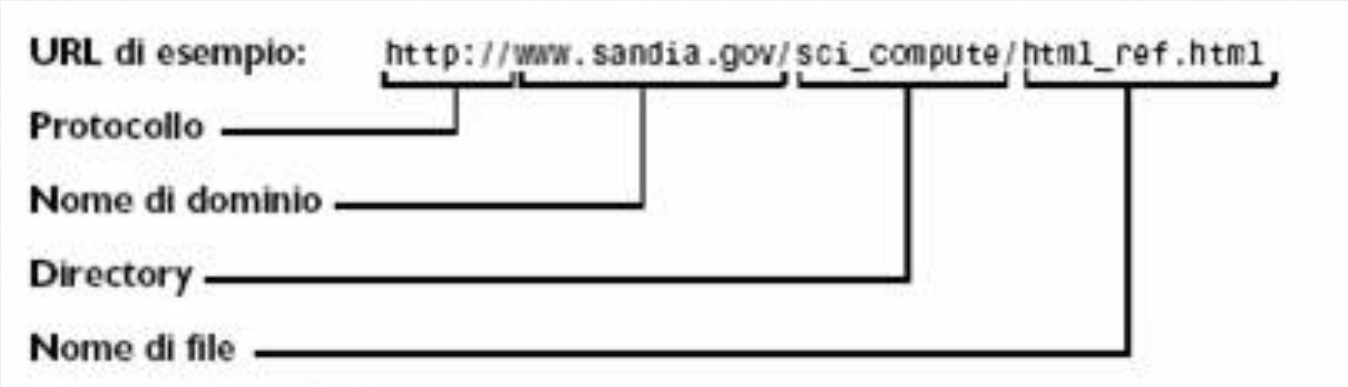
- La storia del WWW comincia al Centro Europeo per le Ricerche Nucleari di Ginevra nel 1991
- Nel Gennaio '92 era disponibile, su FTP anonimo, una prima versione di browser a linee. Marc Andreessen, uno studente del National Center for Supercomputing Applications (NCSA) rilasciò la prima versione di un browser ipertestuale che gestiva, oltre ad http, anche ftp, gopher ed altri protocolli TCP/IP: **Mosaic**
- Il protocollo HTTP introduce il nuovo concetto di **hyperlink**: diventa possibile, con un semplice click, **attingere informazioni in maniera trasparente da nodi differenti, a partire dallo stesso documento**
- Oltre a ciò, si sviluppano le tecniche per immagazzinare nel documento testi, immagini e suoni contemporaneamente



STORIA

- Per la natura stessa di HTTP, cioè di essere indipendente dal tipo di dato che deve essere gestito, si è ritenuto di rendere uniforme (**uniform**) il modo di identificare i siti (**locator**) che sono la fonte delle risorse (**resource**) che HTTP deve poi gestire. La notazione è del tipo:

- protocollo://sito.internet/percorso_dati



STORIA

- Questo tipo di notazione ci consente di avere una sola struttura per tutti i tipi di protocollo gestibili dal **World Wide Web**
- Avremo modo di vedere più avanti applicata ad un browser rendere il WWW uno dei mezzi più potenti di comunicazione su rete di tipo Internet
- La sigla **HTML** è l'acronimo di **HyperText Markup Language** e tradotto letteralmente indica un "linguaggio a marcatori per l'ipertesto"
- Non è un linguaggio di programmazione, ma permette di indicare come disporre gli elementi all'interno di una pagina
- *È un linguaggio di scripting che contiene anche la formattazione del testo, ossia i codici che indicano come e dove deve essere scritta una parte di testo*
- L'HTML è un linguaggio sviluppato su piattaforme incrociate, e quindi funziona (e viene letto) su qualunque piattaforma, su qualunque tipo di computer e con qualunque sistema operativo

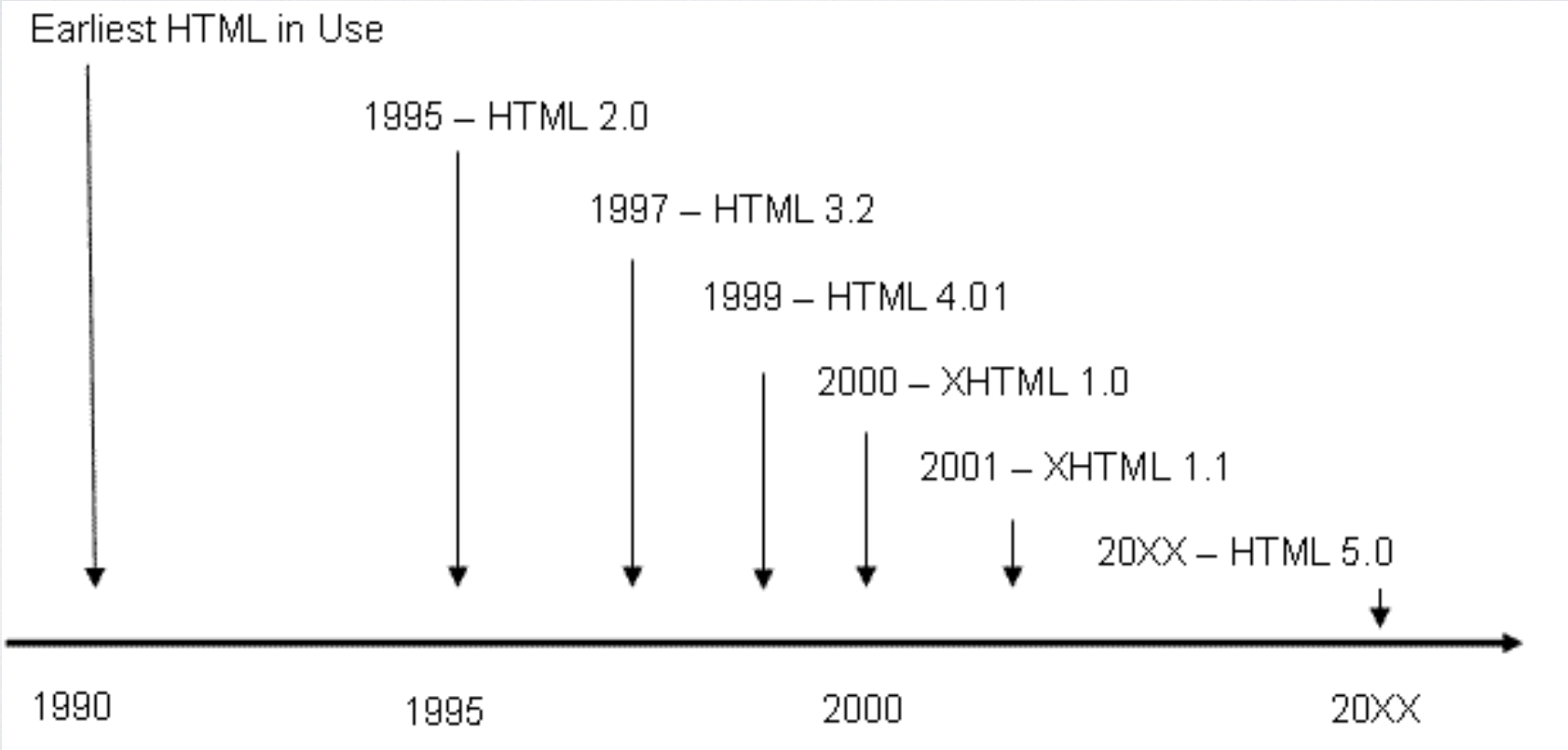
STORIA

- **L'HTML è più che sufficiente a realizzare un sito web** (statico), senza l'ausilio di altri linguaggi di scripting o di programmazione e altre tecniche di implementazione, come javascript, Java, CSS, PHP, ASP o il più recente DHTML, l'HTML dinamico

- Con questi ulteriori strumenti si potrà ottenere una migliore disposizione degli oggetti all'interno della pagina e soprattutto una maggiore interattività tra la pagina e gli utenti, ma gli strumenti che l'HTML ci mette a disposizione costituiscono una discreta attrezzatura per realizzare anche un sito web accattivante

- Un ipertesto è un documento che ha la possibilità di una lettura non lineare, ossia è possibile saltare arbitrariamente da un argomento all'altro selezionando alcuni collegamenti disposti all'interno delle pagine, e ha la possibilità di visualizzare immagini, suoni o animazioni, elementi multimediali

STORIA


Earliest HTML in Use
1995 – HTML 2.0
1997 – HTML 3.2
1999 – HTML 4.01
2000 – XHTML 1.0
2001 – XHTML 1.1
20XX – HTML 5.0
1990
1995
2000
20XX

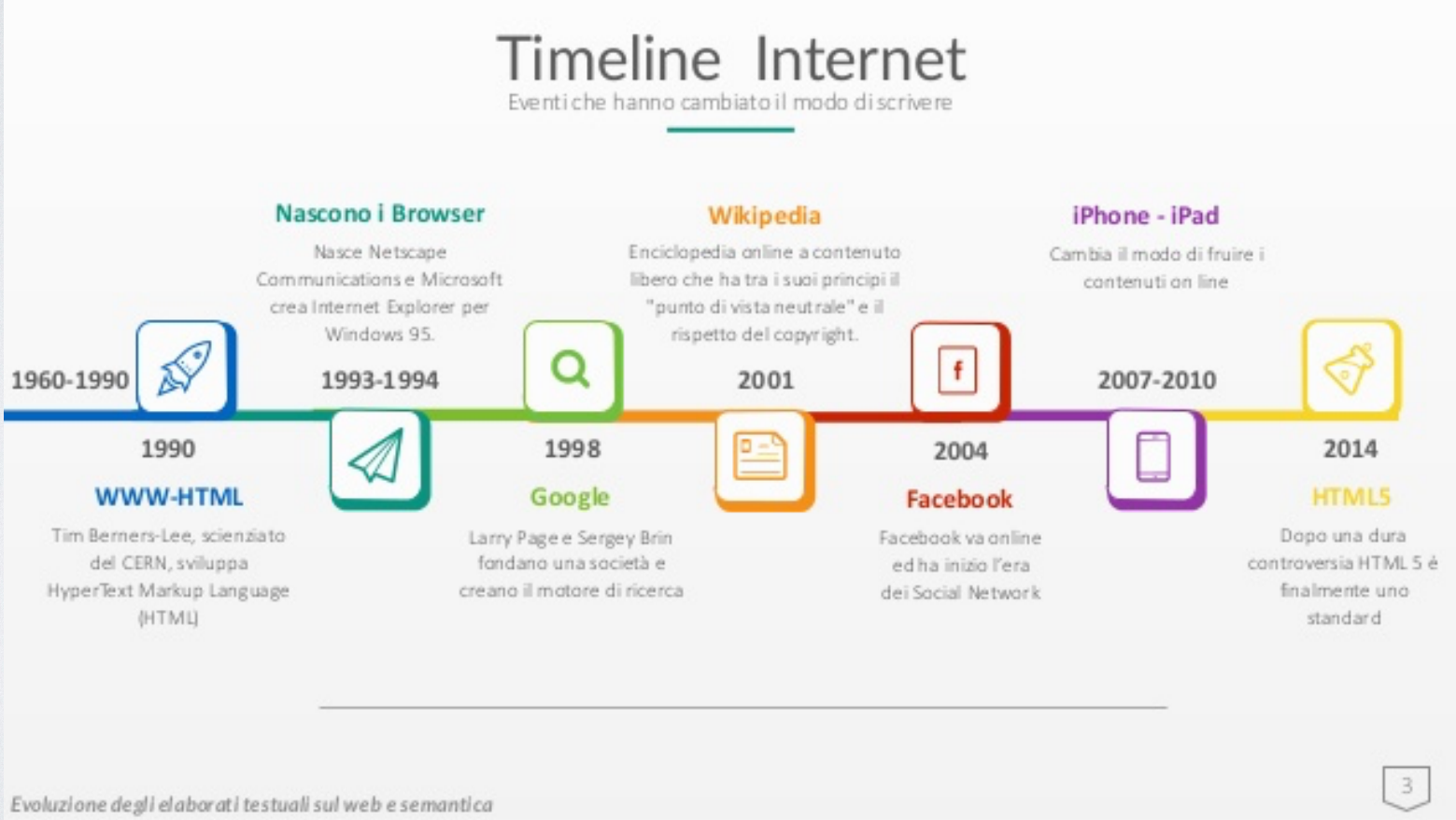
GLI STANDARD

- Da quando Tim Berners Lee, nel 1991, ha definito la prima versione del linguaggio, fino ai giorni nostri, HTML ha continuato ad evolversi fino a maturare nello standard HTML 5.
- Il secondo rilascio delle specifiche (HTML 2) è stato pubblicato dallo IEFT nel 1995, mentre già stava nascendo il W3C: World Wide Web Consortium l'organizzazione fondata dallo stesso Berners Lee con CERN e MIT, che oggi vede coinvolti numerosi istituti universitari e i più importanti attori tecnologici sulla scena mondiale come Apple, Microsoft, Google, Facebook, IBM, Adobe, e moltissimi altri.
- Il W3C si occuperà di traghettare lo standard attraverso la versione 3.2 (1997) fino alla definizione di HTML 4 nel 1998 e HTML 4.01, la versione che ha sostenuto la crescita del Web di tutti gli anni 2000 e che assurgerà a standard ISO proprio a cavallo del terzo millennio.
- Proprio negli anni 2000 nasce XHTML, un filone parallelo che cerca di dare maggior rigore all'HTML, definendolo come "applicazione XML" ne sono già state rilasciate due versioni (XHTML1.0 e XHTML1.1), molto simili a HTML 4.1, poi praticamente abbandonate già nel 2011 quando il W3C decide di affermare lo standard HTML 5 (tutt'ora sussiste una definizione di XHTML5.1 nella bozza delle specifiche HTMI5).

WEB AI GIORNI NOSTRI

- Sarà fatto un excursus delle diverse versione di HTML arrivando possibilmente fino all'ultima versione
- HTML5, che è anche la versione che nasce appositamente per uscire dal solo ambito Web e diventare piattaforma per la creazione di applicazioni, anche desktop e mobile.
- La specifica HTML5 infatti si compone della definizione di:
  - una sintassi per il markup più efficace e adatta alle esigenze più moderne, con l'introduzione di specifici controlli per i form o degli attributi "data-" da arricchire i tag di informazioni specifiche;
  - una serie di API che consentono di gestire, a livello approfondito aspetti come network, multimedia, e hardware dei dispositivi. (gestione del video e dell'audio al monitoraggio delle batterie di un device ecc)
- Questo sviluppo dello standard ha dato il via alla generazione delle cosiddette App mobile ibride, che sfruttano sia HTML5, per creare app che si possono distribuire, come quelle native

STORIA


Timeline Internet
Eventi che hanno cambiato il modo di scrivere
1960-1990
1990
WWW-HTML
Tim Berners-Lee, scienziato del CERN, sviluppa HyperText Markup Language (HTML)
1993-1994
Nascono i Browser
Nasce Netscape Communications e Microsoft crea Internet Explorer per Windows 95.
1998
Google
Larry Page e Sergey Brin fondano una società e creano il motore di ricerca
2001
Wikipedia
Enciclopedia online a contenuto libero che ha tra i suoi principi il "punto di vista neutrale" e il rispetto del copyright.
2004
Facebook
Facebook va online ed ha inizio l'era dei Social Network
2007-2010
iPhone - iPad
Cambia il modo di fruire i contenuti on line
2014
HTML5
Dopo una dura controversia HTML 5 è finalmente uno standard
Evoluzione degli elaborati testuali sul web e semantica
3

HTML: LINK E IPERTESI

- La potenza di Internet consiste nell'essere un insieme esteso di contenuti connessi tra loro, l'HTML e i link sono alla base di questo meccanismo che consente di muoversi rapidamente tra testi, immagini, video, applicazioni e quant'altro creando percorsi i propri percorsi di navigazione in totale autonomia.
- Per accedere rapidamente a una ormai enorme mole di informazioni, sono nati i motori di ricerca, che si basano su una serie di elementi "semantici" per rispondere adeguatamente alle richieste degli utenti
- I motori di ricerca si basano sull'analisi del testo (o del contenuto) e dei collegamenti, fino a sfruttare i comportamenti degli utenti per catalogare le pagine e stabilire il livello di rilevanza (ranking) che hanno a seconda della singola query di ricerca.
- L'ottimizzazione dei contenuti per favorire il posizionamento della pagina per specifiche ricerche si dice SEO (Search Engine Optimization)

COSA SERVE

- Per creare una pagina HTML non occorrono particolari strumenti chissà quanto sofisticati. Basta un normale editor di testo, va benissimo il Notepad di Windows o il Vi o Xedit di Unix/Linux.
- Naturalmente esistono decine di editor specializzati e molti di essi sono anche WYSIWYG (What You See Is What You Get), cioè permettono di creare la pagina disponendo tutti gli elementi proprio come essi appariranno una volta avviato il browser
- Browser. Naturalmente occorre un programma che sia in grado di leggere il file HTML e di visualizzarlo sullo schermo come una pagina; questo per poter vedere il lavoro che abbiamo fatto
- Anche di browser ne esistono decine; i più conosciuti sono Internet Explorer/Edge e FireFox, Chrome
- Questi browser non sono uguali; sono stati messi sul mercato in tempi diversi e i progettisti hanno avuto indicazioni diverse su come implementare alcune funzionalità. Può capitare che una funzionalità sia implementata su un browser ma non sull'altro oppure che per eseguire la stessa funzione sui due browser occorrano due codici distinti; qualora dovesse capitare

COSA SERVE

- Per iniziare basta pochissimo:
- Un browser, ne abbiamo almeno uno già installato nel PC o nei device che utilizziamo.
- Un editor di testi, anche di questi ce ne sono a bizzeffe e tipicamente ce n'è uno in ogni ambiente operativo
- L'importante è ricordare che non dobbiamo utilizzare programmi come Word, OpenOffice o WordPad, che sono "word processor" evoluti e non servono allo scopo

PAGINE WEB

- Con HTML quindi indichiamo, attraverso i tag, quali elementi dovranno apparire su uno schermo e come essi debbano essere disposti.
- Una pagina HTML è rappresentata da un file di testo, ovvero un file che possiamo modificare con programmi come notepad e in genere hanno un nome che finisce con l'estensione .html.
- Ciò significa che HTML non possiede i costrutti propri della programmazione, come i meccanismi "condizionali", che consentono di reagire in modo diverso a seconda del verificarsi di una condizione ("in questa situazione fai questo, in quest'altra fai quest'altro"), o i costrutti iterativi ("ripeti questa azione, finché non succede questo").
- Nota: Si parla in questo caso di linguaggio dichiarativo, che serve appunto ad indicare cosa deve apparire sullo schermo (testi, immagini, suoni, etc.), dove e in che sequenza. Nel caso di linguaggi in cui specifichiamo algoritmi precisi con "strutture di controllo", come C, C++, Java, o anche PHP e JavaScript, parliamo di "linguaggi imperativi".

PAGINE WEB

- Per lavorare con HTML è utile conoscere tutto il cosiddetto "stack", ovvero tutte le tecnologie necessarie a realizzare un sito o una applicazione.
- In passato si utilizzavano alcuni tag HTML per definire font (tipi di carattere), i colori o le dimensioni degli oggetti sullo schermo. Oggi il quadro è definitivamente cambiato e molte di quelle funzionalità sono deprecate favorendo una divisione dei compiti più chiara tra diversi strumenti:
- **HTML:** serve a definire quali sono gli elementi in gioco, stabilire collegamenti (link) tra le pagine e l'importanza (non la forma o il colore) che hanno i testi, creare form per gli utenti, fissare titoli, caricare immagini, video, etc.
- **CSS:** "fogli di stile". Si tratta di una serie di regole che permettono di definire l'aspetto (lo stile) che devono assumere gli elementi sulla pagina. Dimensioni, colori, animazioni, ogni caratteristica visuale può essere manipolata.
- **JavaScript:** è un linguaggio di programmazione che consente di manipolare veramente qualunque cosa nella pagina HTML: lo stile, i contenuti della pagina, ma soprattutto l'interazione con l'utente. Ci permette di creare la logica dell'interfaccia utente (o anche di una applicazione) e di sfruttare le API messe a disposizione dal browser: dalla gestione del mouse al touch, dalla manipolazione delle immagini, alle richieste di dati dinamiche (in modalità Ajax) alla gestione di dati in locale

SBIRCIARE TRA LE PAGINE

- Una delle cose più interessanti di questo linguaggio è che il markup è scritto come un testo, perciò è possibile leggere codice scritto da altri.
- Tutti i browser infatti danno la possibilità di mostrare il codice HTML delle pagine e quasi tutti permettono di farlo
- Non ci sorprenda, né ci scoraggi, trovare un codice molto fitto e complesso dietro a pagine apparentemente molto semplici

STRUMENTI DI SVILUPPO

- Gli strumenti principali che servono a realizzare pagine HTML sono editor di testo, browser, software per manipolare foto o immagini vettoriali, tool per la progettazione e la messa online del sito o il deploy dell'app.

- Un buon ambiente di sviluppo può fare la differenza sia quando iniziamo a sviluppare in HTML, sia quando abbiamo a che fare con la manutenzione di lavori già fatti

- Per ogni lavoro avere gli attrezzi giusti è fondamentale

EDITOR DI TESTO

- La scelta dell'editor è piuttosto importante, è il principale compagno di strada in questo percorso e tipicamente diventa un'estensione naturale del pensiero
- Ce ne sarebbero numerosi altri da citare, oltre a suggerire alcuni nomi vediamo quali sono le caratteristiche e i criteri che ce li hanno fatti scegliere

SYNTAX HIGHLIGHTING

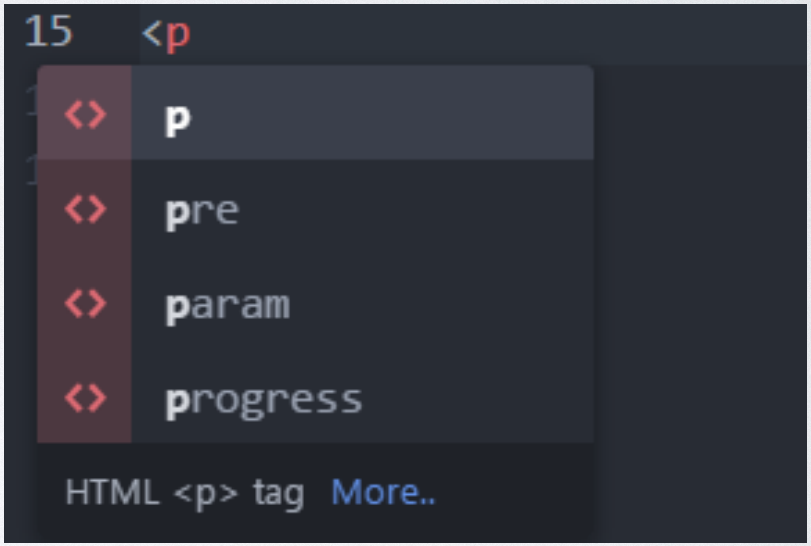
- Serve a evidenziare la sintassi del linguaggio e rendere più leggibile il codice sorgente
- Ogni tag, attributo, numero o parola chiave assume uno specifico colore e diventa subito riconoscibile
- Grazie a questa caratteristica emergono più facilmente eventuali errori di battitura e diventa anche più chiara la struttura della pagina

```html
<!doctype html>
<html lang="it">
<head><title>Ciao Mondo!</title></head>
<body>
  <h1>Ciao Mondo!</h1>
  <p>Questa è la nostra prima pagina HTML!</p>
```

AUTOCOMPLETAMENTO

- Mentre scriviamo il codice l'editor propone di completare automaticamente la parola o il comando che stiamo digitando. Ci sono diversi livelli di autocompletamento anche in base ai linguaggi.

- In HTML ad esempio può risultare comoda la chiusura automatica dei tag: quando finiamo di scrivere viene subito scritto automaticamente lasciandoci la comodità di scrivere dentro altro contenuto.



COMMUNITY E PLUGIN

- Anche se gli editor offrono poche funzioni di base, le esigenze possono portarci a voler aggiungere alcune caratteristiche e diventa importante valutare quanti e quali plugin sono disponibili, quanto è numerosa la base di utenti e quanto è attiva la comunità degli sviluppatori. Un plugin a volte può essere determinante

- Tra i plugin più utili ci sono ad esempio quelli che permettono di memorizzare e replicare degli snippet di codice (o il testo lorem ipsum ad esempio), quelli che permettono il confronto tra versioni diverse dei file (diff).

MACRO E SHORTCUT

- Le scorciatoie da tastiera sono estremamente importanti quando si sviluppa
- Tutti gli editor mettono a disposizione le combinazioni di tasti per le operazioni più comuni: copia, incolla, taglia, undo etc.
- Possiamo aver bisogno però di ulteriori automatismi, come richiamare uno snippet di codice, selezionare direttamente un tag etc
- Per ottenere questo può rivelarsi determinante un editor che consenta di registrare sequenze di azioni (macro) da poter associare a combinazioni di tasti (shortcut).

BROWSER

- **Il programma che interpreta il markup delle nostre pagine HTML e mostra a video tutto ciò che indichiamo si chiama Browser**
- La parola tradotta letteralmente sta per "**colui che dà un'occhiata in giro**" ed è stata utilizzata per definire il tipo di programmi che usiamo per navigare sul Web.
- Per capire meglio come lavorare in HTML è importante soffermarci su alcuni degli aspetti più importanti che riguardano proprio questi strumenti, diventati nel tempo molto di più che semplici software per la "navigazione di Internet"
- Tra i compiti svolti dal browser, sono due quelli che ci interessano di più: il caricamento e la visualizzazione (rendering) della pagina HTML

BROWSER: CARICARE HTML

- Caricare la pagina significa acquisirne il contenuto, più tecnicamente il browser richiede che venga effettuato un trasferimento di file
- Perché sia possibile effettuare le richieste, la pagina HTML, come pure tutti i file ad essa correlati, è raggiungibile facendo riferimento ad un certo indirizzo, come questi che seguono:
- http://www.sito.it/guida/introduzione.html
- file:///C:/Sito/GUIDA/introduzione.html
- Semplificando possiamo dire che grazie agli indirizzi possiamo effettuare richieste:
  - **in "locale"**, quando i file si trovano sul device o sul computer su cui gira il browser
  - **in "remoto"**, quando i file sono su un server da contattare tramite internet (o altra rete).

BROWSER: CARICAMENTO

- La fase di caricamento della pagina è il primo punto cruciale quando si parla di perfomance
- un sito o una applicazione risultino gradevoli da utilizzare bisognerà fare in modo di minimizzare quanto più possibile i tempi di caricamento
- Alcuni di questi accorgimenti come la cache sono inseriti anche nei browser. Inoltre tutti i browser inoltre forniscono dei tool per gli sviluppatori che consentono, tra le altre cose, di monitorare i tempi di caricamento dei file

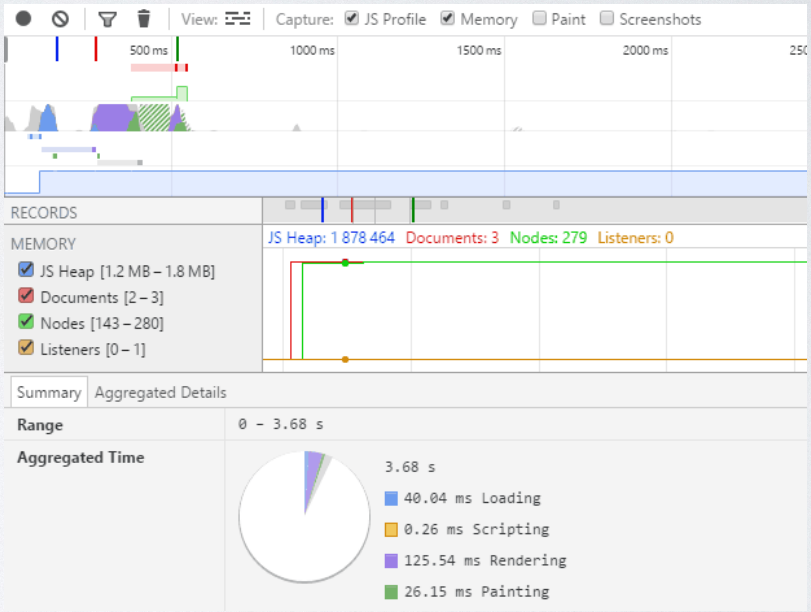
Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
02.%20II%...	Finished	document	Other	0 B	8 ms	
css?family...	200	stylesheet	02. Il Brows...	2.3 KB	157 ms	
page-loadi...	Finished	png	02. Il Brows...	0 B	3 ms	
data:image...	200	svg+xml	02. Il Brows...	(from cache)	0 ms	

BROWSER: RENDERING

- La seconda fase cruciale è quella della visualizzazione ovvero del cosiddetto rendering della pagina, la fase in cui il browser interpreta i documenti HTML e dispone sullo schermo gli elementi (testi, immagini, filmati) a seconda delle indicazioni ivi contenute.

- Il componente del browser che svolge questo compito è detto "Rendering Engine" (motore di rendering), che in sostanza si comporta come un sistema operativo in miniatura e che sfrutta direttamente le caratteristiche hardware del device su cui visualizziamo le nostre pagine.

- In questa fase quindi, specie nei progetti più complessi o quando le pagine sono destinate a dispositivi lenti o con poca memoria, bisogna fare attenzione a non sovraccaricare di lavoro l'engine del browser

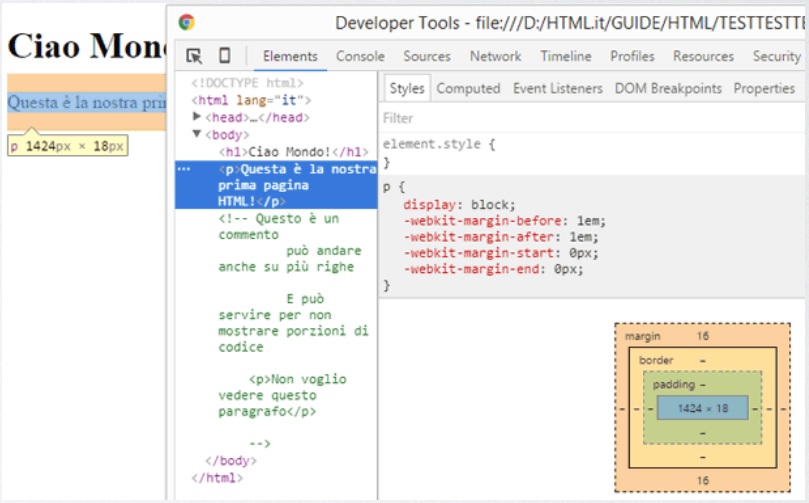


BROWSER PER SVILUPPATORI

- Il browser è fondamentale per testare il lavoro sotto il profilo visuale e non solo. I browser moderni come Google Chrome, Mozilla Firefox, Microsoft Edge e Apple Safari, forniscono gli "strumenti di sviluppo" grazie ai quali possiamo tenere sotto controllo diversi aspetti delle pagine HTML.

- Tra questi strumenti troviamo moltissime cose, dall'analisi del caricamento della pagina all'emulazione dei sensori dei device, dal debug step-by-step del codice JavaScript agli editor di SVG.

- L'esplorazione del codice e modifica dei CSS: Possiamo sezionare la pagina e vedere cosa c'è dietro a ogni elemento. Inoltre possiamo esaminare e modificare il CSS per effettuare delle modifiche e verificare "al volo" gli effetti.



BROWSER: DIFFERENZE

- Anche se praticamente tutti i browser moderni garantiscono una pressoché totale aderenza agli standard del W3C, rimangono ancora alcune piccole differenze nell'implementazione delle direttive, quindi nel modo in cui i singoli browser mostrano i contenuti.
- In sostanza quando lavoriamo al nostro HTML, possiamo trovarci a fare i conti con:
  - la compatibilità dei browser (specie quelli più datati possono dare problemi);
  - le capacità dei dispositivi (in termini di performance e banda ad esempio);
  - le tipologie di schermo cui sono destinate le pagine HTML.
- Se in passato l'idea dominante era quella di tentare di ottenere la stessa visualizzazione su tutti i browser, oggi è fondamentale che i contenuti siano proposti in modo adeguato alla singola situazione (es. mobile, large screen, desktop, vecchi browser, etc.).

ELEMENTI IN HTML

- In una pagina HTML tutti gli elementi sono connotati da tag (letteralmente "etichette")
- Si tratta di marcatori che evidenziano non tanto l'aspetto, quanto il senso, il ruolo, o l'organizzazione che vogliamo assegnare ai contenuti. Ad esempio se indichiamo un titolo con <h1> per noi le parole in quel titolo rappresentano il tema principale della pagina, al di là del modo in cui saranno visualizzate.
- **Questa visione "semantica" degli elementi è fondamentale**, ci aiuta a non perdere di vista il fatto che una pagina deve essere un luogo ordinato di informazioni, soprattutto sul Web. Se pensiamo ad esempio al classico contesto delle ricerche online, risulta naturale comprendere che tanto più una pagina sarà associabile a una keyword o a un tema, tanto più facile sarà trovarla. Per questo sarà importante curare cose come l'organizzazione del testo, la definizione dei titoli, dei link dei grassetti.
- Tuttavia in HTML possiamo anche definire lo scheletro dell'interfaccia utente di una app, e in questo frangente i tag diventano utili come supporto all'organizzazione del layout o alla definizione di aree specifiche per l'esperienza utente.
- In tutti e due i casi continuiamo a non parlare di "grafica", ma di struttura.

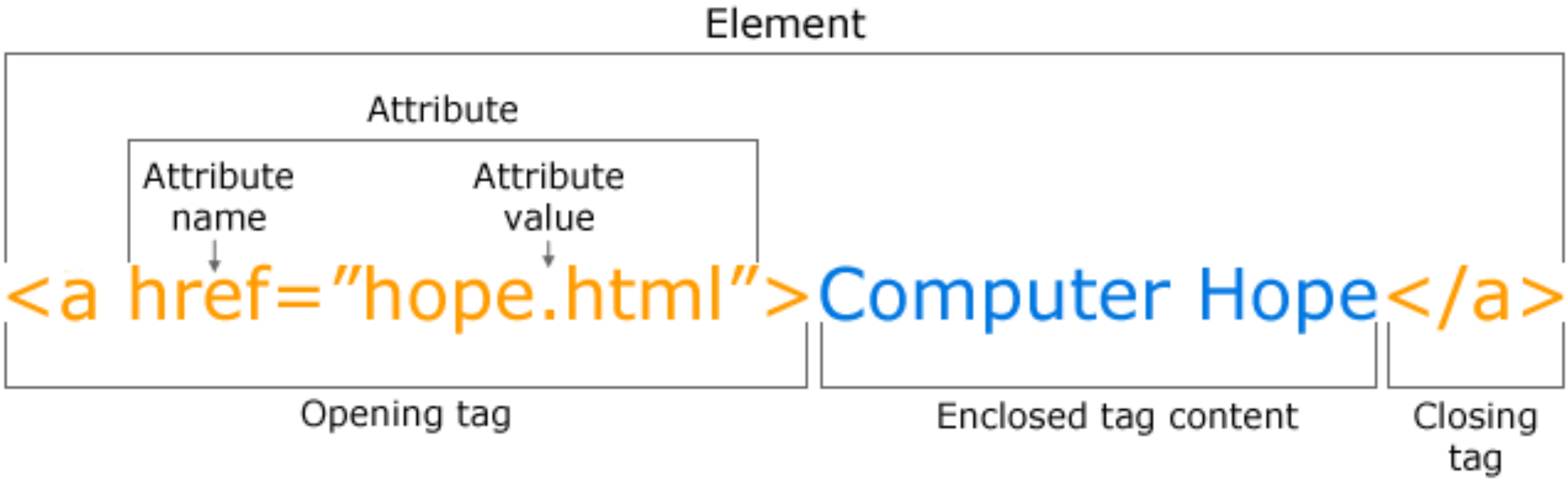
TAG FONDAMENTALI

- **I TAG sono i comandi che definiscono la struttura di una pagina HTML** e che permettono una corretta visualizzazione da parte del browser
- Si riferiscono al titolo della pagina, al colore e alle impostazioni dei caratteri e dello sfondo, all'impostazione del testo, delle immagini e dei collegamenti ipertestuali e in pratica permettono di visualizzare e gestire tutto quello che si vuole far vedere o tenere nascosto nella pagina.
- **Sono racchiusi tra parentesi angolari ("<" e ">")** e possono essere digitati indifferentemente sia in minuscolo che in maiuscolo
- La maggior parte dei tag ha anche un tag correlato che ne indica il termine e si scrive anteponendo al nome del tag il simbolo slash "/"
- **i due tag formano una coppia e si chiamano tag di apertura e chiusura**. Ad esempio, i tag che indicano l'inizio e la fine di un paragrafo si indicano con <P> e </P>

TAG FONDAMENTALI

- I tag fondamentali sono quelli che servono a far capire al browser che quello è un documento HTML, e che deve essere trattato nel modo appropriato
- Un documento HTML è tutto ciò che è contenuto nei due **tag <HTML> e </HTML>**. Questi due tag sono generalmente il primo e l'ultimo di ogni documento HTML, e le prime versioni dei browser ignoravano qualunque cosa fosse all'esterno di questi tag
- Gli attuali browser, sono molto più elastici, e provano ad interpretare qualunque documento come un documento HTML, per cui se si scrive qualcosa (del testo o anche dei comandi) all'esterno di questi tag, questo viene comunque riportato sullo schermo
- **Un documento HTML si compone di due parti**: una intestazione e un corpo, in inglese rispettivamente **header e body**

Breakdown of an HTML Tag



TAG UTILIZZABILI

- Per verificare se una data proprietà è supportata o no dai vari browser, si può utilizzare un servizio come When can I use fruibile dal sito http://caniuse.com/
- Alternativamente le varie funzionalità sono consultabili in modo rapido e intuitivo dal sito web http://html5readiness.com/ tramite un'infografica dinamica.
- Per vedere l'effetto degli esempi riportati si può utilizzare un editor HTML online come http://rendera.heroku.com/

ANATOMIA DI UNA PAGINA

- L'header è un contenitore di informazioni, e racchiude tutte le specifiche, i parametri e le funzioni che non sono utili direttamente all'utente finale, ossia il lettore, ma che vengono utilizzate dai browser, dai motori di ricerca e da tutti quei programmi in grado di leggere un file HTML, per poter meglio gestire il documento

- Il body è la parte che effettivamente contiene tutte le cose che vengono visualizzate o che servono ad impostare la visualizzazione degli oggetti. Ancora una volta, i moderni browser si dimostrano abbastanza robusti ed elastici, visualizzando la pagina anche se la parte di header manca del tutto e se mancano i tag che delimitano la parte del body

ANATOMIA DI UNA PAGINA

- L'intestazione di un documento HTML è quella parte di codice compresa tra i due tag <HEAD> e </HEAD>, e che quindi contiene quei tag utili ai browser, ai server web e ai motori di ricerca.
- Questa sezione contiene il titolo della pagina, e che è generalmente il primo tag dell'intestazione
- Una pagina potrebbe avere per esempio per titolo "Manuale di HTML - I TAG fondamentali" e ciò sarebbe visibile nella barra del titolo della finestra del browser.
- I tag che specificano il titolo della pagina sono <TITLE> e </TITLE>
- Nella sezione di header sono contenuti anche i META-TAG, che portano informazioni aggiuntive per il server web, e per i motori di ricerca che così possono classificare il documento senza caricarlo completamente in memoria
- L'header è inoltre utilizzato per definire particolari funzioni di script che verranno utilizzate all'interno del documento per dare un tocco di interattività alla pagina HTML

ANATOMIA DI UNA PAGINA

- Il corpo del documento contiene l'insieme del testo che deve venire visualizzato e i tag che fanno in modo da dare a quel testo una forma ben precisa, aggiungendo anche grafica, suono ed elementi multimediali
- **Il corpo del documento HTML si indica con i due tag <BODY> e </BODY>** ed è soprattutto del contenuto di questa sezione, appunto quella più importante del documento,
- Un documento HTML è un insieme di righe che hanno l'aspetto mostrato qui sotto

```
<HTML>

 <HEAD>
  <TITLE>Title Text</TITLE>
 </HEAD>

 <BODY>
  <H1>H1 Heading</H1>
  <P>Paragraph 1</P>
  <P>Paragraph 2</P>
 </BODY>

</HTML>
```

ANATOMIA DI UNA PAGINA

```
<HTML>
    <HEAD>
        <TITLE>Titolo del documento</TITLE>
        ....
    </HEAD>
    <BODY>
        ....
        ....
    </BODY>
</HTML>
```

ESEMPIO

- Le pagine HTML sono quindi formate per composizione di contenitori (annidati) l'uno dentro l'altro

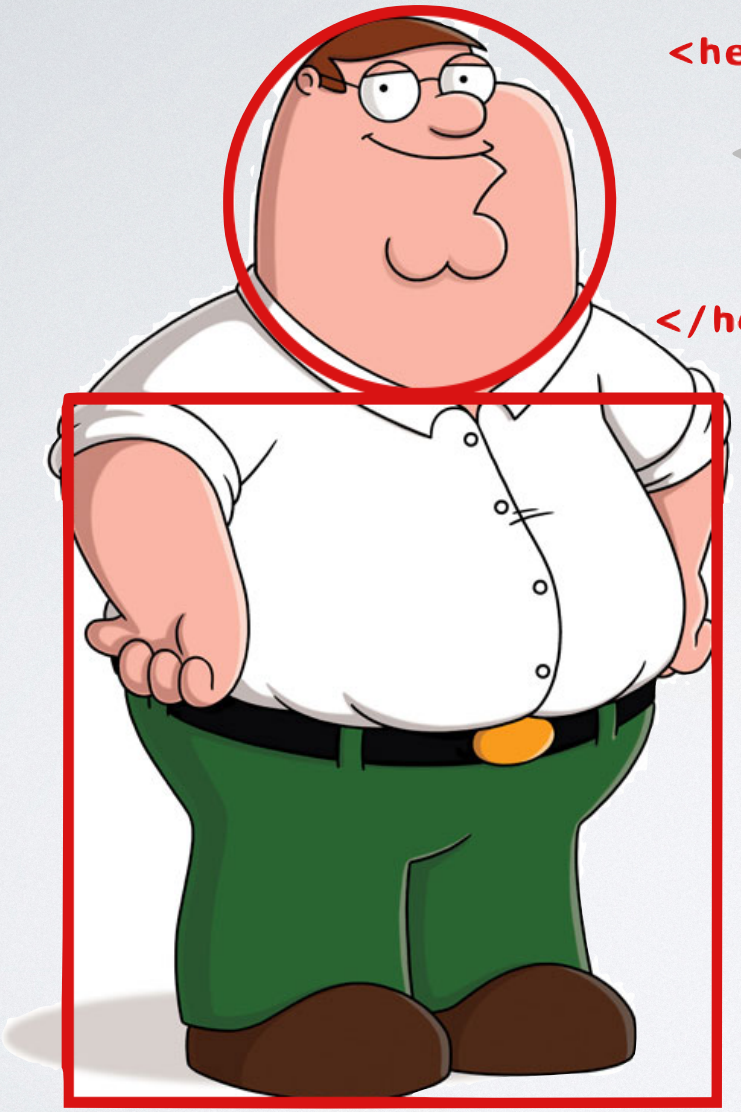
```html
<!doctype html>
<html lang="it">
    <head>
        <title>Ciao Mondo!</title>
    </head>
    <body>
        <h1>Ciao Mondo!</h1>
        <p>Questa è la nostra prima pagina HTML!</p>
        <img src="immagine.jpg">
    </body>
</html>
```

- La struttura che emerge è quella di un albero, in cui i rami sono tutti tag contenitori e le estremità sono composte da testi, immagini o altri elementi come le caselle di input. Tutti questi elementi finali non sono contenitori e non necessitano di un tag di chiusura

ESEMPIO

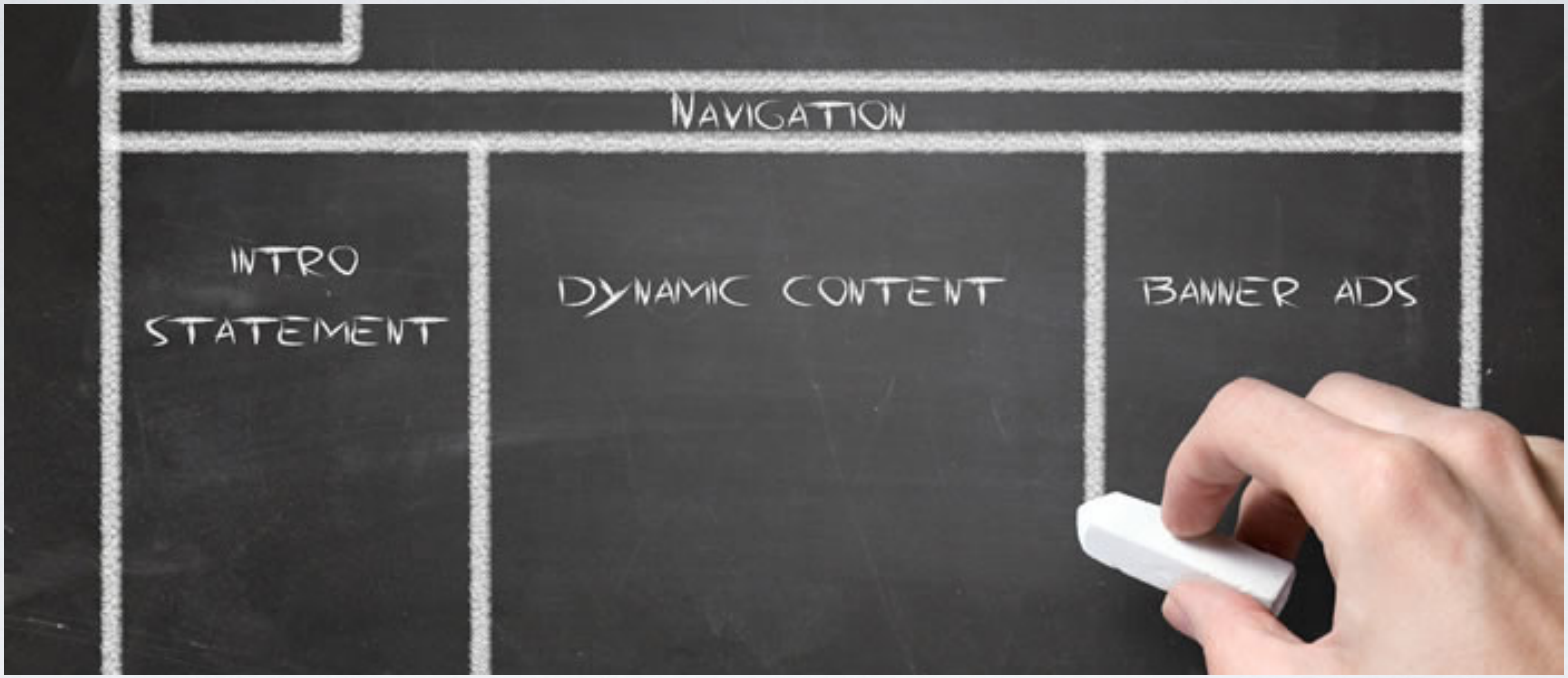
```
html
|
+---head
|   |
|   +---title
|       |
|       +---"Ciao Mondo!"
|
+---body
    |
    +---h1
        |
        +---"Ciao Mondo!"
        |
        p
        |
        +---"Questa è la nostra prima pagina HTML!"
        |
        img
```

- L'albero che abbiamo tracciato ricorda il modo in cui viene rappresentato il documento nella memoria del browser, ovvero il DOM (Document Object Model), il modello del documento HTML di cui l'elemento <html> è il nodo radice



```
<html>
    <head>
        <title>
            Peter Griffin's HTML page
        </title>
    </head>
    <body>
        Page content goes here..
    </body>
</html>
```

Dott. Antonio Giovanni Lezzi



PROGETTAZIONE

il wireframe è solo un abbozzo degli oggetti presenti nella pagina e della loro posizione al suo interno

INTERFACCIA UOMO-MACCHINA

Interfaccia uomo-macchina si riferisce allo strato che separa un essere umano che sta utilizzando una macchina dalla macchina stessa.

Un esempio è l'hardware e il software, che rende possibile ad un operatore il monitoraggio / controllo di macchine.

L'interfaccia utente comprende il flusso di informazioni per il supporto delle decisioni, attraverso:

- messaggi visivi: generalmente forniti da uno schermo o monitor
- messaggi sonori: altoparlanti, sirene, ricetrasmittenti
- azioni di controllo: tastiere, pulsanti, interruttori

INTERFACCIA UOMO-MACCHINA

La progettazione dell'abitacolo di aeroplani comprende la progettazione di interfacce utente altamente specializzate, ma anche applicazioni su larga scala come lavatrici o lavastoviglie necessitano di un'interfaccia utente.

In ogni caso in tale ambito per le interfacce uomo-macchina risultano di primaria importanza i concetti di usabilità e accessibilità per garantire un uso tipicamente user-friendly della macchina stessa a ogni tipologia di soggetto e che spesso ne decidono il relativo successo commerciale come accaduto per i personal computer.

ACCESSIBILITÀ

- L'accessibilità è la caratteristica di un dispositivo, di un servizio, di una risorsa o di un ambiente d'essere fruibile con facilità da una qualsiasi tipologia d'utente.

USABILITÀ

L'usabilità è definita come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti.

In pratica definisce il grado di facilità e soddisfazione con cui si compie l'interazione tra l'uomo e uno strumento (console, leva del cambio, interfaccia grafica, ecc.)

USABILITÀ

- Il problema dell'usabilità si pone quando il modello del progettista (ovvero le idee di questi riguardo al funzionamento del prodotto, che trasferisce sul design del prodotto stesso) non coincide con il modello dell'utente finale (ovvero l'idea che l'utente concepisce del prodotto e del suo funzionamento).

USABILITÀ: PRINCIPALI CARATTERISTICHE

- Efficacia: accuratezza e completezza con cui certi utenti possono raggiungere certi obiettivi in ambienti particolari.
- Efficienza: le risorse spese in relazione all'accuratezza e completezza degli obiettivi raggiunti.
- Soddisfazione: il comfort e l'accettabilità del sistema di lavoro per i suoi utenti e le altre persone influenzate dal suo uso.

USABILITÀ: PRINCIPALI CARATTERISTICHE

- Facilità di apprendimento: l'utente deve raggiungere buone prestazioni in tempi brevi.
- Facilità di memorizzazione: l'utente deve poter interagire con un'interfaccia anche dopo un periodo di lungo inutilizzo, senza essere costretto a ricominciare da zero.
- Sicurezza e robustezza all'errore: l'impatto dell'errore deve essere inversamente proporzionale alla probabilità d'errore.

USABILITÀ: BENEFICI

- aumenta l’efficienza
- aumenta la produttività
- riduce gli errori
- aumenta la sicurezza
- riduce il bisogno di addestramento
- riduce il bisogno di supporto all'utente
- aumenta l'accettazione
- diminuisce i costi di sviluppo

USER EXPERIENCE

- Per esperienza d'uso (UX): ciò che una persona prova quando utilizza un prodotto, sistema o servizio
- L'esperienza d'uso concerne gli aspetti esperienziali, l'attribuzione di senso e valore collegati al possesso di un prodotto e all'interazione con esso, ma include anche le percezioni personali su aspetti quali l'utilità, la semplicità d'utilizzo e l'efficienza del sistema.
- L'esperienza d'uso ha una natura soggettiva, riguarda i pensieri/sensazioni di un individuo nei confronti di un sistema
- dinamica dal momento che si modifica nel tempo al variare delle circostanze.

USER EXPERIENCE

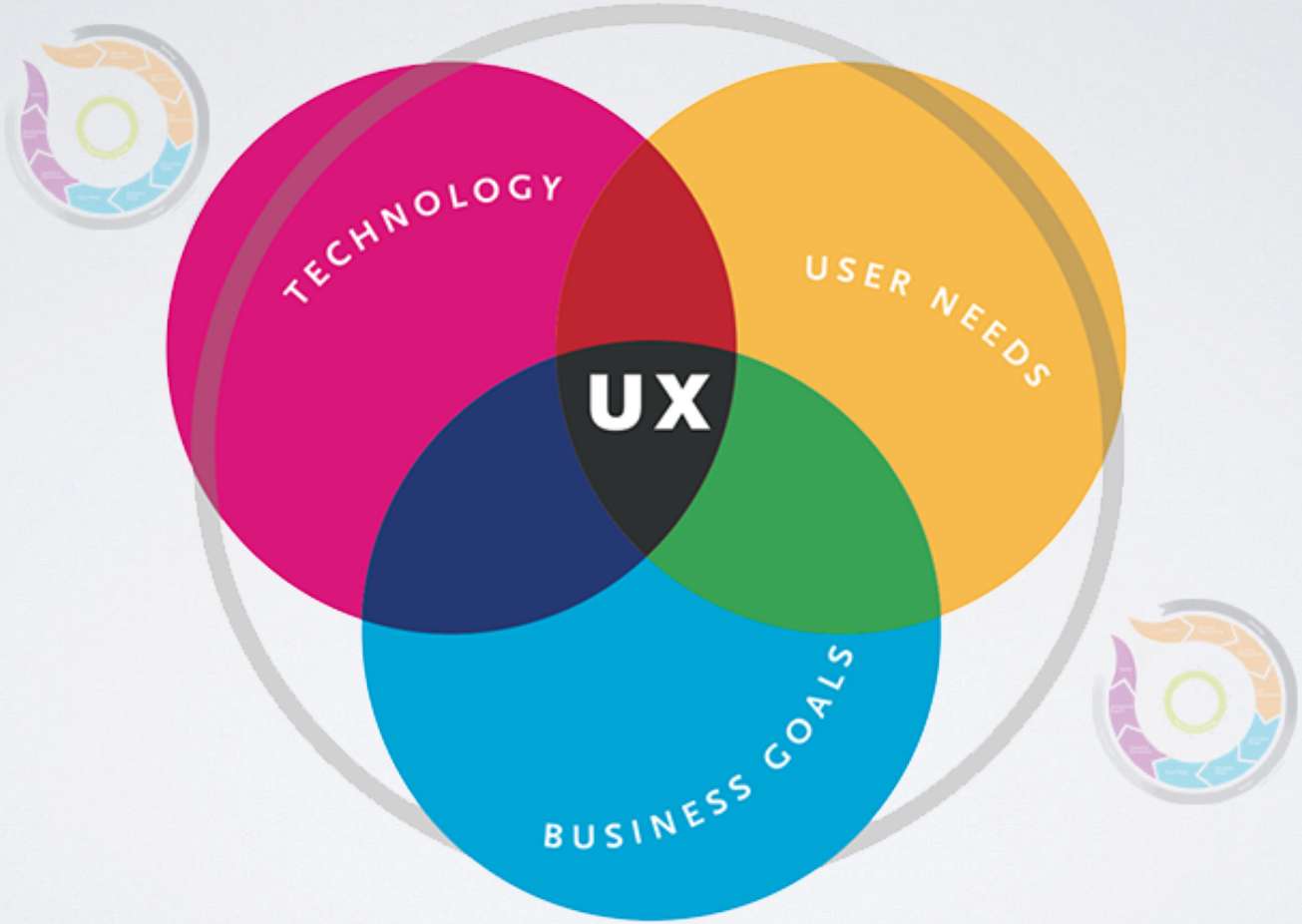
- Numerosi fattori influenzano l'esperienza che un utente vive utilizzando un sistema, suddivisi in tre grandi categorie:
- la condizione dell'utente e la sua esperienza precedente
- le caratteristiche del sistema
- il contesto di utilizzo
- Quindi, studiare il comportamento degli utenti, il loro contesto e le loro interazioni facilita la progettazione del sistema.

ANALISI

- Un principio fondamentale è capire gli utenti e i compiti che intendono svolgere.
- Le interfacce utente devono consentire tali compiti nel modo più immediato e intuitivo possibile. Perciò è importante anche la fase dell'analisi dei compiti.
- Per capire questo è importante coinvolgere nella progettazione l'utente finale: attraverso interviste, sondaggi, questionari, ecc.

PROGETTAZIONE

- La progettazione di interfacce è orientata alla comunicazione con gli utenti finali.
- Lo scopo è di selezionare gli elementi attentamente, definire soluzioni chiare, economiche, convincenti, apprendibili velocemente.
- Una progettazione efficace è in grado di ridurre gli elementi alla loro essenza, regolarizzarli in modo da favorire la loro interpretazione e combinarli in modo da sfruttarli al massimo


TECHNOLOGY
USER NEEDS
UX
BUSINESS GOALS

INTERFACCE E ADATTAMENTO

- Diventa spesso necessario che le interfacce utenti si sappiano adattare al contesto d'uso, che si può considerare sotto tre punti di vista:
- Utente, aspetti importanti sono gli obiettivi e i relativi compiti, le preferenze e il livello di conoscenza
- Dispositivo è importante considerare le modalità supportate, l'ampiezza e la risoluzione dello schermo, le capacità e velocità di connessione con altri dispositivi.
- Ambiente che ha vari aspetti che influenzano l'interazione come il livello di rumore e di luce corrente o gli oggetti che sono disponibili.

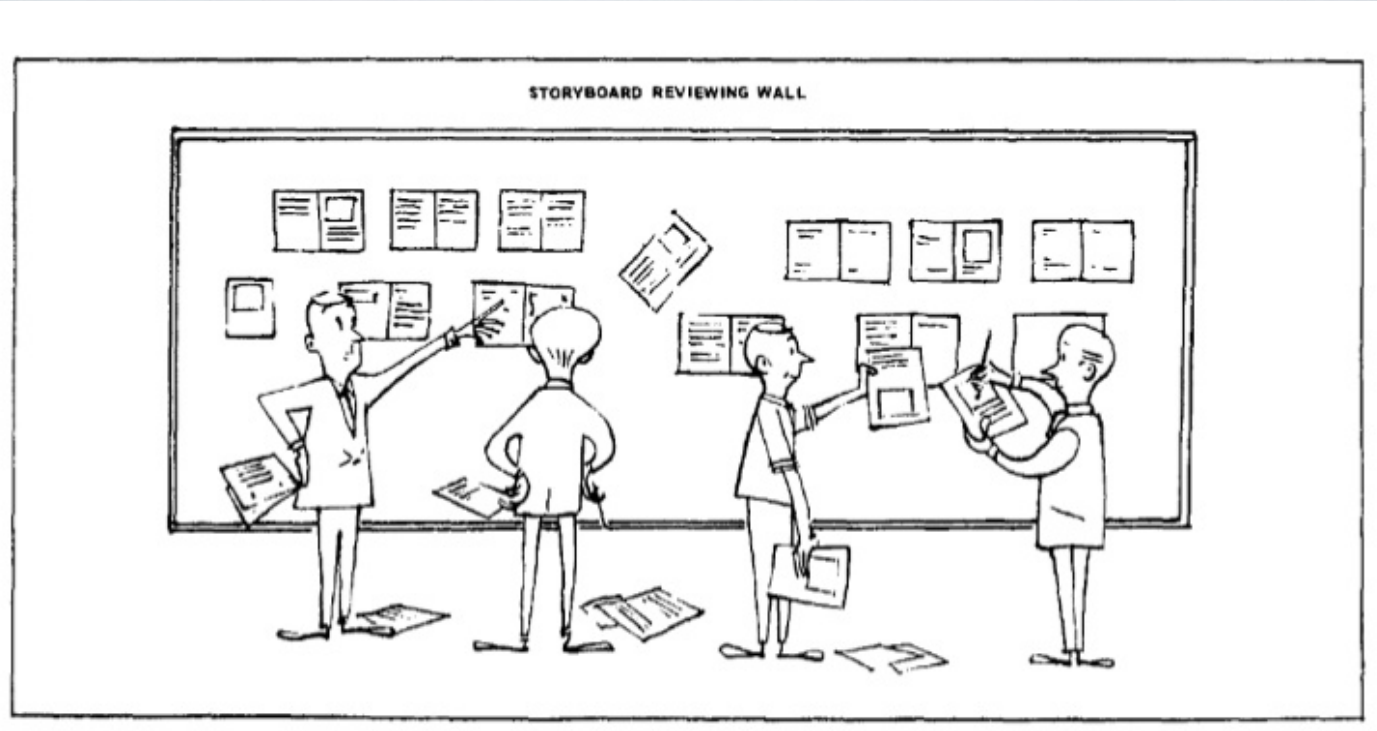
Regola 80/20

80% delle persone
usa il 20% delle funzioni.

COSA È LO STORYBOARD?

- Lo storyboard è una serie di tavolozze/pannello che indicano aspetti e scene chiave della applicazione, azioni, visualizzazioni e annotazioni che definiscono la user experience dell'utente

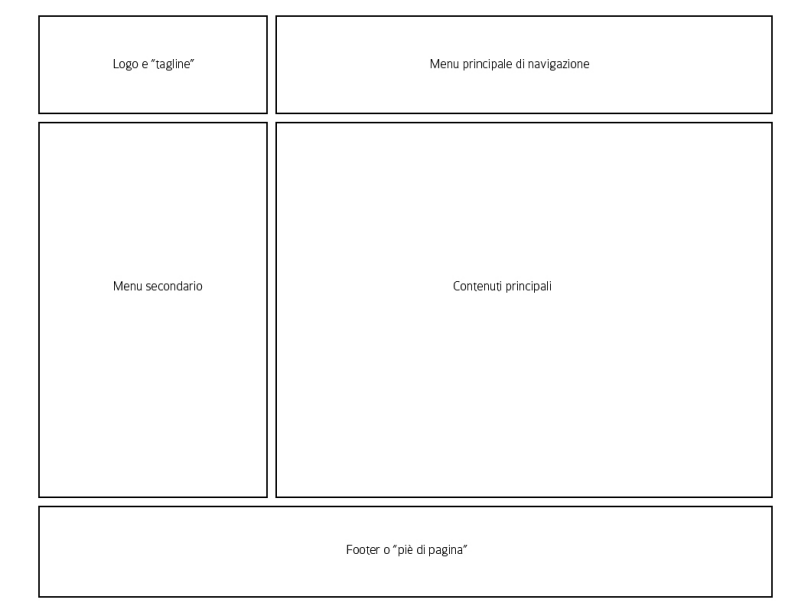
- sono organizzazioni grafiche del pannello visualizzate come illustrazioni in sequenza per poter proporre un percorso interattivo



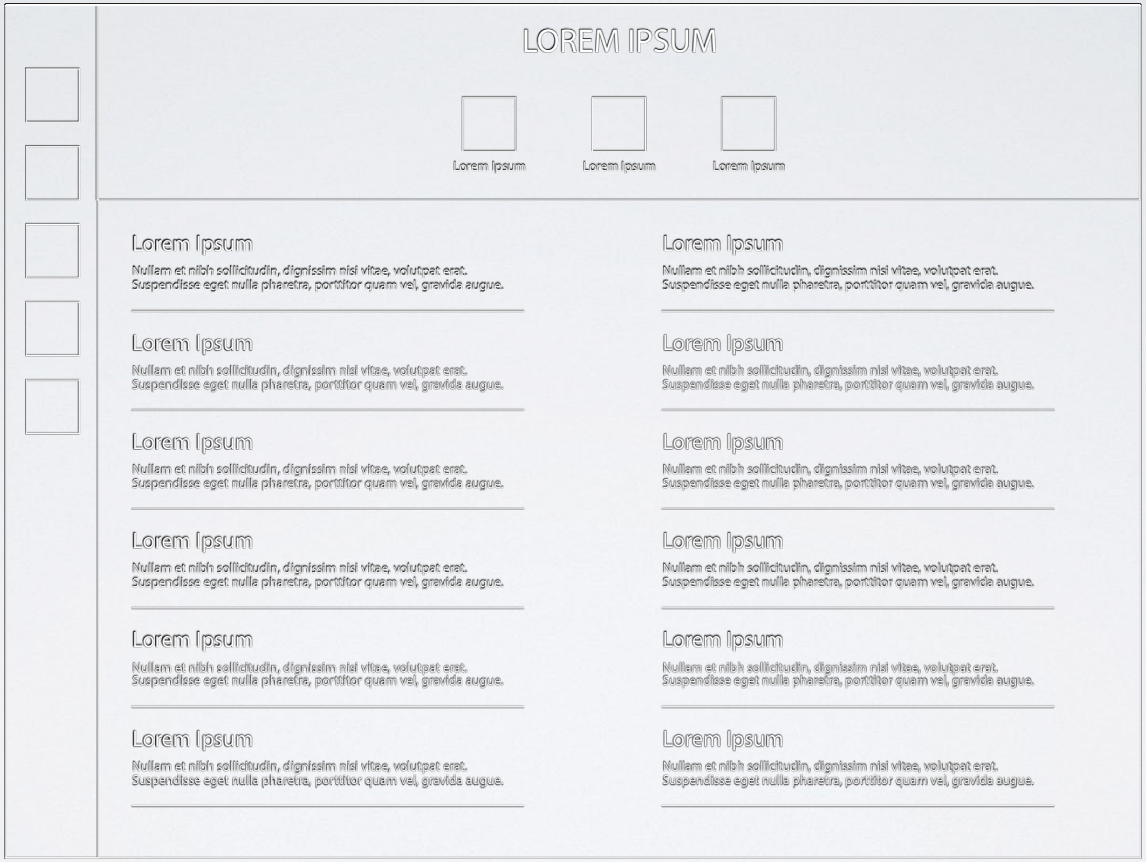
Lo Storyboard è una tecnica presa in prestito dalla industria del cinema. Ogni schermata rappresenta un pezzo di vita di un particolare evento con maggiori dettagli. Ogni pannello deve esser attribuita la stessa importanza e deve esser in grado di comunicare ai visualizzatori lo stesso concetto

WIREFRAME

- Un esempio di wireframe molto semplificato
- potete dettagliarlo anche con ulteriori "box", navigazione e specifiche importanti che vogliate definire con il cliente in questa sede per determinare le funzionalità principali, come il tipo di navigazione e di esposizione dei contenuti.



Si parte da qui...



ESERCIZI

- Analizzare la struttura dei seguenti siti visti da mobile e desktop. Valutare struttura e commentare interfaccia, usabilità, fare considerazioni oggettive (tempo caricamento, dimensione ecc)
- www.wired.it
- https://it.wikipedia.org/wiki/Pagina_principale
- https://www.microsoft.com/it-it/
- http://www.apple.com/it/

ESERCIZI

- Proporre una struttura, tramite wireframe e storyboard di un sito per:
- Pasticceria
- Azienda informatica/tecnologica
- Azienda di servizi (avvocato, commercialista)

Dott. Antonio Giovanni Lezzi

PROGRAMMA DELLA GIORNATA

- Indentazione
- Commenti
- Attributi
- Visibilità del sito
- Impostazioni pagina
- Testo, Formattazione testo, Ipertesto
- Immagini

INDENTATION

- Spesso per sottolineare l'annidamento di un elemento in un altro si usa "indentare il codice", ovvero discostare il contenuto dall'inizio della riga lasciando spazi (o tab). In pratica apertura e chiusura del tag si trovano allo stesso livello, mentre il contenuto viene spostato verso destra di un tab

- Le principali tipologie di visualizzazione in cui si dividono gli elementi sono:

- **Block (elementi blocco):** Elementi che costituiscono un blocco attorno a sé. Non fanno parte di un testo e di conseguenza mandano a capo, come i paragrafi i div o le section. In genere sono elementi contenitore nell'albero che abbiamo tracciato in alto

- **Inline (elementi "in linea"):** Elementi che non vanno a capo e possono essere integrati nel testo, come i link, le immagini, oppure gli span.

COMMENTI

- I commenti in HTML, come in altri linguaggi, permettono di aggiungere al markup una serie di annotazioni che non incidono sul rendering della pagina.
- Per essere meno criptici definiamo le funzioni più comuni svolte dai commenti in HTML:
  - Scrivere commenti/significato generiche al markup;
  - Rendere inattive porzioni di codice in fase di debug o per effettuare dei test;
  - Segnalare la chiusura di blocchi di codice per evitare confusione;
  - Anche se sempre meno, modificare il comportamento della pagina in base al browser (IE) grazie ai commenti condizionali.
- Per aggiungere un commento la sintassi è piuttosto semplice, si racchiude il testo tra <!-- e -->:

```
<!-- questo è un commento -->
```

COMMENTI CONDIZIONALI

- Aggiungiamo questa parte per completezza. In passato Internet Explorer non è stato sempre allineato agli standard HTML e CSS, questo handicap è costato caro al browser Microsoft e gli ha guadagnato la reputazione di browser più odiato, che hanno dovuto trovare una serie di escamotage per ovviare a tutte le differenze introdotte nelle diverse versioni.
- I commenti condizionali sono lo strumento più utile proprio ad adattare i siti alle passate bizzarrie di Explorer, perché consentono di riconoscere la versione di IE (es. la 6, la 7 e la 8) e di poter aggiungere del codice apposito, in genere librerie CSS o polyfiller JavaScript.
- Si tratta in definitiva di commenti con una sintassi specifica:
- <!--[if IE 7]>
  <link rel="stylesheet" type="text/CSS" href="stile-per-IE7.css">
  <![endif]-->
- riconoscibile solo da Internet Explorer (fino alla 9), per tutti gli altri browser infatti sono visti semplicemente come commenti, quindi ignorati nel rendering della pagina.

ATTRIBUTI

- I tag HTML possono essere corredati di uno o più attributi, che servono per meglio indicare la funzione o la tipologia dell'elemento, per memorizzare dati o per arricchire di significato il contenuto.
- Un tag con attributi si scrive in questo modo:
- <tag attributo1="valore1" attributo2="valore2">
- Ecco un esempio pratico:
- <input type="email" name="email" placeholder="Scrivi il tuo indirizzo email...">

ATTRIBUTI

- Il tag input indica genericamente un elemento che consente agli utenti di inserire delle informazioni. Inoltre tramite attributi possiamo indicare che vogliamo un indirizzo email (attributo type) e possiamo comunicarlo all'utente con un messaggio (attributo placeholder).
- In sostanza gli attributi:
  - sono coppie chiave-valore separate dal carattere = (uguale);
  - i valori sono tipicamente racchiusi tra virgolette "", ma è possibile anche utilizzare gli apici '';
- Nota: in alcune rare situazioni, come quando il valore dell'attributo contiene in sé le virgolette, è necessario usare gli apici (es. data-nome = 'Luca "la roccia" Rossi').

ATTRIBUTI COMUNI

- Sono attributi che impattano la rappresentazione grafica degli elementi ed alcuni aspetti dell'interfaccia utente. Eccone alcuni di quelli più legati al testo:
- **title**: Informazione che aiuta a descrivere il contenuto dell'elemento. Il testo viene spesso rappresentato in un tooltip che viene mostrato al passaggio del mouse.
- **lang**: Indica la lingua del contenuto. Es. ciao
- Altri che consentono la selezione e l'applicazione di regole di stile per la rappresentazione grafica come id, class e style sono tra i più utilizzati

ATTRIBUTI COMUNI

- **id**: Serve ad associare un identificatore univoco ad un elemento. Una volta assengato un identificatore, non si dovrà assegnarlo a nessun altro elemento della pagina.
- **class**: Serve a stabilire che l'elemento appartiene ad una serie di "classi". torneremo sull'argomento, per ora ci basti sapere che possiamo applicare a tutti gli elementi di una classe proprietà simili. Possiamo inserire quante classi vogliamo, tutte separate da uno spazio. Es. ciao , ha due classi: "saluto" e "testo-chiaro".
- **style**: Serve ad assegnare delle proprietà grafiche (Stili CSS) all'elemento, questo tipo di assegnamento degli stili viene detto stilizzazione "inline". Anche di questo parleremo in seguito
- **draggable**: Può assumere i valori true o false e serve a stabilire se un elemento sia trascinabile per una operazione di drag-n-drop.

ATTRIBUTI PER EVENTI

- Sono attributi che servono per associare determinati comportamenti dell'interfaccia utente in seguito allo scatenarsi di eventi. I comportamenti sono rappresentati da funzioni JavaScript, come in questo esempio:
- <a href="#" onclick="alert('ciao');">clicca qui</a>
- Nella pratica spesso si preferisce evitare di associare un comportamento direttamente all'interno di un tag, ma non è escluso si possano utilizzare.
- **onclick**: Rileva il click (o il tap) effettuato sull'elemento
- **onload**: Associa un comportamento alla conclusione del caricamento dell'elemento
- **onscroll**: Attiva un comportamento correlato allo scrolling della pagina
- **ondrag**: Si attiva quando iniziamo a trascinare un elemento che abbiamo indicato come draggable="true"

IMPOSTAZIONI DELLA PAGINA

- Il Tag BODY permettono di cominciare a visualizzare qualcosa sullo schermo.
- **Il tag che definisce la pagina il corpo del documento HTML è <BODY ... > e </BODY>**, dove al posto dei puntini ci possono essere dei parametri/attributi opzionali e che permettono di impostare la pagina
- Gli attributi presenti nella definizione del corpo della pagina sono quelli che definiscono la distanza del testo dai bordi della finestra, il colore e le proprietà dello sfondo della pagina, il tipo di testo utilizzato e il colore dei collegamenti ipertestuali

IMPOSTAZIONI DELLA PAGINA

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#BBBBBB" VLINK="#00FF00" LEFTMARGIN="15">
        ....
        Questo è il testo scritto all'interno della pagina, in seguito vedremo di
dargli una formattazione. Si può notare il colore dello sfondo, il margine a
sinistra e il collegamento ipertestuale
        ....
        <a href=""> Collegamento ipertestuale</a>
    </BODY>
</HTML>
```

IMPOSTAZIONI DELLA PAGINA

- <BODY BGCOLOR="#BBBBBB"> imposta lo il colore di fondo della pagina, a tinta unita, in questo caso grigio chiaro; il simbolo # indica che il colore è espresso in esadecimale come combinazione dei tre colori rosso, verde e blu, 2 numeri per colore

- Avremmo anche potuto usare la dicitura BGCOLOR=''lightgray'' ma ogni browser ha le proprie impostazioni e specificando il colore direttamente col proprio valore numerico si ottiene il colore esatto per ciascun visualizzatore. Volendo usare uno sfondo blu si imposta BGCOLOR="#0000FF"

IMPOSTAZIONI DELLA PAGINA

- <BODY BACKGROUND="immagini/sfondo1.jpg"> imposta lo sfondo della pagina come una serie di immagini "sfondo1.jpg" affiancate in modo ripetuto
- L'immagine di sfondo è un file, residente eventualmente in un'altra cartella; essendo un file, che viene trasferito in rete insieme alla pagina, sarebbe bene scegliere un file piuttosto leggero in modo che venga trasferito prima (una pagina caricata velocemente è apprezzata di più)

IMPOSTAZIONI DELLA PAGINA

- La presenza del parametro BACKGROUND non annulla il parametro BGCOLOR, ma l'immagine viene posizionata sopra al fondo a tinta unita, quindi è inutile che siano presenti entrambi nella definizione del body; in caso di rete particolarmente lenta o di immagini pesanti da caricarsi verrà mostrato lo sfondo colorato prima di visualizzare definitivamente l'immagine.

- <BODY BACKGROUND="immagini/sfondo1.jpg" BGPROPERTIES="fixed">
  imposta lo sfondo della pagina come una immagine particolare e lo fissa alla finestra e non lo fa scorrere insieme al testo su e giù per la pagina; con questo parametro sarà solo il testo a scorrere all'interno della finestra, sopra allo sfondo. Il parametro viene ignorato da alcuni browser

IMPOSTAZIONI DELLA PAGINA

- <BODY TOPMARGIN="40"> indica che la pagina ha un margine di (ad esempio) 40 pixel dal bordo superiore della finestra a lei dedicata; è come se tutto il contenuto della pagina fosse spostato 40 punti più in basso. Usato da solo (senza il corrispettivo BOTTOMMARGIN) questo parametro pone una striscia vuota sia all'inizio che alla fine della pagina

- <BODY BOTTOMMARGIN="40"> indica che la pagina ha un margine di (ad esempio) 40 pixel dal bordo inferiore della finestra; è come se alla pagina seguisse una ulteriore striscia vuota alta 40 punti. Il parametro funziona solo con Internet Explorer e viene ignorato dal browser Netscape, su questo browser il parametro usato è MARGINHEIGHT

IMPOSTAZIONI DELLA PAGINA

- <BODY LEFTMARGIN="30"> indica che la pagina ha un margine di (ad esempio) 30 punti dal bordo sinistro della finestra; l'area effettivamente utilizzata per la pagina è minore, ma può tornare utile per dare una diversa impostazione grafica al proprio lavoro. Usato da solo (senza il corrispettivo RIGHTMARGIN) questo parametro pone una striscia vuota sia a sinistra che a destra della pagina. Il parametro funziona solo con Internet Explorer e viene ignorato dal browser Netscape, su questo browser il parametro usato è MARGINWIDTH (vedi più in basso)

- <BODY RIGHTMARGIN="30"> indica che la pagina ha un margine di (ad esempio) 30 punti dal bordo destro della finestra. Il parametro funziona solo con Internet Explorer e viene ignorato dal browser Netscape, su questo browser il parametro usato è MARGINWIDTH (vedi più in basso)

- <BODY MARGINHEIGHT="30" MARGINWIDTH="40"> si usano per impostare le semi-pagine dei frame e indicano il margine della pagina dai bordi dei frame, rispettivamente bordo superiore/inferiore e bordo sinistro/destro.

IMPOSTAZIONI DELLA PAGINA

- <BODY TEXT="#000080"> specifica il colore predefinito del testo all'interno della pagina (in questo esempio, blu scuro); se nella pagina sarà presente del testo privo di formattazione, ad esso sarà dato il colore definito

- <BODY LINK="#0000FF"> specifica il colore predefinito dei collegamenti ipertestuali (vedremo in seguito cosa sono); generalmente sono di colore azzurro ("#0000FF") oppure personalizzati dall'utente mediante le impostazioni del browser

- <BODY VLINK="#FF00FF"> specifica il colore predefinito dei collegamenti ipertestuali già visitati; generalmente sono di colore viola ("#FF00FF") oppure personalizzati dall'utente mediante le impostazioni del browser.

IMPOSTAZIONI DELLA PAGINA

- <BODY ALINK="#FF0000"> specifica il colore predefinito dei collegamenti ipertestuali appena cliccati ma la cui pagina non è ancora stata caricata; generalmente sono di colore rosso ("#FF0000") oppure personalizzati dall'utente mediante le impostazioni del browser.
- <BODY ONLOAD=getimages();> specifica (in questo caso) una funzione da implementare quando viene caricata la pagina
- Con i comandi e i parametri ora introdotti non riusciremo a visualizzare un granché sullo schermo, però già abbiamo dato alla pagina uno sfondo e abbiamo definito i principali parametri con in quali la pagina sarà visualizzata.

TESTO

- **Il contenuto di un sito web è essenziale per una sua buona riuscita**
- Un testo lungo e articolato, sebbene interessante, perde interesse se mostrato senza una degna struttura e impaginazione; l'uso del colore, di tabulazioni, di indentazioni, etc..., rende più leggibile anche il testo più complicato
- È per questo che prima di definire quali caratteri usare, di che dimensione e colore descriviamo come impaginare e posizionare il testo all'interno di una pagina Web.
- L'HTML dispone di TAG che consentono di costruire un layout (visualizzazione) abbastanza preciso del testo, permettendoci di disporlo come lo si vuole, utilizzando titoli, paragrafi, separatori e interlinee.

TESTO

- **Il tag <Hn> definisce un titolo. Il simbolo n viene sostituito da un numero da 1 a 6 che indica la dimensione del titolo; <H1> è il più grande e <H6> il più piccolo**

- Il tag <Hn> ha l'attributo ALIGN, opzionale, che permette di dare l'allineamento al titolo; i valori possibili sono "left", "center", "right" e "justify", ma quest'ultimo ha poco senso, parlando di un titolo

- L'attributo predefinito è l'allineamento a sinistra, "left", e può omettersi; la sintassi corretta, per un titolo di grandezza 2 allineato al centro, è <H2 ALIGN="center">

TESTO

- **Il tag <DIV> è utilizzato per raggruppare vari elementi isolati (testo, immagini e così via) in una singola unità**, alla quale si vuole applicare la stessa formattazione (di solito l'allineamento); esso riconosce l'attributo ALIGN, con gli stessi valori del tag <Hn>; riconosce anche l'attributo ID="nome" utile nei fogli di stile CSS, insieme al tag <STYLE> che vedremo più avanti, per identificare l'oggetto all'interno della pagina.

- **Il tag <P> ... <P>indica un paragrafo (inizio e fine)**; ad un paragrafo viene sempre anteposta una interlinea di una riga e mezza e ad esso segue un'altra interlinea di una riga e mezza. Due paragrafi successivi appaiono separati da una sottile striscia vuota. Esattamente come per i tag precedenti riconosce l'attributo ALIGN, con gli stessi valori, il valore predefinito è ALIGN="left" e può omettersi. *Il tag <P> può essere usato anche da solo (senza il tag di chiusura </P>), purché si inizi il paragrafo successivo con un altro <P> altrimenti il browser suppone di stare ancora nel paragrafo corrente*

TESTO

- **Il tag (o meglio il comando di markup)
 equivale esattamente a un ritorno a capo**.il testo interponendo una interlinea singola.
- **Il tag <DIR> inserisce una tabulazione**. Le tabulazioni in HTML hanno un valore fisso (in genere una 80-ina di pixel a seconda del browser) e non possono essere modificate. Le tabulazioni possono essere inserite una dentro l'altra per ottenere una tabulazione doppia, tripla, etc... Vedere come esempio il titolo e l'argomento del presente capitolo, in alto nella pagina.
- **Il tag <BLOCKQUOTE> è identico al tag <DIR> e inserisce una tabulazione, dello stesso valore**. Le tabulazioni possono essere inserite una dentro l'altra per ottenere una tabulazione doppia, tripla, etc...

TESTO

- **Il tag <HR> inserisce una riga orizzontale (di solito con una ombreggiatura) di separazione tra due paragrafi consecutivi**; non c'è il tag di chiusura e riconosce 4 attributi: ALIGN che definisce l'allineamento della riga con gli stessi valori dei tag precedenti (il valore predefinito è ALIGN="center"); WIDTH indica la lunghezza in pixel della linea, se questo numero è seguito dal simbolo % allora si tratta di una percentuale della larghezza della colonna; SIZE indica lo spessore in pixel della linea; NOSHADE (senza parametri) elimina l'ombreggiatura

- Ad esempio, il comando <HR SIZE="2" WIDTH="70%" NOSHADE> visualizza la linea mostrata di seguito

TESTO

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
        ....
    </HEAD>
    <BODY>
        <H2>Titolo della pagina</H2>
        <P ALIGN="justify">Questo è un testo giustificato (notare l'allineamento su entrambi i lati) scritto
in un paragrafo. Quando la pagina termina la frase viene troncata e la riga allineata. Notare che se anche si
va a capo nel testo HTML, nella
        pagina non si va a capo, perché il browser non trova alcun comando per farlo. Stesso discorso se si
lasciano molti spazi tra una parola e l'altra, il browser li ignora e visualizza uno spazio solo.<BR>
        Qui va a capo senza lasciare una interlinea. Qui sotto vediamo una linea allineata a destra, molto
spessa.</P>

        <HR ALIGN="right" SIZE="8" WIDTH="50%">
        <DIR>Questo è invece un testo allineato a sinistra, a cui è stato posto una tabulazione; il testo
risulta spostato più a destra.</DIR>
        ....
    </BODY>
</HTML>
```

TESTO

Titolo della pagina

Questo è un testo giustificato (notare l'allineamento su entrambi i lati) scritto in un paragrafo. Quando la pagina termina la frase viene troncata e la riga allineata. Notare che se anche si va a capo nel testo HTML, nella pagina non si va a capo, perché il browser non trova alcun comando per farlo. Stesso discorso se si lasciano molti spazi tra una parola e l'altra, il browser li ignora e visualizza uno spazio solo.
Qui va a capo senza lasciare una interlinea. Qui sotto vediamo una linea allineata a destra, molto spessa.

Questo è invece un testo allineato a sinistra, a cui è stato posto una tabulazione; il testo risulta spostato più a destra.

....

VISIBILITÀ DI UN SITO

- Gli utenti in internet effettuano ogni giorno milioni di ricerche e il gioco sta nel "farsi trovare" facendo visitare il proprio sito e non quello della concorrenza
- È una questione di ascesa verso la popolarità del sito o di declino nell'anonimato dove "nessuno ti conosce e nessuno ti visita"
- Ho visto più volte dei progetti validissimi fallire solamente perché non riuscivano a diventare popolari, la chiave di tutto era il loro posizionamento nei motori di ricerca

VISIBILITÀ DI UN SITO

- La maggior parte del traffico che arriva dalla rete proviene dai motori di ricerca. Se un sito non è visibile allora è come se non esistesse e nessun motore di ricerca lo prende in considerazione

- I dati relativi ad una ricerca condotta dall'azienda Nielsen Media **"L'importanza del posizionamento nei motori di ricerca"**. Come si arriva ad un sito web:

70%	Motori di ricerca e Directory
12%	Link da altro sito
10%	Link dalle email
5%	Passaparola
3%	Guide cartacee, pubblicità
2%	Casualmente o altro…

CRAWLER

- Un crawler (detto anche web crawler, spider o robot), è un software che analizza i contenuti di una rete (o di un database) in un modo metodico e automatizzato, in genere per conto di un motore di ricerca
- Un crawler è un tipo di bot (programma o script che automatizza delle operazioni)
- I crawler solitamente acquisiscono una copia testuale di tutti i documenti visitati e le inseriscono in un indice
- Sul Web, il crawler si basa su una lista di URL da visitare fornita dal motore di ricerca (il quale, inizialmente, si basa sugli indirizzi suggeriti dagli utenti o su una lista precompilata dai programmatori stessi)

CRAWLER

- Durante l'analisi di un URL, identifica tutti gli hyperlink presenti nel documento e li aggiunge alla lista di URL da visitare. Il processo può essere concluso manualmente o dopo che un determinato numero di collegamenti è stato seguito.
- Inoltre i crawler attivi su Internet hanno la facoltà di essere indirizzati da quanto indicato nel file "robots.txt" posto nella root del sito. All'interno di questo file, è possibile indicare quali pagine non dovrebbero essere analizzate
- Il crawler ha la facoltà di seguire i consigli, ma non l'obbligo.

META TAG

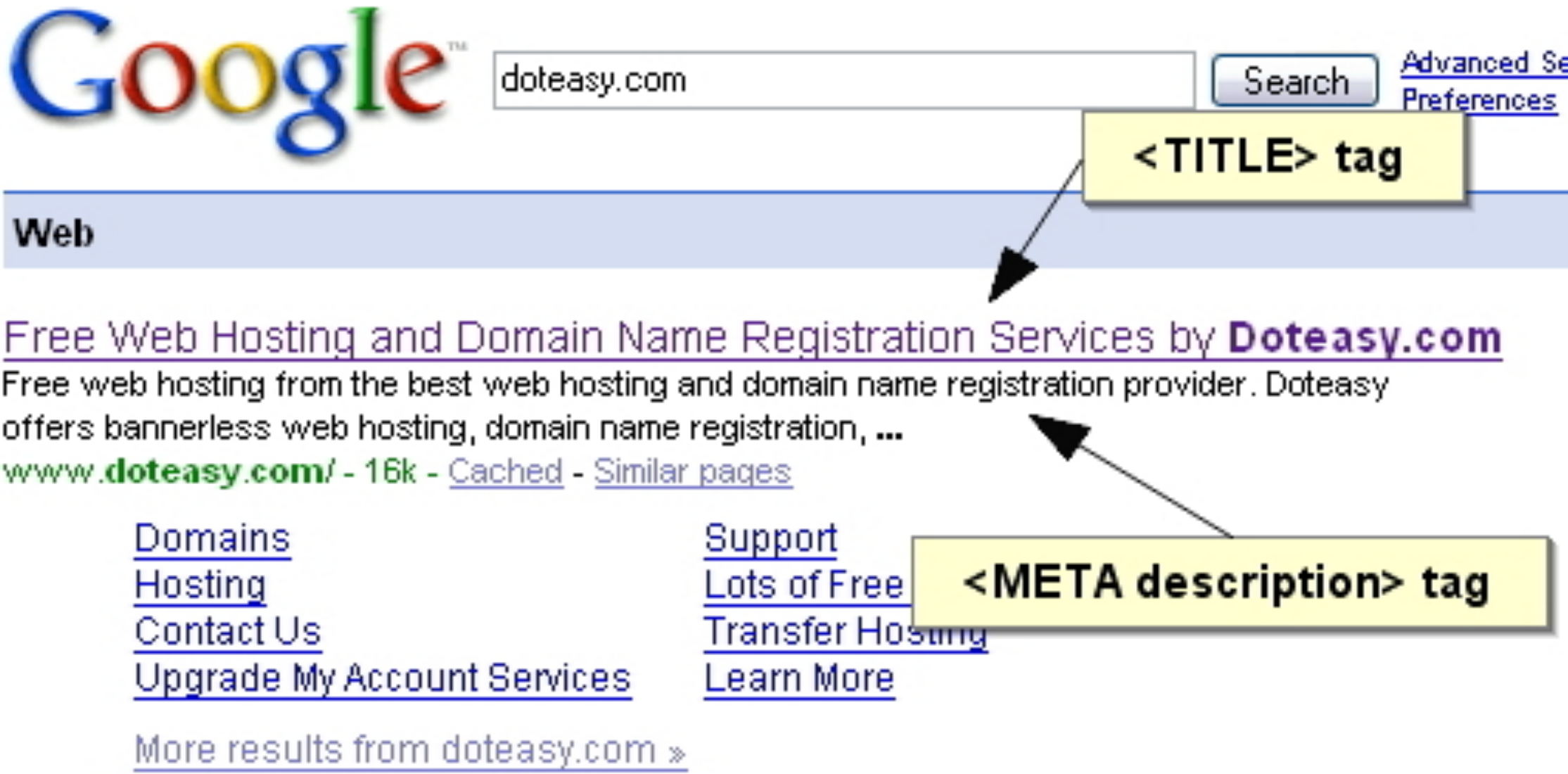
- **I meta-tag sono pseudo-comandi** che non hanno un'azione diretta nei confronti della pagina, ma **hanno il fine di migliorare i rapporti con i programmi con cui la pagina si relaziona, come il browser, il server web, i motori di ricerca ...**
- I meta-tag sono identificati dal tag <META ...> e non hanno il tag correlato </...>; sono infatti dei veicoli di informazioni, e le informazioni sono portate da ciascun comando <META …>
- al posto dei puntini vanno inseriti gli attributi dei meta-tag, che possono essere HTTP-EQUIV, NAME e CONTENT
- <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252''> indica che il contenuto del documento è di tipo html e usa quel particolare set di caratteri.
- <META HTTP-EQUIV="Set-Cookie" CONTENT="ID=EiG4HtfyyUIyf; path=/''> specifica al server web in quale cartella disporre un eventuale cookie, quei file che vengono posti sul computer del visitatore e che servono a monitorare la navigazione e a memorizzare elementi

META TAG

- <META HTTP-EQUIV="Content-Language" CONTENT="It"> indica che la pagina è in italiano (utile ai motori di ricerca)
- <META HTTP-EQUIV="Refresh" CONTENT="25; URL='http://www.canoro.org'"> serve per dare un certo dinamismo che permette di caricare automaticamente una ulteriore pagina dopo che questa pagina è stata caricata. Il numero 25 indica i secondi che devono passare prima di caricare la nuova pagina. URL="http://..." specifica la nuova pagina da caricare. Utile per reindirizzare automaticamente il visitatore da una pagina ad un'altra
- <META NAME="Generator" CONTENT="FrontPage"> indica il programma utilizzato per creare il documento
- <META NAME="Description" CONTENT="Questo sito parla di una guida di HTML"> fornisce una descrizione del documento (utile ai motori di ricerca).

META TAG

- <META NAME="Keywords" CONTENT="manuale di html, html, imparare l'html …"> contiene un elenco di parole chiave utili ai motori di ricerca e ad eventuali utenti per cercare il documento su internet.
- <META NAME="Author" CONTENT="‘Antonio Lezzi"> specifica l'autore del documento.
- <META NAME="Reply-To" CONTENT="‘antoniolezzi@dominio.it"> indica la email dell'autore, alla quale si vuole essere contattati per chiarimenti
- Esistono moltissimi altri parametri da attribuire ai meta-tag, di più o meno vaga utilità


Google
doteasy.com
Search
Advanced Se
Preferences
<TITLE> tag
Web
Free Web Hosting and Domain Name Registration Services by Doteasy.com
Free web hosting from the best web hosting and domain name registration provider. Doteasy offers bannerless web hosting, domain name registration, ...
www.doteasy.com/ - 16k - Cached - Similar pages
Domains
Hosting
Contact Us
Upgrade My Account Services
Support
Lots of Free
Transfer Hosting
Learn More
<META description> tag
More results from doteasy.com »

META TAG

```
<HTML>
    <HEAD>
        <TITLE>Titolo del documento</TITLE>
        <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
        <META NAME="Keywords" CONTENT="manuale di html, html, imparare l'html ...">
        <META NAME="Author" CONTENT="“Antonio Lezzi">
        ....
    </HEAD>
    <BODY>
        ....
        ....
    </BODY>
</HTML>
```

FATTORI DI INDICIZZAZIONE

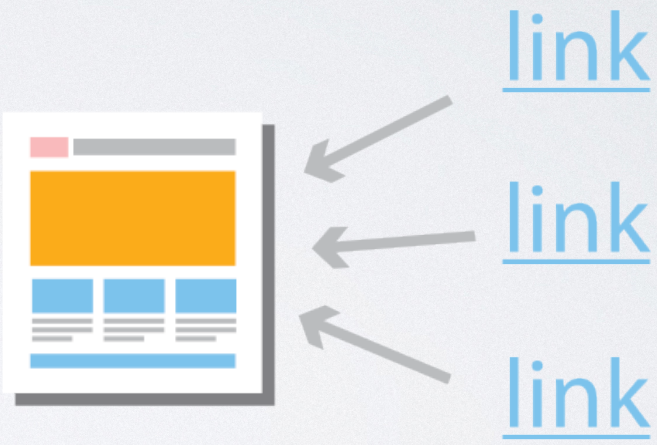
- Google non si concentra solo sul dominio ma dà molta importanza ai contenuti anche se posso assicurarti che nella maggior parte dei casi, siti anche non eccellenti con la parola chiave nel dominio corretta, vengono spesso trovati in prima pagina

- **Estensione e keyword giusta del dominio non bastano**
  Ci sono altri fattori importanti per l'indicizzazione di Google

- Se hai trovato l'estensione che ritieni più coerente con gli argomenti trattati nel tuo sito ed una corretta keyword (parola chiave) per il tuo dominio devi sapere che questo non basta soprattutto nel lungo periodo

FATTORI DI INDICIZZAZIONE

- Gli altri fattori per una buona indicizzazione su google sono:
- **Fattori Esterni principali:**
    1. Storicità/Età del tuo sito
    2. Numero di click ricevuti
    3. Link popularity Pagerank/Trustrank (non ufficiale)
- **Fattori Interni principali:**
    1. Home Page
    2. Title
    3. Metatag
    4. Piè di pagina
    5. Link all' interno delle pagine



FATTORI DI INDICIZZAZIONE

- Msn e Yahoo seguono meno regole rispetto a Google: questo aspetto non si rivela sempre un vantaggio perché rende questi due motori paradossalmente più rigidi
- Non si possono sottovalutare eventuali differenze perché essere visibile anche su di loro è di fondamentale importanza per la popolarità di un sito e il numero di visite ricevute

FORMATTAZIONE TESTO

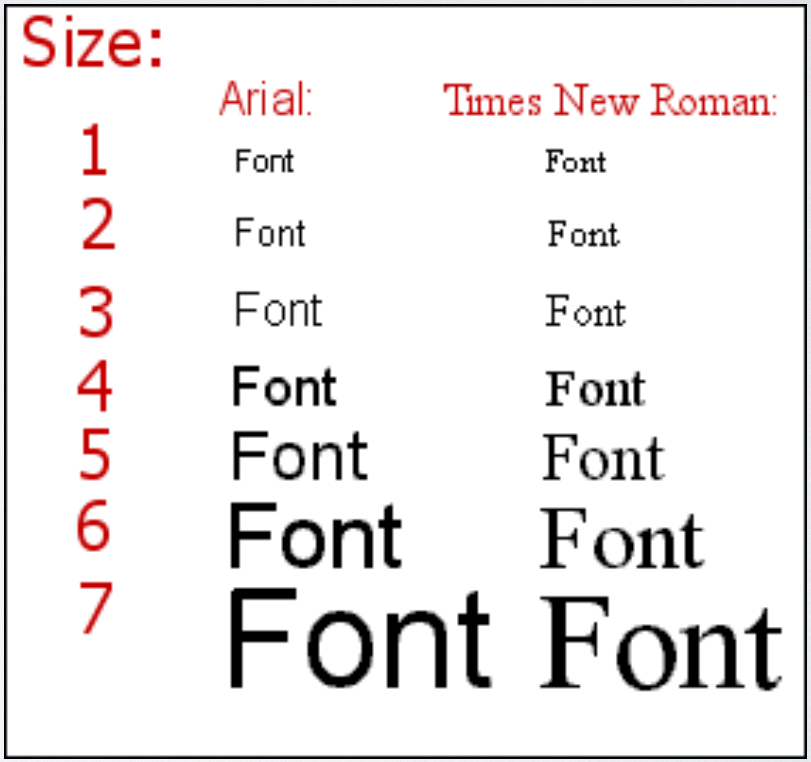
- **Per scrivere in grassetto si usa il tag contenitore <B>...</B>** (oppure si può anche usare <STRONG>, è "lo stesso"); la lettera B è l'iniziale della parola bold, grassetto appunto
- **Per scrivere in corsivo si usa il tag contenitore <I>...</I>** (oppure si può anche usare <EM>); è l'iniziale della parola italic. Tag equivalenti, anche se nati con un significato logico diverso, sono <CITE> oppure <ADDRESS>
- **Per scrivere in sottolineato si usa il tag contenitore <U> e </U>**; tutto il testo compreso tra i due tag verrà visualizzato come sottolineato
- È possibile aumentare o diminuire la dimensione dei caratteri da visualizzare; i tag <BIG> e l'altro <SMALL> rispettivamente consentono di aumentare e diminuire di un punto le dimensioni del carattere

FONT

- **il tag <FONT>...</FONT> consente di fornire una maggiore formattazione al testo**

- L'attributo FACE="..." indica il tipo di carattere da utilizzare, se il computer su cui si sta guardando la pagina web non ha installati i tipi di carattere in questione prova ad utilizzarne uno simile o quelli che ha come predefiniti (in genere il "Times New Roman'')

- La dimensione del carattere può essere specificata con l'attributo SIZE="n", in cui n rappresenta un valore assoluto (da 1 a 7) oppure un valore relativo (da -2 a +4) in relazione alla dimensione corrente del carattere. SIZE="1" indica il carattere più piccolo



FONT

- L'attributo COLOR="..." determina il colore del carattere; al posto dei puntini va il nome o il numero del colore (espresso come stringa esadecimale nelle combinazioni di rosso, verde e azzurro)

- Ad esempio il colore rosa pallido può essere indicato sia con COLOR="RosyBrown" che con COLOR="#BC8F8F"

FORMATTAZIONE TESTO

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFEBCD">
        <P ALIGN="center">
        <FONT SIZE="5" COLOR="#006699"><B>Le fasi lunari</B> </FONT></P>
        <FONT FACE="Comic Sans MS">Mostriamo ora come con un testo preformattato si possa ottenere una
tabella. Lasciamo il <I>Courier</I> come carattere, avremmo anche potuto cambiarlo.</FONT>
        <PRE>
        Mese | Giugno | Luglio | Agosto |
        Fase| | | | -------|--------|--------|--------|
        Nuova | 5 | 4 | 3 |
        Primo | 13 | 12 | 11 |
        Piena | 20 | 20 | 18 |
        Ultimo| 27 | 26 | 25 |
        </PRE>
        <SAMP>Questo è invece un testo scritto con carattere monospazio (<U>qui è sottolineato</U>), ma non è
preformattato, infatti si può notare come vada a capo in modo diverso nel documento e nella pagina.</SAMP>
    </BODY>
</HTML>
```

FORMATTAZIONE TESTO

Le fasi lunari

Mostriamo ora come con un testo preformattato si possa ottenere una tabella. Lasciamo il *Courier* come carattere, avremmo anche potuto cambiarlo.

```
Mese    | Giugno | Luglio | Agosto |
Fase    |        |        |        |
-------|--------|--------|--------|
Nuova   |    5   |    4   |    3   |
Primo   |   13   |   12   |   11   |
Piena   |   20   |   20   |   18   |
Ultimo  |   27   |   26   |   25   |
```

Questo è invece un testo scritto con carattere monospazio (qui è sottolineato), ma non è preformattato, infatti si può notare come vada a capo in modo diverso nel documento e nella pagina.

IMMAGINI

- Molto probabilmente il WWW (World Wide Web, ossia il Web a livello mondiale) non avrebbe avuto lo sviluppo che lo ha visto protagonista in questi ultimi 15 anni se non ci fossero state le immagini, se non fosse stato possibile introdurre elementi di grafica nelle pagine che si presentavano

- Le immagini, che possono essere fotografie, disegni o anche effetti colorati, contribuiscono a dare un tocco di vitalità alla pagina e, soprattutto, ad attirare maggiormente l'occhio del visitatore.

- Esistono diversi formati di immagine, decine; l'HTML ne riconosce ufficialmente solo: GIF e JPEG. Un terzo formato, il PNG, potrebbe lentamente sostituire le GIF per ragioni che riguardano i diritti di autore, ma fino ad oggi è stato implementato solo in parte

- Di solito è conveniente usare il formato JPEG per le fotografie, ricche di dettagli, e riservare il formato GIF per i disegni e i diagrammi a schema realizzati con appositi programmi di disegno

IMMAGINI

- I browser attuali sono molto robusti e permettono di visualizzare un numero elevato di formati grafici ma è bene non allontanarsi troppo dagli standard per questioni di compatibilità e di spazio occupato

- Oltre ai programmi di disegno e alla propria macchina fotografica digitale o il proprio scanner, le immagini possono essere reperite sul Web o nei siti FTP, prestando la dovuta attenzione all'utilizzo di materiale tutelato da copyright

IMMAGINI

- **Il tag necessario per l'inserimento di una immagine è <IMG>** che non ha tag di chiusura, una volta inserita l'immagine non ha senso dire al browser dove l'immagine finisce
- Il tag <IMG> scritto da solo non ha però alcun effetto sulla pagina; esso riconosce alcuni attributi, di cui uno è obbligatorio: occorre specificare qual è l'immagine da inserire
- **L'attributo SRC="..."** indica il percorso dell'immagine da caricare
- può essere assoluto, e quindi essere del tipo SRC="http://www.sito.it/cartella/immagine.gif"
- oppure, se l'immagine si trova ospitata sullo stesso sito, può essere relativo, del tipo SRC="cartella/immagine.gif"
- Una precisazione importante riguardo ai nomi delle immagini e in generale di qualunque file. I server di tipo Unix/Linux fanno distinzione tra nomi MAIUSCOLI e minuscoli, è bene fare attenzione sul tipo di carattere che si usa, e nel dubbio preferire l'uso del carattere minuscolo
- **l'importante è specificare il percorso corretto**

IMMAGINI

- Quindi ci sono gli attributi opzionali, ma che servono se si vuole dare una impostazione diversa all'immagine introdotta
- Se si carica la pagina web su un browser in cui lo scaricamento delle immagini è stato disattivato (per ragioni di velocità dei collegamenti, ma la pagina sembrerà più spoglia) è possibile non privare gli utenti di tutte le informazioni
- **l'attributo ALT="..." permette di inserire un testo alternativo al posto dell'immagine**. Questo testo viene visualizzato quando si passa sopra l'immagine con il mouse, quando l'immagine manca oppure quando si utilizza un browser testuale.
- **Gli attributi WIDTH="x" e HEIGHT="y" determinano le dimensioni della immagine, in larghezza e in altezza**. I numeri x e y possono essere sia numeri assoluti ed in tal caso indicano il numero di pixel che misura l'immagine, oppure una percentuale della dimensione della finestra
- Se questi attributi sono assenti l'immagine viene mostrata alla sua grandezza naturale
- Se è presente uno solo dei due attributi, l'altro si adegua in proporzione

IMMAGINI

- Su alcuni browser è implementato **l'attributo STYLE="..."** che racchiude questi due attributi in uno solo, **la sua sintassi è: STYLE="height:x; width:y"**

- **L'attributo BORDER="x" indica lo spessore in pixel della cornice che si vuole mettere attorno all'immagine**; il colore del bordo è nero; se è assente, l'immagine viene visualizzata senza bordo aggiuntivo

IMMAGINI

- L'attributo ALIGN="..." è qui più complesso che negli altri tag in cui compariva. Può avere i seguenti valori: "left", "right", "bottom", "top" e "middle" (oppure "absmiddle"); ce ne sono altri, ma hanno lo stesso significato di questi ultimi tre
- I valori "left" e "right" indicano che l'immagine è messa alla sinistra o alla destra di ciò che segue, se c'è un testo, e, se l'immagine fosse piuttosto grande, questo verrebbe disposto su più righe accanto all'immagine
- I valori "bottom", "top" e "middle" specificano che l'immagine viene visualizzata nella riga in cui è posizionata e il testo è allineato in una sola riga (anche se l'immagine è grande) al bordo inferiore o superiore o al centro dell'immagine. Se questo attributo è assente, il valore predefinito è "bottom".
- Infine gli attributi HSPACE="x" e VSPACE="y" indicano quanti pixel di spazio vuoto lasciare attorno alla immagine, rispettivamente a sinistra/destra e sopra/sotto, come una cornice vuota

IMMAGINI

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE> </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <P ALIGN="justify">
        <IMG SRC="nonna.jpg" ALIGN="left" WIDTH="60">
            Qui l'immagine è stata messa a sinistra e il testo alla sua destra (con allineamento giustificato) su più righe.
        </P>
        <P ALIGN="justify">Il testo continua al lato dell'immagine nonostante il segno di paragrafo.</P>
        Qui invece si è inserita <IMG SRC="haha.gif" ALIGN="absmiddle"> una immagine dentro il testo, e siccome era piccola si è
potuta inserirla centrata con il testo, in modo da trovare una cosa accettabile graficamente.
        <P ALIGN="justify">
            <IMG SRC="manca.jpg" WIDTH="122" HEIGHT="50" ALT="Immagine mancante, compare un testo"><BR>
                Qui sopra l'immagine non viene trovata, oppure non viene mostrata, e al suo posto compare il testo alternativo.
Notare come il testo viene troncato se non entra nello spazio. Con Netscape il testo compare solo quando si passa col mouse sopra
l'immagine.
        </P>
        <P ALIGN="justify">
            <IMG SRC="nonna.jpg" ALIGN="right" WIDTH="60" HEIGHT="60" HSPACE="15" BORDER="3"> Qui l'immagine è stata messa a
destra; l'immagine appare distorta perché le abbiamo dato delle dimensioni non proporzionate con le originali. Si noti lo spazio
orizzontale lasciato ai lati della immagine.
        </P>
        ....
    </BODY>
</HTML>
```

IMMAGINI

Qui l'immagine è stata messa a sinistra e il testo alla sua destra (con allineamento giustificato) su più righe.

Il testo continua al lato dell'immagine nonostante il segno di paragrafo.

Qui invece si è inserita una immagine dentro il testo, e siccome era piccola si è potuta inserirla centrata con il testo, in modo da trovare una cosa accettabile graficamente.

Immagine mancant

Qui sopra l'immagine non viene trovata, oppure non viene mostrata, e al suo posto compare il testo alternativo. Notare come il testo viene troncato se non entra nello spazio. Con Netscape il testo compare solo quando si passa col mouse sopra l'immagine.

Qui l'immagine è stata messa a destra; l'immagine appare schiacciata perché le abbiamo dato delle dimensioni non proporzionate con le originali. Si noti il bordo e lo spazio orizzontale lasciato ai lati della immagine.

IPER TESTO

- Abbiamo visto come inserire testo e immagini all'interno di un documento web e più avanti vedremo come dar loro una diversa impostazione grafica, come sistemarli meglio mediante le tabelle e gli elenchi.
- **il collegamento ipertestuale.** È chiamato anche link e la sua funzione è di rimandare ad un'altra sezione della pagina stessa, oppure ad un'altra pagina presente sullo stesso sito o su un sito esterno. Esso permette inoltre di scaricare o eseguire programmi, immagini, file audio e video, presenti sul sito o su altri server, e tante altre cose.
- Consente inoltre una lettura non sequenziale del documento, secondo la volontà del visitatore, che può saltare facilmente da un documento all'altro o da una sezione all'altra.
- **Il tag utilizzato per introdurre un collegamento ipertestuale è <A>, che però scritto così non ha alcun effetto e necessita di alcuni attributi**
- Gli attributi possibili sono 6 e servono a definire la destinazione del collegamento, e il modo in cui questo collegamento viene espletato. Gli attributi sono: HREF, NAME, TITLE, TARGET, REL e REV

IPER TESTO

- L'attributo che permette di creare un collegamento ipertestuale verso un'altra sezione o un altro documento è HREF, che definisce appunto il collegamento ipertestuale (Hypertext REFerence)
- L'attributo NAME="..." permette di inserire un segnalibro all'interno di una pagina, in modo da potercisi riferire tramite un collegamento situato in un'altra sezione della pagina o in un'altra pagina. È come mettere una bandieruola all'interno della pagina ed utilizzarla come riferimento per la navigazione. Usare un segnalibro all'interno di un collegamento ipertestuale vuol dire recarsi non all'inizio del documento da ad un punto preciso della pagina, esattamente lì dove è posizionato il segnalibro.
- La sintassi corretta del tag in questo caso è: <A NAME="nome">[testo opzionale]</A>.
  Siccome questo tag serve esclusivamente per porre un segnalibro all'interno del documento il testo può anche non essere scritto. Il testo "nome" invece è obbligatorio e deve essere univoco
- all'interno del documento altrimenti il computer dovrebbe scegliere tra due elementi uguali. I segnalibri vengono molto utilizzati per creare glossari, FAQ (Frequently Asked Questions) ed elenchi in genere.

IPER TESTO

- L'attributo HREF e la sua sintassi corretta è <A HREF="indirizzo">[Testo HTML]</A>.
- La dicitura "testo html specifica un testo o anche un qualsiasi contenuto di tipo HTML che indica l'ipertesto all'interno della pagina; può essere un testo, una immagine o un qualunque altro elemento HTML. Nel caso sia un testo questo viene evidenziato con una sottolineatura e in colore azzurro (ma ciò si può anche modificare) mentre nel caso sia una immagine questa viene in genere evidenziata con un bordo colorato (che si può anche togliere).
- La dicitura "indirizzo" indica l'indirizzo assoluto o relativo di un qualsiasi documento o file
- Si parla di indirizzo assoluto se esso è nella forma "http://www.canoro.org/guide/html/corso.html", perché viene specificato l'indirizzo del server web su cui è posto il documento
- L'indirizzo è invece relativo se si considera a partire dalla posizione attuale del documento, ad esempio "../sport/schede.html". Se il documento sta nella cartella corrente (quella in cui sta la pagina che stiamo visualizzando) allora si può scrivere solo il nome del file, ad esempio "html-09.html"

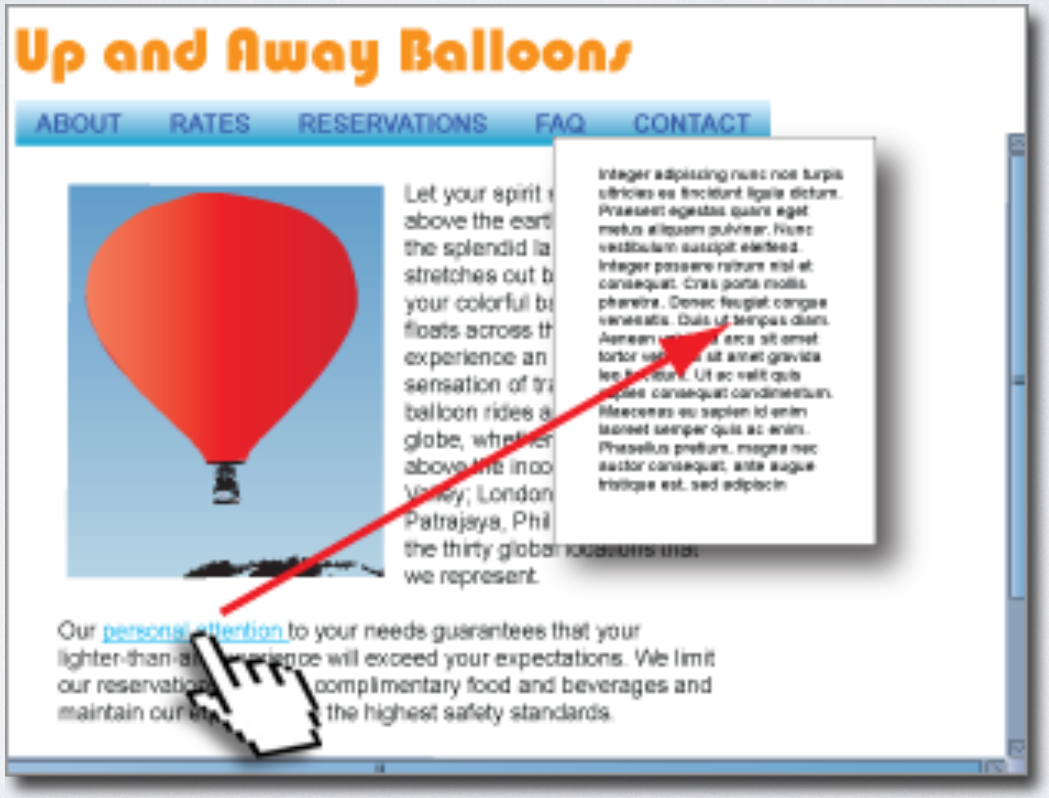
IPER TESTO

- Infine ci si può riferire ad un particolare punto di una pagina web, e non solo all'inizio del documento. Ciò si può fare mediante i segnalibri introdotti appena sopra. Il nome del segnalibro si indica alla fine dell'indirizzo, preceduto dal simbolo #, ad esempio, se avessimo messo un segnalibro alla fine di questa pagina e lo avessimo chiamato "fondo" si può fare un collegamento ad esso scrivendo "#fondo" come indirizzo.

- Mediante i segnalibri vale la stessa distinzione tra indirizzi relativi e assoluti; ad esempio le tre scritte qui sotto sono equivalenti:
  <A HREF="http://www.canoro.org/guide/html- 10.html#fondo"> Clicca</A> (indirizzo assoluto)

- <A HREF="html-10.html#fondo">Clicca</A> (indirizzo relativo)
  <A HREF="#fondo">Clicca</A> (indirizzo relativo)
  e producono lo stesso effetto di indirizzare l'utente alla fine della pagina corrente, se questa si chiama "html-10.html"

IPER TESTO


Up and Away Balloons
ABOUT
RATES
RESERVATIONS
FAQ
CONTACT
Our personal attention to your needs guarantees that your
the highest safety standards.

IPER TESTO

- Naturalmente in fondo alla pagina avremo messo il segnalibro: <A NAME="fondo"></A>.
  Infine si può usare un collegamento ipertestuale per mostrare una immagine, oppure per ascoltare o scaricare una sequenza audio/video oppure per scaricare qualunque file.
- Ecco alcuni esempi; nel primo descriviamo come legare una immagine ad un collegamento ipertestuale (si clicca e si vede l'immagine), nel secondo si consente all'utente di scaricare un piccolo file.
- 1° esempio:
- <A HREF="napoli.jpg"><IMG SRC="mini-napoli.jpg"></A>
- Questa riga produce l'effetto di visualizzare una immaginina e di legare ad essa un collegamento ipertestuale per visualizzare la stessa immagine ingrandita; per far aprire l'immagine in un'altra finestra si utilizza l'attributo TARGET, senza quell'attributo l'immagine sarebbe visualizzata nella finestra corrente, sostituendola e facendo sparire il documento di partenza.

IPER TESTO

- 2° esempio:
  <A HREF="battle.exe">CLICCA QUI</A> per scaricare il gioco....
  Questa riga produce l'effetto di visualizzare alcune parole a cui è associato un link che porta ad un file eseguibile, un gioco di Battaglia Navale per PC. All'utente basterà fare click e, a meno che non ci siano impostazioni di protezione contro i file eseguibili da parte del suo browser, potrà scaricare immediatamente il file sul proprio pc o avviarlo direttamente

- Il collegamento si adatta a qualunque tipo di file, eseguibile, compresso, immagine, testo, etc..

- Quando si mettono collegamenti per scaricare file di grandi dimensioni è sempre una buona usanza segnalare all'utente la dimensione dei file

IPER TESTO

- L'attributo TARGET="..." all'interno di un collegamento ipertestuale permette di aprire la nuova pagina in una finestra specifica che vogliamo noi, e non soltanto nella finestra attiva, ossia quella in cui è presente il collegamento ipertestuale
- Questo attributo è utilizzato soprattutto con i frame, che sono delle divisioni della finestra del browser in più finestre e che permettono la visualizzazione contemporanea di più pagine web e soprattutto una navigazione più snella. Ebbene, dando un nome a ciascun frame è possibile indirizzare ciascun collegamento ipertestuale in un frame piuttosto che in un altro; è possibile, inoltre, far aprire la nuova pagina in una altra finestra, e questo è utile se si vuole che il visitatore navighi nella nuova pagina pur continuando a visualizzare la pagina di partenza
- I valori che può assumere l'attributo TARGET sono "_blank", per indicare che la nuova pagina verrà visualizzata in una nuova finestra; "_parent", che, in una divisione in più frame, indica la pagina che contiene il frame attivo (se una pagina è divisa in frame, questo attributo cancella la divisione e la nuova pagina sarà visualizzata nella pagina intera); "_top", che cancella la divisione in frame e apre la nuova pagina a finestra intera; inoltre questo attributo può assumere i valori "nome", in cui evidentemente il testo "nome" indica il nome che verrà di volta in volta dato a ciascun frame della divisione

Dott. Antonio Giovanni Lezzi

PROGRAMMA DELLA GIORNATA

- Caratteri speciali
- Favicon
- Tabelle
- Celle delle tabelle
- Elenchi
- Contenitori

CARATTERI SPECIALI

- Ci sono molti caratteri che, se scritti direttamente in un file HTML, non vengono visualizzati correttamente del browser perché sono caratteri speciali
- Rientrano in questo insieme lettere accentate, lettere con dieresi o con cediglia, simboli particolari quali ©, ®, §
- Sono caratteri speciali anche '<' e '>', per un'ovvia ragione: quando il browser li incontra non pensa che essi siano caratteri da visualizzare come tutti gli altri, ma pensa che si tratti dell'inizio o della fine di un tag
- L'International Standards Organization ha definito un insieme standard di caratteri chiamato Latin-1 che contiene tutti i caratteri e simboli utilizzati dalle lingue occidentali (per l'esattezza, contiene 236 caratteri differenti). In questo standard (ISO 8859-1) ad ogni carattere è associato un codice numerico, e ad alcuni anche un nome

CARATTERI SPECIALI

- In HTML è possibile visualizzare un qualunque carattere dell'insieme Latin-1 utilizzando uno di questi due sistemi:
  - simbolo '&' seguito dal nome del carattere seguito da un punto e virgola
    esempio: à
  - simbolo '&' seguito dal simbolo '#' e poi dal numero del carattere seguito da un punto e virgola
    esempio: à
- Volendo questo sistema può essere utilizzato per visualizzare qualunque carattere; naturalmente esso è necessario solo per i caratteri speciali

Carattere Speciale	HTML Code *(case sensitive)*	Unicode	Nome o significato
«	«	«	virgolette sinistre in stile europeo
»	»	»	virgolette destre in stile europeo
–	–	–	trattino o meno
—	—	—	trattino lungo
¡	¡	¡	punto esclamativo invertito
¿	¿	¿	punto di domanda invertito
"	"	"	virgolette
“	“	“	virgolette sinistre inclinate
”	”	”	virgolette destre inclinate
‘	‘	‘	virgoletta sinistra inclinata
’	’	’	virgoletta destra inclinata
(blank space)			spazio vuoto

Simboli			
&	&	&	e commerciale
¢	¢	¢	simbolo di *cent*
©	©	©	simbolo di copyright
÷	÷	÷	simbolo di divisione
>	>	>	simbolo maggiore
<	<	<	simbolo minore
µ	µ	µ	simbolo micron
·	·	·	punto centrale
¶	¶	¶	delimitatore di paragrafo o simbolo di paragrafo
±	±	±	più o meno
€	€	€	simbolo valuta Euro
£	£	£	simbolo valuta Sterlina britannica *Pound*
®	®	®	simbolo di *marchio registrato*
§	§	§	delimitatore di sezione
™	™	™	simbolo *trademark*
¥	¥	¥	simbolo valuta Yen Giapponese

Diacritici			
á	á	á	*a* minuscola accentata con accento acuto
Á	Á	Á	*A* maiuscola accentata con accento acuto
à	à	à	*a* minuscola accentata con accento grave
À	À	À	*A* maiuscola accentata con accento grave
â	â	â	*a* minuscola con accento circonflesso
Â	Â	Â	*A* maiuscola con accento circonflesso
å	å	å	*a* minuscola con anello o occhiello
Å	Å	Å	*A* maiuscola con anello o occhiello
ã	ã	ã	*a* minuscola con tilde
Ã	Ã	Ã	*A* maiuscola con tilde
ä	ä	ä	*a* minuscola con dieresi
Ä	Ä	Ä	*A* maiuscola con dieresi
æ	æ	æ	*ae* minuscola con legatura fonetica
Æ	Æ	Æ	*AE* maiuscola con legatura fonetica

CARATTERI SPECIALI COMUNI

Simbolo	Codice HTML
à	à
è	è
é	é
ì	ì
ò	ò
ù	ù
&	&

FAVICON

- Favicon è un termine inglese, contrazione di favorite icon e indica un'icona associata a una particolare pagina web
- La favicon è una piccola immagine, spesso un logo, pertinente ai contenuti del sito web correlato. La favicon viene visualizzata alla sinistra dell'URL nella barra degli indirizzi di un browser, nel momento in cui si naviga su un sito che ne è provvisto
- Nata come funzionalità di Microsoft Internet Explorer versione 5, in seguito è stata integrata su molti altri browser, tra cui Firefox, Mozilla, Opera, Safari, Chrome e Konqueror
- In origine la favicon era semplicemente posta nella directory radice del webserver con il nome favicon.ico e usata direttamente da Internet Explorer
- Anche se questo metodo rimane funzionante, è stato introdotto un apposito tag HTML per specificare la posizione del file

FAVICON

- Il tag viene posto nella sezione head di un file HTML con sintassi:
  <link rel="icon" href="http://[percorso]/favicon.ico" />

- In questo modo ogni immagine delle giuste dimensioni (16×16 o 32×32 ma anche 48×48 pixel, se .ico) può essere usata come favicon. Alternativamente al formato .ico, possono essere usati anche i formati .gif e .png di qualsiasi dimensione.

- Sfruttando le potenzialità del formato .gif è possibile creare delle favicon animate, anche se sono supportate solo da alcuni browser. Per usare i formati GIF o PNG i tag rispettivamente necessari sono:
  <link rel="icon" href="http://[percorso]/favicon.gif" type="image/gif" />
  <link rel="icon" href="http://[percorso]/favicon.png" type="image/png" />

- Se si carica una favicon per la prima volta, oppure si sostituisce una precedente, per visualizzarla nel browser può essere necessario cancellare la cache, non essendo a volte sufficiente ricaricare la pagina

TABELLE

- Le prime versioni di HTML non prevedevano un layout elaborato, né tantomeno la nozione di tabella.
- Le tabelle sono in assoluto lo strumento di formattazione più usato; infatti, non vengono utilizzate solo per visualizzare dati di fogli elettronici e database, ma anche per implementare testo su colonne multiple, o posizionare oggetti in punti particolari della pagina.
- **Una tabella è definita dal tag contenitore <TABLE> ... </TABLE>** all'interno del quale si imposta la tabella riga per riga e poi cella per cella

TABELLE

- **All'interno della tabella una riga si specifica mediante il tag <TR> ... </TR> (Table Row);** tutto quello che è compreso tra questi tag appartiene alla stessa riga della tabella.

- **All'interno di una riga vengono poi specificate le celle della tabella, mediante il tag contenitore <TD> ... </TD> (Table Data)**

- All'interno di questi tag abbiamo volutamente lasciato dei puntini .... Questo per indicare che al loro posto va inserito qualcosa; ed infatti non avrebbe senso definire un elemento contenitore quale una tabella o una cella e poi non metterci nulla dentro, tranne in pochi casi, quando si vuole dare una certa impostazione grafica alla tabella o ad un gruppo di tabelle.

TABELLE

- Vediamo subito l'esempio di una tabella semplice, ma prima definiamo un attributo del tag <TABLE>, l'attributo BORDER="n" che serve a dare un bordo alla tabella e alle sue celle; se è assente significa che la tabella non ha bordo e saranno visibili solo i contenuti delle celle, ma non la struttura. Il numero n sta ad indicare lo spessore del bordo in punti.

- La sintassi è <TABLE BORDER="1">

- Vogliamo ora realizzare una tabella composta da 5 righe per 4 colonne

TABELLE

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#ECAFAF">
        <CENTER><H2>Vendite Hardware</H2>
            <TABLE BORDER="1">
                <TR>
                    <TD></TD><TD>1995</TD><TD>1996</TD><TD>1997</TD>
                </TR>
                <TR>
                    <TD>Computer</TD><TD>23</TD><TD>41</TD> <TD>123</TD>
                </TR>
                <TR>
                    <TD>Stampanti</TD><TD>7</TD><TD>31</TD><TD>98</TD>
                </TR>
                <TR>
                    <TD>Scanner</TD><TD>-</TD><TD>2</TD><TD>11</TD>
                </TR>
                <TR>
                    <TD>Modem</TD><TD>12</TD><TD>24</TD><TD> 47</TD>
                </TR>
            </TABLE>
        </CENTER>
    </BODY>
</HTML>
```

TABELLE

Vendite Hardware

	2003	2004	2005
Computer	23	41	123
Stampanti	7	31	98
Scanner	-	2	11
Modem	12	24	47

TABELLE

- Da questo esempio vediamo come non ci siano grosse difficoltà nell'uso di tabelle nell'impostazione del testo, e neanche ce ne saranno quando andremo a costruire tabelle più complesse con qualche elemento grafico in più
- Quello che occorre sempre tenere a mente è che HTML interpreta le tabelle leggendone prima una riga e poi l'altra, ed è così che dovremo cercare di costruire una tabella, curandone l'impostazione una riga dopo l'altra
- Il tag <TABLE> riconosce gli attributi ALIGN, WIDHT, HEIGHT, BORDER, CELLSPACING e CELLPADDING

TABELLE

- L'attributo ALIGN serve per allineare la tabella con il bordo sinistro o destro della finestra oppure mettere la tabella al centro della pagina; i valori che può assumere sono infatti "left", "right" e ‘‘center"
- Gli attributi WIDTH e HEIGHT servono per impostare le dimensioni della tabella, in punti o in percentuale della intera finestra. Porre WIDTH="100%" vuol dire far occupare alla tabella tutto lo spazio a sua disposizione in senso orizzontale
- L'attributo CELLSPACING="n" consente di specificare la distanza tra i bordi di due celle adiacenti o tra una cella ed il bordo esterno; l'impostazione di default assegna a tale attributo il valore 2
- L'attributo CELLPADDING="n" consente di impostare la distanza tra il testo ed il bordo della cella circostante; se non impostato assume il valore 1

TABELLE

- Il tag <TR> è il più semplice e riconosce gli attributi ALIGN e VALIGN
- Il primo serve, all'interno di tutte le celle della riga, per allineare il testo a sinistra, destra o al centro ("left", "right" e "center"); il valore predefinito è "left" e questo attributo può essere annullato reimpostando l'allineamento cella per cella
- L'attributo VALIGN serve per allineare il testo in senso verticale, se c'è lo spazio; i valori possibili sono "top", "middle" e "bottom" e allineano il testo in alto, al centro e in basso dentro ciascuna cella della riga; naturalmente anche questo attributo può essere annullato reimpostando l'allineamento cella per cella

TABELLE

- Il tag <TD> riconosce gli attributi ALIGN, VALIGN, WIDTH, HEIGHT, ROWSPAN, COLSPAN, BGCOLOR e NOWRAP
- i primi due hanno lo stesso significato che in <TR> e servono per allineare il contenuto della cella in senso orizzontale e verticale
- WIDTH e HEIGHT impostano le dimensioni della cella, rispettivamente la larghezza e l'altezza, entrambi assumono valori espressi sia in numero di punti che in percentuali dello spazio a loro disposizione
- ROWSPAN="x" e COLSPAN="y" permettono di unire due o più celle, e specificano di quanto la cella deve essere espansa; indicare COLSPAN="3" vuol dire che la cella in questione deve occupare lo spazio di 3 colonne
- L'attributo BGCOLOR, come si può ben immaginare permette di impostare il colore di fondo della cella, espresso in notazione esadecimale "#RRGGBB" o mediante il nome del colore.

TABELLE

- L'attributo NOWRAP che forza il contenuto della cella a rimanere visualizzato sulla stessa riga, senza ritorno a capo; volendo andare a capo si utilizzano i tag <BR> e <P>
- Il tag <CAPTION>, posto all'interno della dichiarazione della tabella, permette di inserire una intestazione o una didascalia alla tabella. Esso accetta l'attributo ALIGN con i valori "top" e "bottom", con il quale è possibile mettere il titolo sopra o sotto alla tabella.
- Infine nel visualizzare le colonne è possibile dar loro una intestazione, mediante il tag <TH> (Table Heading) al posto dei tag <TD>. Nell'esempio che segue aumenteremo le dimensioni della tabella mediante gli attributi WIDTH e HEIGHT posti dentro il tag <TABLE>; questi attributi ammettono come valore una percentuale della finestra a loro disposizione oppure il numero di punti che si vuole come dimensione per la tabella
- Notiamo come il testo inserito nel tag <CAPTION> non subisce formattazione, e quindi sarebbe bene che gliene dessimo una; mentre il tag <TH> centra il testo e lo visualizza in grassetto

TABELLE

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE> ...
    </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <CENTER><H2>Vendite Hardware</H2>
            <TABLE BORDER="1" WIDTH="90%" HEIGHT="160">
                <CAPTION ALIGN="top">Negli anni 2003-2005 </CAPTION>
                <TR>
                    <TH></TH><TH>1995</TH><TH>1996</TH><TH> 1997</TH>
                </TR>
                <TR>
                    <TD>Computer</TD><TD>23</TD><TD>41</TD> <TD>123</TD>
                </TR>
                <TR>
                    <TD>Stampanti</TD><TD>7</TD><TD>31</TD> <TD>98</TD>
                </TR>
                <TR>
                    <TD>Scanner</TD><TD>-</TD><TD>2</TD><TD>11</TD>
                </TR>
                <TR>
                    <TD>Modem</TD><TD>12</TD><TD>24</TD><TD>47</TD>
                </TR>
            </TABLE>
        </CENTER>
        ....
        <P ALIGN="justify">Notiamo come il testo inserito nel tag <CAPTION> non subisce formattazione, e quindi sarebbe bene che gliene dessimo una;
mentre il tag <TH> centra il testo e lo visualizza in grassetto.
    </BODY>
</HTML>
```

TABELLE

Vendite Hardware

Negli anni 2003-2005			
	2003	**2004**	**2005**
Computer	23	41	123
Stampanti	7	31	98
Scanner	-	2	11
Modem	12	24	47

....

CELLE DELLE TABELLE

- Cosa può contenere una cella?
- In pratica tutto, qualunque testo HTML, un testo, delle immagini
- La possibilità di porre qualunque cosa all'interno di una cella di una tabella permette di ottenere particolari layout grafici che non si potrebbero realizzare in altro modo
- Togliendo infine i bordi della tabella e facendone sparire la struttura è possibile ottenere effetti grafici straordinari.
- Mediante una tabella è possibile ad esempio impostare un menù di collegamenti, mettendo una immagine in ciascuna cella della tabella, e legando ad essa un collegamento ipertestuale. In questo modo si rende la visualizzazione indipendente da tutte le opzioni selezionate per la visualizzazione nei browser (ad esempio la dimensione dei caratteri)

CELLE DELLE TABELLE

- Disponendo di tante immagini ciascuna per un collegamento, l'impostazione mediante una tabella rende la visualizzazione dell'intero menù indipendente dalle dimensioni dello schermo, infatti mettendo semplicemente le immagini una dietro l'altra e andando a capo col comando <BR> risultava che con uno schermo troppo piccolo le immagini che non entrano nello schermo vanno comunque a capo.

- L'esempio che segue mostra un menù di collegamenti realizzato tramite una tabella. Per semplicità nel testo dell'esempio sono state tagliate alcune righe della tabella, ed è mostrata la sintassi solo della prima ed ultima riga

CELLE DELLE TABELLE

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <CENTER><H2>Menù di collegamenti</H2>
            <TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
                <TR>
                    <TD><A HREF="attivita.html"><IMG SRC="1.gif"></A></TD> <TD><A HREF="notizie.html"><IMG SRC="2.gif"></A></TD>
                    <TD><A HREF="catalogo.html"><IMG SRC="3.gif"></A></TD> <TD><A HREF="adesioni.html"><IMG SRC="4.gif"></A></TD>
                </TR>
                [...]
                <TR>
                    <TD> </TD>
                    <TD COLSPAN="2"><A HREF="iscrizione.html">
                        <IMG SRC="13.gif"></A></TD> <TD> </TD>
                </TR>
            </TABLE>
        </CENTER>
        ....
    </BODY>
</HTML>
```

CELLE DELLE TABELLE

CELLE DELLE TABELLE

- Nella tabella abbiamo impostato BORDER, CELLSPACING e CELLPADDING tutti a zero per fare in modo che le immagini venissero tutte attaccate l'una all'altra
- L'esempio fa uso anche dell'attributo COLSPAN che verrà descritto immediatamente di seguito

CELLE DELLE TABELLE

- Gli attributi più complessi del tag <TD>, complessi non perché siano di difficile comprensione o utilizzo, ma perché un loro utilizzo eccessivo all'interno della progettazione di una tabella può portare ad effetti indesiderati se si fa un po' di confusione.
- Gli attributi ROWSPAN="x" e COLSPAN="y" che permettono di espandere una cella in senso orizzontale e verticale, e di occupare il posto di un numero di celle specificato dal prodotto dei valori x e y. Se non sono specificati si intendono ovviamente ROWSPAN="1" e COLSPAN="1", ossia una cella occupa il posto esattamente di 1 cella.
- Specificare COLSPAN="3" vuol dire che la cella si espanderà di 2 posti in senso orizzontale, occupando lo spazio di 3 celle, e la riga avrà due celle in meno.
- Occorre tenere in considerazione che l'espansione viene effettuata solamente con celle adiacenti che appartengono esclusivamente alla stessa riga e alla stessa colonna

CELLE DELLE TABELLE

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
        .... </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <CENTER><H2>Espansioni orizzontali e verticali</H2>
            <TABLE BORDER="1" WIDTH="90%">
                <TR>
                    <TD ALIGN="center">UNO</TD>
                    <TD ALIGN="center" COLSPAN="2">DUE</TD>
                    <TD ALIGN="center">TRE</TD>
                    <TD ALIGN="center">QUATTRO</TD>
                </TR>
                <TR>
                    <TD ALIGN="center">CINQUE</TD>
                    <TD ALIGN="center">SEI</TD>
                    <TD ALIGN="center">SETTE</TD>
                    <TD ALIGN="center" ROWSPAN="3">OTTO</TD>
                    <TD ALIGN="center">NOVE</TD>
                </TR>
                <TR>
                    <TD ALIGN="center">DIECI</TD>
                    <TD ALIGN="center">UNDICI</TD>
                    <TD ALIGN="center">DODICI</TD>
                    <TD ALIGN="center">TREDICI</TD>
                </TR>
                <TR>
                    <TD ALIGN="center">QUATTORDICI</TD>
                    <TD ALIGN="center">QUINDICI</TD>
                    <TD ALIGN="center">SEDICI</TD>
                    <TD ALIGN="center">DICIASSETTE</TD>
                </TR>
            </TABLE>
        </CENTER>
    ....
    <P ALIGN="justify">Notiamo come la riga 1 sia formata da 4 elementi perché l'elemento "DUE" occupa 2 posizioni; la riga 2 è formata da tutti e 5 gli elementi, di cui uno si espande in verticale; le
righe 3 e 4 si compongono di 4 elementi, perché l'elemento "OTTO" appartiene alla seconda riga.
    </BODY>
</HTML>
```

CELLE DELLE TABELLE

Espansioni orizzontali e verticali

<table>
<tr><td>UNO</td><td colspan="2">DUE</td><td>TRE</td><td>QUATTRO</td></tr>
<tr><td>CINQUE</td><td>SEI</td><td>SETTE</td><td rowspan="3">OTTO</td><td>NOVE</td></tr>
<tr><td>DIECI</td><td>UNDICI</td><td>DODICI</td><td>TREDICI</td></tr>
<tr><td>QUATTORDICI</td><td>QUINDICI</td><td>SEDICI</td><td>DICIASSETTE</td></tr>
</table>

....

Notiamo come la riga 1 sia formata da 4 elementi perché l'elemento "DUE" occupa 2 posizioni; la riga 2 è formata da tutti e 5 gli elementi, di cui uno si espande in verticale; le righe 3 e 4 si compongono di 4 elementi, perché l'elemento "OTTO" appartiene alla seconda riga.

CELLE DELLE TABELLE

- Avremmo potuto anche unire le celle DUE, SEI e SETTE in modo che la cella che ne risultava veniva parzialmente ad intersecarsi con la cella OTTO

- In tal modo le celle SEI e SETTE sparivano, la cella DUE avrebbe avuto ROWSPAN="2" e COLSPAN="2", per cui la prima riga avrebbe avuto quattro celle, la seconda ne avrebbe avute 3, la terza e la quarta rimangono con 4 elementi

ELENCHI

- Per visualizzare una serie di elementi o un elenco di oggetti ci si potrebbe anche accontentare dei normali paragrafi
- Nella vita di tutti i giorni, facciamo spesso uso degli elenchi, per fare la lista della spesa (tipico), per fare la lista dei nostri appuntamenti... Persino le cartelle le directory del nostro computer o le canzoni di un CD sono visualizzate tramite un elenco...
- Per questo l'HTML ci mette a disposizione ben 5 tipi di elenchi, sebbene solo tre siano realmente utilizzati, ci occuperemo:
- **elenchi non numerati (o non ordinati)**
- **elenchi ordinati**
- **elenchi di definizioni**

ELENCHI NON ORDINATI

- Gli elenchi non ordinati (o puntati, o non numerati) sono elenchi i cui elementi sono descritti senza una sequenza logica o temporale ma che sono in qualche modo correlati
- sono oggetti che sono semplicemente raggruppati in una unica lista.
- Tutti i tipi di elenco sono rappresentati da HTML allo stesso modo, con un tag che definisce l'elenco e un altro che indica ciascuna voce dell'elenco
- Il tag che indica un elenco non ordinato è <UL> (Unordered List), mentre il tag che indica una voce dell'elenco è <LI> (List Item), che può non avere il tag di chiusura </LI>
- Entrambi i tag possono avere l'attributo TYPE che permette di scegliere il tipo dei punti elenco e i valori che può assumere questo attributo sono "circle", "square" o "disc"; il valore predefinito è "disc", un cerchio pieno
- Specificare l'attributo TYPE nel tag <UL> vuol dire assegnare a tutte le voci lo stesso simbolo, specificarlo nel tag <LI> vuol dire assegnare quel simbolo a quel particolare elemento soltanto

ELENCHI NON ORDINATI

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
        ....
    </HEAD>
    <BODY BGCOLOR="#FFEBCD">
        <H2 ALIGN="center">Elenchi non ordinati</H2>
        <P ALIGN="justify">Elenco dei programmi presenti su questo server.
        <UL TYPE="square">
            <LI>ACDSee - Un programma di grafica
            <LI TYPE="disc">CuteFTP - Un programma per trasferire i file su un server FTP
            <LI>Gozilla - Facilita il download dei file <LI>Napster - Condividere gli MP3 in rete
            <LI TYPE="circle">Winamp - Lettore MP3 <LI>Winzip - Programma per comprimere i file
        </UL>
        ....
        <P ALIGN="justify">Notiamo come tutti gli elementi sono impostati a square mediante la
definizione nel tag </UL>, mentre per la seconda e quinta voce sono stati scelti simboli diversi.
    </BODY>
</HTML>
```

ELENCHI NON ORDINATI

Elenchi non ordinati

Elenco dei programmi presenti su questo server.

- ✍ ACDSee - Un programma di grafica
- ? CuteFTP - Un programma per trasferire i file su un server FTP
- ✍ Gozilla - Facilita il download dei file
- ✍ Napster - Condividere gli MP3 in rete
- o Winamp - Lettore MP3
- ✍ Winzip - Programma per comprimere i file

....

ELENCHI NON ORDINATI

- Notiamo come tutti gli elementi sono impostati a square mediante la definizione nel tag </UL>, mentre per la seconda e quinta voce sono stati scelti simboli diversi.

- Nello scrivere il codice, abbiamo dato una certa indentatura alle voci interne degli elenchi in modo che siano meglio visibili in fase di analisi e correzione del testo HTML

ELENCHI ORDINATI

- Gli elenchi ordinati sono utilizzati per descrivere una ben precisa sequenza di eventi o operazioni e a tale scopo ogni voce è contrassegnata da un simbolo di ordine come ad esempio un numero o un carattere.
- A tale scopo HTML ci fornisce il tag <OL> (Ordered List), che serve appunto ad indicare un elenco numerato di elementi
- Per indicare ciascuna voce dell'elenco su utilizza il tag <LI> (List Item), esattamente come per gli elenchi non ordinati.
- Questi tag possono avere l'attributo TYPE, che permette di scegliere il simbolo da usare come numerazione e mentre il tag <OL> può avere anche l'attributo START, che permette di specificare il valore da cui iniziare il conteggio
- È possibile utilizzare per ciascuna voce un simbolo diverso e partire la numerazione dell'elenco da un numero diverso da 1

ELENCHI ORDINATI

- L'attributo TYPE può assumere i valori "1" (predefinito), "A", "a", "I" e "i", che stanno ad indicare:
  1 - Stile di enumerazione standard: 1, 2, 3 ...
  A - mediante caratteri alfabetici maiuscoli: A, B, C ...
  a - mediante caratteri alfabetici minuscoli: a, b, c ...
  I - mediante caratteri romani maiuscoli: I, II, III, IV ... i - mediante caratteri romani minuscoli: i, ii, iii, iv ...

- L'attributo START può assumere valori da -2147483647 a 2147483647, e dare un numero che vada fuori da questo range fa si che la numerazione riparta dall'estremo opposto

- È da ricordare che solo lo stile di numerazione standard permette di utilizzare numeri negativi. Inoltre per specificare il punto di partenza per gli altri tipi si deve sempre utilizzare un numero (senza segno) e non quindi dei caratteri come sarebbe logico attendersi. Sarà quindi poi il browser a convertire il numero (inteso come indice di posizione) nel simbolo dello stile prescelto

ELENCHI ORDINATI

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
        .... </HEAD>
    <BODY BGCOLOR="#FFEBCD">
        <H2 ALIGN="center">Elenchi ordinati</H2>
        <P ALIGN="justify">Istruzioni per installare un programma
        <OL START="2">
            <LI>Decomprimere il file ZIP
            <LI>Eseguire il programma SETUP.EXE
            <LI TYPE="a"> Scegliere la directory di installazione
            <LI TYPE="I">Proseguire con l'installazione
        </OL>
    </BODY>
</HTML>
```

ELENCHI ORDINATI

Elenchi ordinati

Istruzioni per installare un programma.

2. Decomprimere il file ZIP
3. Eseguire il programma SETUP.EXE
d. Scegliere la directory di installazione
V. Proseguire con l'installazione

....

ELENCHI ORDINATI

- Notiamo come il conteggio parte volontariamente da 2 anziché da 1, e come gli elementi "4" e "5" sono visualizzati con altri simboli.
- La maggior parte dei tag contenitore di HTML può essere nidificata all'interno di altri tag; a questa regola non sfuggono gli elenchi. Unendo gli stili di elenco puntato e numerato possiamo ottenere un elenco più complesso

ELENCHI NIDIFICATI

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE> </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <H2 ALIGN="center">Elenchi annidati</H2>
        <P ALIGN="justify">Calendario delle attività.
        <OL>
            <LI>Febbraio
                <UL type="disc">
                    <LI>Visita alla galleria nazionale
                    <LI><I>Holiday on ice</I> a Londra
                </UL>
            <LI>Marzo
                <UL type="circle">
                    <LI>Escursione ad Assisi
                    <LI>Concerto di Offanbach
                    <LI>Visita al museo delle cere
                </UL>
                <LI>Aprile
                    <UL type="square">
                        <LI>Concerto di Barbican
                        <LI>Visita agli scavi di Ostia
                    </UL>
        </OL> ...
        <P ALIGN="justify">Nello scrivere il codice, abbiamo dato una certa indentatura alle voci interne degli elenchi in modo che siano
meglio visibili in fase di analisi e correzione del testo HTML.
    </BODY>
</HTML>
```

ELENCHI ORDINATI

Elenchi annidati

Calendario delle attività.

1. Febbraio
   - ? Visita alla galleria nazionale
   - ? *Holiday on ice* a Londra
2. Marzo
   - o Escursione ad Assisi
   - o Concerto di Offanbach
   - o Visita al museo delle cere
3. Aprile
   - ✍ Concerto di Barbican
   - ✍ Visita agli scavi di Ostia

....

ELENCHI, ALTRE TIPOLOGIE

- Oltre agli elenchi puntati e numerati, esiste un altro tipo di elenco, molto vicino agli elenchi non numerati
- È l'elenco di definizioni, ed è utilizzato quando si vuole visualizzare degli elementi a coppie, come un termine e la sua definizione, un capitolo e i suoi contenuti, o semplicemente si vuole realizzare un glossario
- L'implementazione di questo tipo di elenco è leggermente diversa da quella degli altri elenchi, perché occorre mostrare 2 elementi, che chiameremo per comodità "termine" e "definizione"
- Per specificare un elenco di definizioni si usa il tag <DL>, (Definition List)
- ogni termine verrà indicato tramite il tag <DT>; (Definition Term) e ogni definizione mediante il tag <DD>; (Definition Data)
- È facile notare che <DL> viene utilizzato allo stesso modo di <OL> o <UL>, mentre al posto di <LI> adesso abbiamo i due tag <DT> e <DD>
- Nessuno di questi tag riconosce attributi, vanno utilizzati semplicemente così, come li abbiamo introdotti

ELENCHI, INTERLINEA

- Un occhio di riguardo merita l'interlinea, ossia lo spazio tra una voce e l'altra dell'elenco
- Non tutti i tipi di elenco sono uguali e parimenti i browser non si comportano tutti allo stesso modo
- Quando si costruisce un elenco è bene controllare che ciò che viene visualizzato corrisponde o meno a quello che vorremmo rappresentare sullo schermo, e adottare una soluzione o l'altra a seconda dell'effetto grafico che si vuole ottenere.
- Naturalmente il testo presente nelle definizioni può essere formattato; negli esempi abbiamo sempre usato lo stesso tipo di carattere, ma ovunque questo può essere formattato diversamente, un diverso font, diversa dimensione, colore, può essere messo in grassetto, sottolineato, corsivo...

ELENCHI

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#80DCDC">
        <H2 ALIGN="center">Elenchi di definizioni</H2>
<P><B>Sommario</B>
        <DL>
            <DT>Inserimento del testo <DD>Impostazione del
testo <DD>Formattazione del testo
            <DT>Elementi multimediali <DD>Inserimento di
immagini <DD>Suoni <DD>Animazioni
        </DL> ....
    </BODY>
</HTML>
```

Elenchi di definizioni

Sommario

Inserimento del testo

Impostazione del testo
Formattazione del testo

Elementi multimediali

Inserimento di immagini
Suoni
Animazioni

....

CONTENITORI

- I tag HTML che normalmente vengono utilizzati come "contenitori" per i blocchi di testo sono:
  - Paragrafi - <p>
  - Div - (contenitore generico)
  - Span -
- Dal punto di vista visivo, la differenza tra questi tag consiste nel fatto che p produce un ritorno a capo ed un'interlinea vuoto; div produce solo il ritorno a capo; span non produce alcun ritorno a capo (normalmente è utilizzato all'interno di un <p> o).
- Da un punto di vista "tecnico" possiamo dire che mentre <p> e sono elementi di tipo block, è un elemento di tipo inline che, tradotto in parole semplici, significa che segue il flusso della linea senza generare un ritorno a capo.

CONTENITORI

- È bene precisare ancora che mentre <p> e sono dei tag specifici per il testo, il tag è un tag "contenitore" generico che può essere utilizzato per gestire la posizione e l'aspetto di qualsiasi elemento della pagina
- Ha caratteristiche polimorfiche (mediante l'uso dei CSS il tag può persino sostituire l'uso delle tabelle!)

CONTENITORI

```
<html>
<head>
<title>Titolo della pagina</
title>
</head>
<body>

<p>Testo...</p>
<div>Testo...</div>
<div>Testo... <span>Testo...</
span></div>

</body>
</html>
```

Testo...

Testo...
Testo... Testo...

CONTENITORI

- È possibile utilizzare l'uno, l'altro o l'altro ancora a seconda delle specifiche esigenze, tenendo presente che è possibile modificare i comportamenti di default, descritti in precedenza, attraverso i CSS.
- Per quanto riguarda il tag span è opportuno puntualizzare che il suo utilizzo non comporta in modo predefinito, nessun cambiamento visivo per il testo marcato, il suo utilizzo tipico consiste nel raggruppare elementi testuali per poi stilizzari mediante i CSS oppure manipolarli attraverso Javascript. Vediamo un esempio di utilizzo del tag span per effettuare una semplice stilizzazione:
- Sto leggendo una lezione del Manuale di HTML sulle slide di Antonio
- Possiamo quindi dire, forse impropriamente, che span è un tag di marcatura del testo "neutro" in quanto il suo aspetto e/o il suo funzionamento sono delegati allo sviluppatore mediante l'utilizzo dei fogli di stile CSS.

DIV

- Il tag DIV è uno dei tag più importanti del linguaggio HTML in quanto può essere definito come un elemento "neutro" del markup grazie al quale è possibile assolvere diversi compiti tra cui, soprattutto, la definizione di diverse porzioni all'intero di una pagina web.
- Il termine DIV nasce come abbreviazione di "divide" (separatore) e, in un certo senso, la funzione di questo tag consiste proprio in questo: creare delle sezioni all'interno di una pagina web al fine di separare diverse aree della pagina come, ad esempio, l'header, il menu ed il footer
- Nel web-design tradizionale (cioè precedente ad HTML5), le aree della pagina vengono definite mediante dei DIV opportunamente stilizzati mediante l'uso dei CSS. In questo modo, pertanto, il DIV estrinseca la sua funzionalità tipica

DIV

```
<!DOCTYPE html>
<html>
...
<body>

  <div id="container">

    <div id="header">...</div>

    <div id="menu">...</div>

    <div id="content">...</div>

    <div id="footer">...</div>

  </div>

</body>
</html>
```

DIV

- Come detto il tag DIV è "neutro" nel senso che, se non opportunamente stilizzato, non offre alcun risultato visibile all'interno della pagina web se non un semplice ritorno a capo (questo perchè il DIV, tecnicamente, è un elemento di tipo block e produce, cioè, un ritono a capo). Quindi:
  * Questa è una linea... *
  * Questa è un'altra linea… *

- Ne consegue (tornando al codice del primo esempio) che senza un'opportuna stilizzazione CSS il risultato sarà del tutto insoddisfacente: a differenza di altri tag che posseggono un aspetto caratterizzante (si pensi alle tabelle) il tag DIV è "invisibile" perché appartiene unicamente alla struttura della pagina.

- Scopo dei DIV, quindi, è quello di fungere da semplici contenitori il cui aspetto, dimensione, posizione e funzione devono essere definiti, di volta in volta, dal designer della pagina web.

- Normalmente, quindi, a ciascun div è assegnata una classe o un ID oppure, più raramente, una stilizzazione inline

- La conoscenza dei fogli di stile CSS, quindi, è un requisito indispensabile per lavorare con questo tag

DIV

- Per finire è bene ricordare che il tag DIV, proprio a causa della sua duttilità, è estremamente prezioso e spesso utilizzato in comunione con Javascript per fungere, ad esempio, da placeholder per il caricamento di contenuti dinamici. Vediamo un banalissimo esempio:

```
<script>
function carica_dati() {
  document.getElementById("caricadati").innerHTML = 'Contenuto
dinamico';
}
</script>
<div id="caricadati"></div>
<a href="javascript:carica_dati()">clicca qui</a>
```

Dott. Antonio Giovanni Lezzi

INTRODUZIONE AI FOGLI DI STILE

- L'HTML in origine è nato come linguaggio per formattare i documenti presenti sul Web. Proprio per questo motivo il contenuto (ad esempio qui il mio testo) e i tag che indicano uno stile o una colorazione del contenuto (ad esempio <font color="red">, che colora il testo di rosso) si trovavano mischiati allo stesso livello.
- Tuttavia vari anni di Web hanno fatto nascere l'esigenza di separare il contenuto dalla presentazione del contenuto medesimo
- La particolarità di questo linguaggio è che prevede (e incoraggia) la presenza di fogli di stile multipli, che agiscono uno dopo l'altro **in cascata**
- Questo permette di avere controllo sia da parte dell'autore che del lettore di un documento HTML (ognuno di quali può applicare uno o più fogli di stile al documento)

INTRODUZIONE AI FOGLI DI STILE

- Un'altra caratteristica vincente di CSS è l'indipendenza dall'insieme di elementi e attributi HTML
  - si può usare con qualunque versione di HTML
  - si può usare anche con linguaggi di markup simil-HTML (tipicamente XML)
- Se per esempio avessi tutti i titoli del mio documento in rosso e in grassetto, e a un certo punto decidessi di trasformarli in verde e in corsivo, con l'HTML classico (cioè l'HTML 3.2) dovrei andare a modificarmi a mano ogni tag contente le indicazioni della formattazione

INTRODUZIONE AI FOGLI DI STILE

```
<p>
  <font color="red">
    <b>titolo 1</b>
  </font>
</p>
```

diventerebbe:

```
<p>
  <font color="green">
    <i>titolo 1</i>
  </font>
</p>
```

INTRODUZIONE AI FOGLI DI STILE

- Ma se questa operazione non comporta difficoltà su una singola pagina, diventa insostenibile (o quantomeno difficoltosa, tanto che converrebbe scrivere un programma che effettuasse la conversione al posto nostro) su website molto grandi, a volte di centinaia di pagine.

- Proprio per questo è nata l'esigenza di separare il contenuto (la scritta "titolo 1"), dalla formattazione (il colore rosso e il grassetto)

- Per farlo è necessario utilizzare i fogli di stile, e il contenuto della pagina vista pocanzi diventerebbe qualcosa di questo genere:

```
<p class="formattaTitoli">
  titolo 1
</p>
```

la colorazione del testo verrebbe affidata alla classe "formattaTitoli", descritta altrove del documento, o su un file separato. Dunque basta editare la classe "formattaTitoli" per cambiare l'aspetto anche di centinaia di pagine.

INTRODUZIONE AI FOGLI DI STILE

- I CSS servono per gestire tutto il layout di un sito Web
- Grazie ad essi è possibile intervenire sulla formattazione del testo, sul posizionamento degli elementi grafici e sulla disposizione che questi elementi avranno rispetto a diversi media e device
- L'acronimo CSS sta per Cascading Style Sheets (fogli di stile a cascata) e designa un linguaggio di stile per i documenti web
- I CSS istruiscono un browser o un altro programma utente su come il documento debba essere presentato all'utente, per esempio definendone i font, i colori, le immagini di sfondo, il layout, il posizionamento delle colonne o di altri elementi sulla pagina, etc.

STORIA DEI CSS

- La storia dei CSS procede su binari paralleli rispetto a quelli di HTML, di cui vuole essere l'ideale complemento. Da sempre infatti, nelle intenzioni degli uomini del W3C, HTML dovrebbe essere visto semplicemente come un linguaggio strutturale, con lo scopo attinente la presentazione di un documento.
- L'ideale perseguito da anni si può sintetizzare con una nota espressione: separare il contenuto dalla presentazione.
- La prima specifica ufficiale di CSS (CSS 1) risale al dicembre del 1996
- Nel maggio 1998 è stata la volta della seconda versione: CSS 2. Niente stravolgimenti, ma molte aggiunte rispetto alla prima. CSS 2 non è altro che CSS 1 più alcune nuove proprietà, valori di proprietà e definizioni per stili non canonici come quelli rivolti alla stampa o alla definizione di contenuti audio
- Mentre prendeva corpo la specifica relativa ai CSS3, veniva anche effettuata una revisione della seconda specifica, denominata CSS 2.1, che ha raggiunto lo stato di raccomandazione ufficiale nel giugno 2011

USO DI CSS CON HTML

- HTML 4 prevede che tutti i tag di un documento possano avere attributi id e class
- **id** deve essere associato ad un identificativo univoco di un'occorrenza di un tag nell'intero documento (non può essere ripetuto)
- **class** deve essere associato ad un identificativo di classe che può essere usato più volte nel documento CSS può usare i valori di id e class per specificare che una regola si applica ad uno specifico elemento del documento o a tutti gli elementi della stessa classe

PERCHÉ SI CHIAMA CSS

- Il concetto di cascata in CSS significa che regole successive che si possono applicare ad un elemento del documento HTML devono essere applicate in sequenza
  Ad esempio, applicando in sequenza:
  p { font-family: Arial; font-size: 12pt; }
  p { color: red; font-size: 11pt; }
  p { margin-left: 15pt; color: green; }

- otterremo lo stesso risultato di
  p { font-family: Arial; font-size: 11pt; margin-left: 15pt; color: green; }

- Questa funzionalità è utile in particolare in due casi:

  - Quando si prevede di applicare più fogli di stile (insiemi di regole) allo stesso documento HTML

  - Quando si vogliono applicare regole generali (ad esempio a tutti i tag <p>) + regole specifiche per singoli elementi o classi (ad esempio a tutti i tag <p class="menu">)

ELEMENTI BLOCK

- Una pagina HTML resa a schermo da un browser è composta, di fatto, da un insieme di rettangoli (box). Non importa che si tratti di paragrafi, titoli, tabelle o immagini: si tratta sempre di box rettangolari.
- Osservando attentamente l'immagine si intuisce, però, che non tutti i box sono uguali. Alcuni contengono altri box al loro interno; altri sono invece contenuti all'interno dei primi. Alcuni, se si trovano (come accade) in mezzo al testo, lasciano che esso scorra loro intorno senza interrompere il suo flusso e senza andare a capo
- Questa considerazione molto semplice ci offre la rappresentazione della fondamentale distinzione tra elementi blocco (quelli contrassegnati dal bordo nero) ed elementi inline (quelli circondati dal bordo rosso).
- Gli elementi blocco sono box che possono contenere altri elementi, sia di tipo blocco che di tipo inline. Quando un elemento blocco è inserito nel documento viene automaticamente creata una nuova riga nel flusso del documento. Possiamo verificarlo inserendo in una pagina HTML queste due righe di codice:

```
<h1>Titolo</h1>
<p>Paragrafo</p>
```

- Le parole "titolo" e "paragrafo" appariranno su due righe diverse, perché <h1> e <p> sono elementi blocco.

ELEMENTI INLINE

- Gli elementi inline non possono contenere elementi blocco, ma solo altri elementi inline (oltre che, ovviamente, il loro stesso contenuto, essenzialmente testo)
- Come si può notare, quando sono inseriti nel documento non danno origine ad una nuova riga.
- Accanto a queste due fondamentali tipologie di elementi, il modello di formattazione dei CSS prevede altre due categorie: quella degli elementi lista e quella degli elementi connessi alle tabelle
- Tutti gli elementi HTML, in sintesi, hanno una loro modalità di presentazione predefinita e corrispondente alle tipologie appena viste
- Tramite i CSS possiamo modificare tale modalità attraverso la proprietà display. Grazie a quest'ultima, per fare solo un esempio, possiamo fare in modo che un titolo h1 (elemento blocco) venga mostrato come un elemento in linea; oppure far sì che un elemento a (inline) sia reso come un elemento blocco

ELEMENTI RIMPIAZZATI E NON

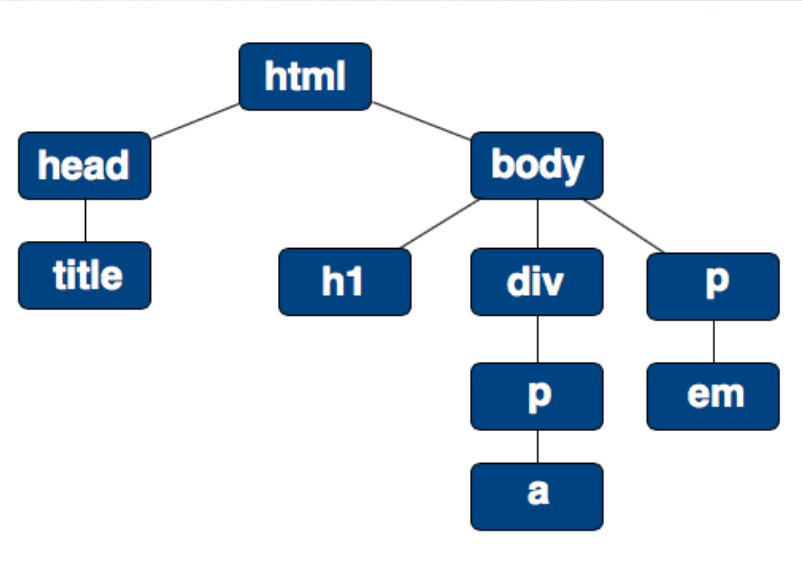
- Un'altra distinzione da ricordare è quella tra elementi rimpiazzati ed elementi non rimpiazzati
- I primi sono elementi di cui uno user agent (il motore di un browser) conosce solo le dimensioni intrinseche, ovvero, quelli in cui altezza e larghezza sono definite dall'elemento stesso e non da ciò che lo circonda
- L'esempio più tipico di elemento rimpiazzato è <img> (un'immagine), altri elementi rimpiazzati sono: <input>, <textarea>, <select> e <object>
- Tutti gli altri elementi sono in genere considerati non rimpiazzati.
- La distinzione è importante perché per alcune proprietà è diverso il trattamento tra l'una e l'altra categoria, mentre per altre il supporto è solo per la prima, ma non per la seconda

STRUTTURA AD ALBERTO DI UN DOCUMENTO

- Altro concetto fondamentale da assimilare per una corretta applicazione dei CSS è quello della struttura ad albero di un documento
- Il meccanismo fondamentale dei CSS è infatti l'ereditarietà
- Esso fa sì che molte proprietà impostate per un elemento siano automaticamente ereditate dai suoi discendenti
- Sapersi districare nella struttura ad albero significa padroneggiare bene questo meccanismo e sfruttare al meglio la potenza del linguaggio
- Tutti i concetti che spiegheremo qui di seguito sono definiti nel cosiddetto Document Object Model (DOM), lo standard fissato dal W3C per la rappresentazione dei documenti strutturati.

STRUTTURA AD ALBERTO DI UN DOCUMENTO

```
<html>
  <head>
    <title>Struttura del doc</title>
  </head>
  <body>
    <h1>Titolo</h1>
    <div>
      <p>Primo<a
href="pagina.htm">paragrafo</a>.</p>
    </div>
    <p>Secondo <em>paragrafo</em>.</p>
  </body>
</html>
```



GERARCHIA

- Il documento è una perfetta forma di gerarchia ordinata in cui tutti gli elementi hanno tra di loro una relazione del tipo genitore-figlio (parent-child in inglese). Ogni elemento è genitore e/o figlio di un altro.
- Un elemento si dice genitore (parent) quando contiene altri elementi. Si dice figlio (child) quando è racchiuso in un altro elemento. In base a queste semplici indicazioni possiamo analizzare il nostro documento.
- Ad esempio, <body> è figlio di <html>, ma è anche genitore di <h1>, e <p>.
  Quest'ultimo è a sua volta genitore di un elemento .
- Si potrebbe concludere che anche <body> sia in qualche modo genitore di . Non è esattamente così.
- Introduciamo ora un'altra distinzione, mutuata anch'essa dal linguaggio degli alberi genealogici, quella tra antenato (ingl: ancestor) e discendente (ingl: descandant).

GERARCHIA

- La relazione parent-child è valida solo se tra un elemento e l'altro si scende di un livello
- Come in un albero familiare si indica la relazione tra padre e figlio, pertanto possiamo dire che <head> è figlio di <html>, che <a> è figlio di <p>, etc. Tra e <a>, invece, si scende di due livelli: diciamo allora che è un antenato di <a> e che questo è rispetto al primo un discendente.
- L'albero del documento può essere letto non solo in senso verticale, ma anche orizzontale. In tal senso, gli elementi che sono posti sullo stesso livello, ovvero quelli che hanno lo stesso genitore, si dicono fratelli (ingl: siblings). Nel nostro esempio, h1, div e p sono fratelli rispetto all'elemento body.
- Infine, c'è un solo elemento che racchiude tutti e non è racchiuso: <html>. Continuando con la metafora familiare potremmo dire che è il capostipite, ma in termini tecnici si dice che esso è l'elemento radice (ingl: root).

CSS TIPOLOGIE

- le tre tipologie di dichiarazioni che è possibile inserire in un CSS:
  - regole
  - commenti
  - direttive @-rules

```
body {
 background: white;
 color: black;
}

/* Stili per i titoli h1 */
h1 {
 color: red;
 font: 36px Helvetica, Arial, sans-serif;
}

/* Colore del testo delle liste */
li {color: green;}

/* Colore dei titoli h1 per la stampa */
@media print {
 h1 {color: black;}
}
```

REGOLA CSS

- la selezione del codice precedente è composta da due blocchi principali:
  - il selettore;
  - il blocco delle dichiarazioni.
- Il selettore serve a definire la parte del documento cui verrà applicata la regola
- In questo caso, ad esempio, verranno formattati tutti gli elementi <h1>: lo sfondo sarà rosso, il colore del testo bianco
- I selettori possono essere di diverso tipo (verranno approfonditi ulteriormente, per il momento sia chiara la funzione che essi svolgono)

REGOLA CSS

- Il blocco delle dichiarazioni è delimitato rispetto al selettore e alle altre regole da due parentesi graffe, la prima di apertura e la seconda di chiusura
- Al suo interno possono trovare posto più dichiarazioni. Una dichiarazione è composta da una coppia:
  - proprietà;
  - valore.
- La **proprietà** definisce un aspetto dell'elemento/selettore da modificare (margini, colore di sfondo, larghezza, etc) secondo il valore espresso. Proprietà e valore devono essere separati dai **due punti**
- Le dichiarazioni vanno invece separate con un **punto e virgola**. Non è obbligatorio, ma è buona norma mettere il punto e virgola anche dopo l'ultima dichiarazione del blocco

RISPETTO DELLE REGOLE

- Una limitazione fondamentale da rispettare è: per ogni dichiarazione non è possibile indicare più di una proprietà, mentre è spesso possibile specificare più valori. Questa regola è pertanto errata:

```
body {color background: black;}
```

- Mentre questa è perfettamente valida e plausibile:

```
p {font: 12px Verdana, arial;}
```

- Gli spazi bianchi lasciati all'interno di una regola non influiscono sul risultato. Il consiglio, anzi, è di lasciare sempre uno spazio tra le varie parti per una migliore leggibilità.

PROPRIETÀ SINGOLE E SINTASSI ABBREVIATA

- Nelle definizione delle regole è possibile fare uso di **proprietà singole e proprietà a sintassi abbreviata.** Con questa espressione traduciamo il termine inglese **shorthand properties**, alla lettera, con il termine scorciatoie.
- Le proprietà singole sono la maggior parte. Con esse impostiamo, per un dato elemento o selettore, un singolo aspetto: il colore, la dimensione del testo, il font da utilizzare, etc.
- Con le shorthand properties è possibile invece definire con una sola dichiarazione un insieme di proprietà
- Ogni elemento presenta sui suoi quattro lati un certo margine rispetto a quelli adiacenti. È possibile definire per ciascuno di essi un valore usando quattro proprietà singole separate:
    - margin-top
    - margin-right
    - margin-bottom
    - margin-left

PROPRIETÀ SINGOLE E SINTASSI ABBREVIATA

- La regola sarebbe questa:

```
div {
 margin-top: 10px;
 margin-right: 5px;
 margin-bottom: 10px;
 margin-left: 5px;
}
```

- Lo stesso risultato si può ottenere usando la proprietà a sintassi abbreviata margin:

```
div {margin: 10px 5px 10px 5px;}
```

- Approfondiremo nel corso dell'analisi delle proprietà usi e costrutti sintattici di ciascuna, per il momento ci limitiamo all'elenco:
  - `background | border | border-top | border-right`
  - `border-bottom | border-left | border-width | border-color`
  - `list-style | margin | padding | outline`

COMMENTI NEL CSS

- Nello snippet di codice visto ad inizio lezione, le parti racchiuse tra i segni /* e */, rappresentano **commenti al codice**.

```
/* Stili per i titoli h1 */

/* Colore del testo delle liste */

/* Colore dei titoli h1 per la stampa */
```

- I commenti non sono interpretati dal browser. Sono utili nei CSS, come nei linguaggi di programmazione, per aggiungere annotazioni esplicative di vario tipo a beneficio di chi scrive e consulta il codice.

@-RULES NEL CSS

- Il terzo tipo di dichiarazione che è possibile inserire in un CSS è rappresentato dalla cosiddette direttive @-rules

- Il nome deriva dal fatto che questo particolare tipo di istruzione è contrassegnato nella sua definizione dal simbolo/prefisso @ (at):

```
@media print {
 h1 {color: black;}
}
```

- Basti per il momento questa nota sulla definizione sintattica

- 1. I valori di una proprietà non vanno mai messi tra virgolette. Le uniche eccezioni riguardano i valori espressi da stringhe di testo e i nomi dei font formati da più di una parola. Un esempio:

```
/* Valori espressi per la proprietà <code>content</code>
   con una stringa di testo   */
content: "Viva i CSS";

/* Definizione del font Times New Roman */
p {font-family: "Times New Roman", Georgia, serif;}
```

- 2. Quando si usano valori numerici con unità di misura, non bisogna lasciare spazio tra numero e sigla dell'unità. È corretto quindi scrivere 15px oppure 15em. È invece sbagliato usare 15 px o 15 em. In questi casi la regola sarà ignorata o mal interpretata.

TIPI DI VALORE

- Nei CSS i valori possono essere espressi da:
  - numeri, possono essere definiti come numeri interi (1, 23, 45, etc.) o in virgola mobile (1.2, 3.45, 4.90, etc.)
  - unità di misura
  - percentuali
  - codici per la definizione dei colori
  - URI
  - parole chiave (keywords)
  - stringhe di testo

```
/* Altezza di linea con un numero */
p {line-height: 1.2;}

/* Larghezza con unità di misura */
div {width: 300px;}

/* Larghezza in percentuale */
div {width: 60%;}

/* Colore con codice esadecimale */
body {background-color: #2795b6;}

/* URL per un'immagine di sfondo */
body {background-image: url(sfondo.jpg);}

/* Ripetizione dello sfondo con una keyword */
body {background-repeat: no-repeat;}

/* Stringa di testo   */
content: "Viva i CSS";
```

UNITÀ DI MISURA PER LE DIMENSIONI

Unità di misura	Descrizione
in (inches – pollici)	classica misura del sistema metrico americano. Praticamente nullo il suo valore su un supporto come un browser web
cm (centimetri)	stesso discorso visto per i pollici, la difficoltà maggiore sta nella resa su monitor, che è altra cosa rispetto alla carta stampata.
mm (millimetri)	vedi quanto detto per centimetri e pollici.
pt (points – punti)	unità di misura tipografica destinata essenzialmente a definire la dimensione dei font. Il suo utilizzo è di fatto limitato ai CSS per la stampa.
pc (picas)	unità poco usata. 1 pica equivale a 12 punti.
em (em-height)	unità di misura di ampio utilizzo se si desidera impostare le dimensioni dei font o dei box con un unità di misura relativa.
ex (ex-height)	di fatto inesistente il suo utilizzo. 1ex equivale all'altezza del carattere x minuscolo del font scelto.
px (pixels)	unità di misura ideale per gli schermi. È quella più usata e facile da comprendere.

CSS E I VALORI

- **Percentuale:** Un valore espresso in percentuale è da considerare sempre relativo rispetto ad un altro valore, in genere quello espresso per l'elemento parente. Si esprime con un valore numerico seguito (senza spazi) dal segno di percentuale: 60% è pertanto corretto, 60 % no.

- **Colori:** Sui diversi modi per esprimere il valore di un colore

```
body { color: red; }
h1 { color: #00ff00; }
p { color: rgb(0,0,255);}
```

- **Stringhe:** Alcune proprietà dei CSS possono avere come valore una stringa di testo da inserire come contenuto aggiunto nel documento. I valori espressi da stringhe vanno sempre racchiusi tra virgolette. Le proprietà in questione sono due: content e quotes.

CSS E I VALORI

- **Valori URI:** Si tratta di URL che puntano a documenti esterni (in genere immagini, come negli sfondi). Possono essere URL assoluti o relativi
  In questo caso il percorso fa sempre riferimento alla posizione del foglio di stile e non del documento HTML.
  La definizione dell'indirizzo è sempre introdotta dalla parola chiave url e va inserita tra parentesi tonde, con o senza virgolette. Queste possono essere singole o doppie. Un esempio:

```
background-image: url(sfondo.jpg);
background-image: url('sfondo.jpg');
background-image: url("sfondo.jpg");
```

CSS ESTERNI E INTERNI

- È esterno un foglio di stile definito in un file separato dal documento, si tratta di semplici documenti di testo modificabili anche con un editor di testo ai quali si assegna l'estensione .css.
- Un foglio di stile si dice invece interno quando il suo codice è compreso in quello del documento
- A seconda che si lavori con un CSS esterno o interno variano sintassi e modalità di inserimento. Rispetto a queste diverse modalità si parla di fogli di stile collegati, incorporati o in linea

COLLEGAMENTI

- Per caricare un foglio di stile esterno in un documento esistono due possibilità. La prima è quella che fa uso dell'elemento HTML <link>

- La dichiarazione va sempre collocata all'interno della sezione <head> del documento HTML:

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

ELEMENTO LINK

- L'elemento <link> presenta una serie di attributi di cui è importante spiegare significato e funzione:
  - **rel**: descrive il tipo di relazione tra il documento e il file collegato. È obbligatorio. Per i CSS due sono i valori possibili: stylesheet e alternate stylesheet.
  - **href**: serve a definire l'URL assoluto o relativo del foglio di stile. È obbligatorio
  - **type**: identifica il tipo di dati da collegare. Per i CSS il valore da usare è text/css. L'attributo non è più obbligatorio a partire dalla versione 5 del linguaggio HTML.
  - **media**: con questo attributo si identifica il supporto (schermo, stampa, etc.) cui applicare un particolare foglio di stile. È un attributo opzionale

@IMPORT DI UN CSS ESTERNO

- Un altro modo per caricare CSS esterni è usare la direttiva @import all'interno dell'elemento <style>:

```
<html>
 <head>
  <style>
   @import url(stile.css);
  </style>
 </head>
 <body>
 [...]
</html>
```

- Il CSS va collegato definendo un URL assoluto o relativo da racchiudere tra parentesi tonde (ma vedremo che altri modi sono accettati) e che la dichiarazione deve chiudersi con un punto e virgola

- Questa modalità di inserimento di CSS esterni è comunque ormai in disuso. Il metodo da prediligere è quello basato sull'elemento link.

FOGLI INCORPORATI

- I fogli incorporati sono quelli inseriti direttamente nel documento HTML tramite il tag <style>. Anche in questo caso la dichiarazione va posta all'interno della sezione <head>:

```
<html>
 <head>
  <style type="text/css">
   body {background: white;}
   p {color: black;}
   [...]
  </style>
 </head>
 <body>
 [...]
</html>
```

- Come si vede, la parte di codice che ci interessa inizia con l'apertura del tag <style>. Esso può avere due attributi:
    - type (opzionale);
    - media (opzionale).
- Per entrambi gli attributi valgono le osservazioni viste in precedenza. Seguono le regole del CSS e la chiusura di </style>.

CSS IN LINEA

- L'ultimo modo per formattare un elemento con i CSS consiste nell'uso dell'attributo HTML style. Esso fa parte della collezione di attributi HTML definiti globali: si tratta di quegli attributi applicabili a tutti gli elementi

- La dichiarazione avviene a livello dei singoli tag contenuti nella pagina e per questo si parla di fogli di stile in linea. La sintassi generica è la seguente:

```
<elemento style="regole_di_stile">
```

- Se, ad esempio, si vuole formattare un titolo h1 in modo che abbia il testo di colore rosso e lo sfondo nero, scriveremo:

```
<h1 style="color: red; background: black;">...</h1>
```

- Le cose da osservare nel codice sono due. Come valore di style si possono dichiarare più regole di stile. Esse vanno separate dal punto e virgola. I due punti si usano invece per introdurre il valore della proprietà da impostare. Esattamente come si fa con i CSS esterni e incorporati.

CONSIGLI

- A questo punto è giusto chiedersi: quando usare l'una o l'altra soluzione? Il punto di partenza nella risposta deve essere questo: i risultati nella formattazione del documento non cambiano
- L'uso estensivo di fogli in linea rischia di compromettere uno dei principali vantaggi dei CSS, ovvero avere pagine più leggere e facili da gestire. Intervenire nei meandri di una pagina per andare a modificare uno stile e ripetere l'operazione per quante sono le pagine del nostro sito può diventare davvero frustrante
- Il loro uso è considerato deprecato anche dal W3C. Nelle più moderne pratiche di sviluppo, l'uso di CSS in linea è quasi sempre limitato a regole inserite dinamicamente tramite Javascript.
- Dunque, usare sempre fogli di stili esterni. Ricorrere a quelli incorporati per particolari esigenze di formattazione su elementi specifici di una certa pagina. Rifuggire fin quando è possibile dall'uso di CSS in linea.

STYLE SU DIV E SPAN

- Con CSS si possono specificare regole di presentazione da associare a qualunque tag, talvolta si vorrebbero definire regole da applicare
- ad un'intera porzione del documento HTML (che potrebbero includere molti tag diversi),
- a un piccolo frammento di testo all'interno di un paragrafo (che potrebbe non essere delimitato da un tag)

STYLE SU DIV E SPAN

- Per rendere possibile la definizione di queste regole in HTML sono stati definiti i tag e
- Gli attributi id e class potranno poi essere usati per differenziare le varie occorrenze di e nel documento e applicarvi regole di stile diverse

STYLE SU DIV E SPAN

```
<html>
    <head><title>Prova div e span</title></head>
    <style type="text/css">
    div.titolo {
        text-align: center;
        color: red; }
    span#qp {
        background-color: yellow;
    }
    </style>
    <body>
        <div class="titolo">
            <h1>Titolo principale</h1>
            <h2>Sottotitolo</h2> </div>
        <div class="corpo">
            <p>Questo &egrave; il corpo del documento in cui evidenziamo
            <span id="qp">questa parte</span> tramite l'uso del tag
            span</p> </div>
    <body>
</html>
```

CSS COME USARLI

- selettore { proprietà1:valore1; proprietà2:valore2; . . . }
- Il selettore può avere una delle seguenti forme:
- **Selettore di tipo.** Un tag (senza parentesi <,>): body, p, table, div, span, . . .
- **Selettore di prossimità.** Hanno tre forme: E F, E>F e E+F, dove E ed F sono tag. Significano che la regola si applica agli elementi F che siano discendenti (ossia contenuti), figli diretti o immediatamente seguenti ad elementi E
- **Selettore di attributi.** E[attr] e E[attr=val]. Significano che la regola si applica agli elementi E in cui l'attributo attr è usato con qualunque valore o con valore specificato

CSS COME USARLI

- Il selettore può avere una delle seguenti forme:
- **Selettore di classe.** Può avere due forme E.val e E#val, che corrispondono a E[class="val"] e E[id=‘‘val"]
- Significano che la regola si applica a tutti gli elementi E in cui gli attributi class e id, rispettivamente, sono assegnati a val,
  Si può omettere E, e in questo caso la regola si applica a tutti gli elementi in cui class vale val, o all’elemento (unico) in cui id vale val, qualunque esso sia
- **Selettore di pseudo-classe.** Può avere una delle seguenti forme:
  - E:first-child si applica all’elemento che è primo figlio di E (per ogni occorrenza di E nel documento)
  - E:link e E:visited si applicano ai link non ancora o già visitato
  - E:hover, E:active e E:focus si applicano agli elementi solo quando vi passi sopra il mouse, vi si clicchi sopra o l’elemento `e selezionato

CSS COME USARLI

- Il selettore può avere una delle seguenti forme:
- **Selettore di pseudo-elementi.** Può avere una delle seguenti forme:
  - E:first-line si applica alla prima riga di testo contenuta all'interno di E
  - E:first-letter si applica alla prima lettera contenuta all'interno di E
  - E:before e E:after si possono usare per aggiungere qualcosa (un'immagine, un testo,....) immediatamente prima o dopo l'elemento E
- **Selettore universale.** Ha la forma * e si applica a tutti gli elementi del documento HTML
- **Raggruppamento di selettori.** Indicando più selettori separati da virgole si ottiene che la regola viene applicata a tutti i selettori specificati

CSS ESEMPIO

```
    <body>
        <h1>Ricetta: la <span class="piatto">pasta al pomodoro</span></
h1> <p>Vediamo ora come preparare un'ottima <span class="piatto">pasta
al pomodoro.</span> Innanzitutto, mettere a bollire in una pentola
capiente abbondante acqua.</p>
        <p>Tagliare a fettine sottili la <span class="ingr">cipolla</
span> e soffriggerla con l'<span class="ingr">olio</span> in una
padella fino a quando non sar&agrave; ben dorata. In seguito aggiungere
i <span class="ingr">pomodori</span> e il
        <span class="ingr">basilico</span>.</p>
        <p>Quando bolle l'acqua, salarla e versare gli <span
class="ingr"> spaghetti</span>. Una volta che gli spaghetti sono cotti,
scolarli e aggiungervi il sugo di pomodoro.</p>
    </body>
```

CSS ESEMPIO

```
<style type="text/css">
    /* Prima parte */
    body { background-color: #CCFF88;
           font-family: sans-serif; }
    h1 { color: red;
         text-align: center; }
    /* Seconda parte */
    span.ingr { background-color: yellow;}
    p:first-letter { font-size: 300%;
        float: left; }
    /* Terza parte */
    span.ingr, span.piatto { background-color: yellow; }
    span:hover { background-color: red; }
    p:first-letter { font-size: 300%;
        float: left; }
</style>
```

PROPRIETÀ DI BACKGROUND

- Le seguenti proprietà riguardano l'aspetto dello sfondo del documento:
- **background-color:** specifica il colore dello sfondo
- **background-image:** specifica l'immagine da usare come sfondo (di default ripetuta in orizzontale e verticale fino a riempire la finestra)
- **background-repeat:** specifica se l'immagine di sfondo deve essere ripetuta solo in orizzontale, solo in verticale o non deve essere ripetuta;
- **background-attachment:** specifica se l'immagine di sfondo deve avere posizione fissa o essere "scrollata" assieme alla pagina
- **background-position:** specifica la posizione in cui l'immagine di sfondo deve essere visualizzata

PROPRIETÀ DEL TESTO

- Le seguenti proprietà riguardano l'aspetto del testo contenuto in un tag del documento:
- **color:** specifica il colore del testo
- **text-align:** specifica l'allineamento del testo (sinistra, destra, centrato e giustificato)
- **text-decoration:** specifica testo decorato (sottolineato, barrato, lampeggiante, ecc...)
- **text-transform:** specifica se il testo deve essere visualizzato in maiuscolo o minuscolo
- **text-indent**: specifica l'indentazione della prima linea di testo Altre: line-height, vertical-align, word-spacing, direction

PROPRIETÀ DEI FONT

- I font da usare per il testo contenuto in un dato elemento del documento HTML possono essere descritti in modo generico usando un descrittore di classe generico (serif, sans-serif o monospace)
- Oppure possono essere descritti indicando esattamente il nome del font da usare (es. Times New Roman)
- Il font da usare si specifica tramite la proprietà font-family
- La proprietà font-family si aspetta una lista di descrittori di font (separati da virgole).
  - Il browser scandirà la lista fino a che non trova un font conosciuto;
  - Per questo spesso la lista consiste di tre elementi: un nome di font specifico, un nome di font alternativo e simile al primo, una classe di font generica
- NOTA: Se il nome del font contiene spazi, bisogna racchiuderlo tra virgolette!

PROPRIETÀ DEI FONT

- La dimensione del font può essere specificata in modo assoluto o relativo:
- **Una dimensione assoluta** può essere espressa in pixel: es. 12px
- **Una dimensione relativa** può essere espressa in vari modi:
  - **In percentuale**, rispetto alla dimensione del del contesto (elemento contenitore): es. 80%
  - **In maniera relativa rispetto alla dimensione del font impostata dal browser**, usando l'unità di misura em (di default 1em = 16px)
  - **In maniera simbolica** attraverso i valori small, medium, large, smaller, larger,.....
- I font con dimensione assoluta a volte (specie con vecchie versioni di IE) non vengono ridimensionati correttamente:
  - Una soluzione valida `e impostare il la dimensione per il tag body usando un valore percentuale, e le altre dimensioni usando em

PROPRIETÀ DEI FONT

- Le seguenti proprietà riguardano i font usati nel documento:
  - **font-family:** specifica (un elenco) di font da usare per un elemento
  - **font-style:** specifica se il testo è normale o corsivo
  - **font-size:** specifica la dimensione del font (assoluta o relativa)
  - **font-weight:** specifica "il peso" (lo spessore) del font con un valore numerico (da 100 a 700) o con parole chiave (e.g. bold)
  - **font-variant:** specifica se il testo deve essere visualizzato in minuscolo

PROPRIETÀ DELLE LISTE

- Esistono inoltre proprietà specifiche per le liste (numerate e non numerate) che consentono di impostare il simbolo (o l'immagine) da visualizzare a sinistra di ogni elemento di una lista non numerata, o di impostare il formato della numerazione degli elementi di una lista numerata.
- Queste proprietà (che non vediamo in dettaglio) sono: **list-style, list-style-image, list-style-position e list-style-type**
- Ad esempio: list-style:none; toglie il punto nero dagli elementi di una lista non numerata

IMMAGINE SFONDO RESPONSIVO

```
body {
    /* Posizione immagine di sfondo */
    background-image: url(images/background-photo.jpg);

    /* Immagine di sfondo è centrata verticalmente e orizzontalmente */
    background-position: center center;

    /* Lo sfondo non è ripetuto */
    background-repeat: no-repeat;

    /* L'immagine di sfondo è fissata e non si muove quando l'altezza del contenuto è più grande dell'altezza
dell'immagine */
    background-attachment: fixed;

    /* Rende la dimensione dell'immagine di sfondo ridimensionabile in base alla dimensione del contenitore
del body */
    background-size: cover;

    /* Il colore di sfondo viene mostrato mentre l'immagine di sfondo viene caricata */
    background-color: #464646;
}
```

IMMAGINE ARROTONDATA

```
.img-circle {
      border-radius: 50%;
  }


<img class="img-circle"
  src="kitten.jpg">
```

POSITION

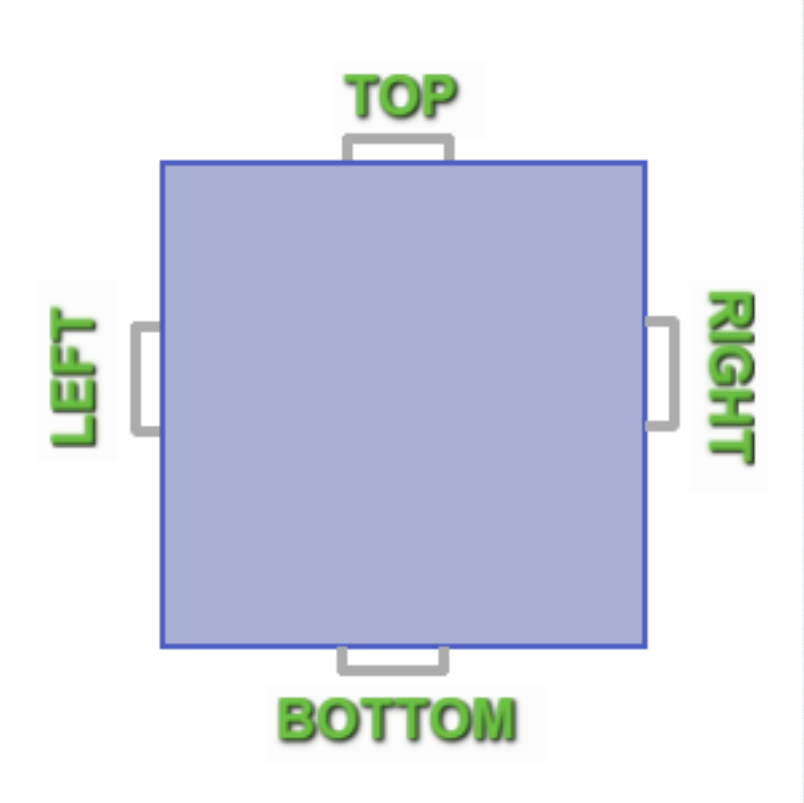
- position è la proprietà fondamentale per la gestione della posizione degli elementi: determina la modalità di presentazione di un elemento sulla pagina
- Si applica a tutti gli elementi e non è ereditata
- ***selettore {position: valore;}***

POSITION

- I valori con cui è possibile definire la modalità di posizionamento sono quattro:
  - **static;** È il valore di default, quello predefinito per tutti gli elementi non posizionati secondo un altro metodo. Rappresenta la **posizione normale** che ciascuno di essi occupa nel flusso del documento.
  - **relative;**
  - **absolute;**
  - **fixed;**

POSITION RELATIVE

- L'elemento viene posizionato relativamente al suo box contenitore ed è rappresentato dal posto che l'elemento avrebbe occupato nel normale flusso del documento
- La posizione viene impostata con le proprietà top, left, bottom o right. Nel posizionamento relativo esse non indicano un punto preciso, ma l'ammontare dello spostamento in senso orizzontale e verticale rispetto al box contenitore.
- Con il posizionamento relativo, si va a modificare la posizione naturale di un elemento traslandola attraverso l'uso di top, left, bottom o right.
- un elemento posizionato relativamente è come se lasciasse una sorta di segnaposto nella posizione in cui naturalmente dovrebbe essere.
- Ora è importante fissare un concetto essenziale, ovvero come il posizionamento relativo venga effettuato tramite le proprietà top, left, bottom o right



POSITION RELATIVE

- La metafora più semplice per comprendere il funzionamento è immaginare che sui quattro lati di un elemento posizionato relativamente ci siano quattro maniglie che si possono tirare o spingere: un valore positivo equivale a spingere, mentre un valore negativo equivale a tirare l'elemento.

- Per esempio, se viene specificato left: 30px significa che l'elemento viene spinto da sinistra di 30 pixel. Dichiarando invece top: -20px è come se tirassimo l'elemento dal suo lato superiore, con il conseguente effetto di traslarlo verso l'alto di 20 pixel.

- Con il posizionamento relativo andrebbero specificati al massimo uno o due valori tra top, left, bottom e right.

POSITION ABSOLUTE

- L'elemento, o meglio, il box dell'elemento, viene rimosso dal flusso del documento ed è posizionato in base ai valori forniti con le proprietà top, left, bottom o right.
- Il posizionamento assoluto (position: absolute;) avviene sempre rispetto al box contenitore dell'elemento. Questo è rappresentato dal primo elemento antenato (ancestor) che abbia un posizionamento diverso da static.
- Se tale elemento non esiste il posizionamento assoluto avviene in base all'elemento radice html, che in condizioni standard coincide con l'area del browser che contiene il documento e che ha inizio dall'angolo superiore sinistro di tale area
- Un elemento posizionato in modo assoluto scorre insieme al resto del documento.

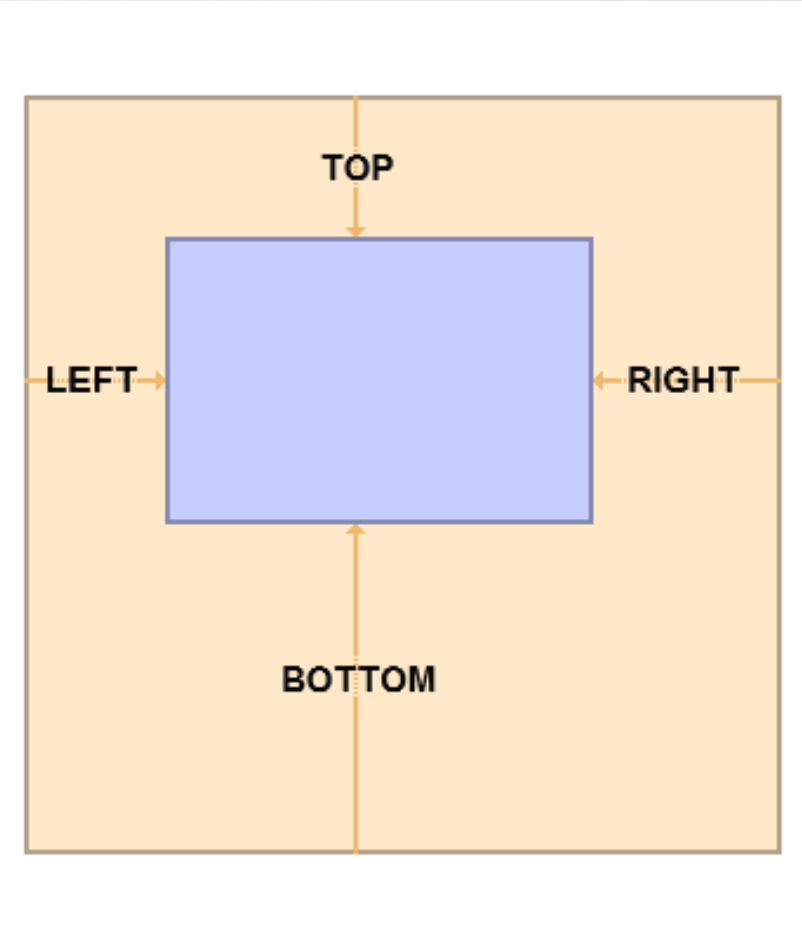
POSITION ABSOLUTE

- Il modo pi semplice per posizionare assolutamente un elemento rispetto ad un altro, consiste nel dichiarare per quest'ultimo un posizionamento relativo senza coordinate e nel dichiarare `position: absolute` per il primo con le coordinate

```
#box-1 {position: relative;}
#box-2 {
  position: absolute;
  top: 0;
  left: 20px
}
```

POSITION ABSOLUTE

- Un elemento posizionato assolutamente è di default reso block-level, indipendentemente dalla sua natura iniziale
- È come se, insieme a position: absolute dichiarassimo ogni volta implicitamente display: block
- Il fatto che tutti gli elementi posizionati assolutamente sono promossi a elementi block-level significa in sostanza che è possibile attribuire loro dimensioni esplicite
- Non solo è possibile, ma è consigliabile impostare quantomeno una larghezza.
- Per quanto riguarda le proprietà top, left, bottom o right, si potrebbe pensare che questi valori siano da intendersi come delle coordinate. In realtà non è proprio così, ed è più facile pensare i valori che queste proprietà possono assumere come vere distanze.



POSITION ABSOLUTE

- Nell'immagine sopra, il riquadro rosa è il blocco contenitore dell'elemento posizionato assolutamente, in azzurro. Vengono evidenziate visivamente le quattro proprietà, che rappresentano ciascuna la distanza tra i due lati concordi dell'elemento posizionato assolutamente e il suo blocco contenitore

- Per esempio, top rappresenta la distanza tra il lato superiore dell'elemento posizionato assolutamente e quello del suo blocco contenitore.

- Per posizionare assolutamente un elemento, è possibile specificare da una a quattro tra le proprietà top, right, bottom e left. Tipicamente, però, se ne dichiarano due (o una sola): tre o quattro valori hanno l'effetto di dare conflitti o modificare le dimensioni dell'elemento posizionato assolutamente.

POSITION FIXED

- Usando questo valore, il box dell'elemento viene, come per absolute, sottratto al normale flusso del documento
- La differenza sta nel fatto che per fixed il box contenitore è sempre la cosiddetta viewport
- Con questo termine si intende la finestra principale del browser, ovvero l'area del contenuto
- Altra differenza fondamentale: un box posizionato con fixed non scorre con il resto del documento. Rimane, appunto, fisso al suo posto

```
/* definisco il contenitore */
div.contenitore {
  position: relative;
  width: 400px;
  height: 200px;
}
/* definisco alcune caratteristiche comuni a tutti
e tre i DIV presenti nel contenitore */
.contenitore > div {
  position: absolute;
  width: 100px;
  height: 100px;
}
/* posiziono il box N.1 */
div.box1 {
  top: 0px;
  left: 0px;
}
/* posiziono il box N.2 */
div.box2 {
  bottom: 5px;
  left: 10px;
}
/* posiziono il box N.3 */
div.box3 {
  top: 30%;
  left: -25px;
}
```

DISPLAY

- Questa proprietà serve per controllare con i CSS il modo in cui un elemento viene reso dal browser.
- Ogni elemento HTML ha un suo comportamento predefinito rispetto alla presentazione: si parla di elementi blocco, elementi in linea, elementi di lista e elementi di tabella.
- Ebbene, con la proprietà display possiamo modificare, quando e se necessario, questo comportamento di default. La proprietà display è ereditata

DISPLAY

- **block:** l'elemento viene reso come un elemento blocco
- **inline:** l'elemento a cui viene applicata assume le caratteristiche degli elementi inline
- **inline-block:** l'elemento può assumere, come gli elementi blocco, dimensioni esplicite (larghezza e altezza), margini e padding, ma come tutti gli elementi inline, si disporrà orizzontalmente e non verticalmente, potendo essere circondato dal testo ed essendo sensibile all'allineamento verticale
- **none:** l'elemento non viene mostrato; o meglio: è come se non fosse nemmeno presente nel documento, in quanto non genera alcun box; l'uso del valore none è uno dei mezzi con cui, nei CSS, si può nascondere un elemento

VISIBILITY

- Per certi versi strettamene connessa alla proprietà display è la proprietà visibility
- Attraverso questa proprietà viene gestita, come è facile intuire, la visibilità dell'elemento, consente di creare elementi invisibili all'interno della pagina.
- Questa proprietà ammette due soli valori:
  - **visible** - (default) l'elemento è visibile;
  - **hidden** - l'elemento è invisibile.
- A questo punto, però, si rende necessaria una precisazione: gli elementi nascosti mediante visibility: hidden, a differenza di quelli cui è stato applicato display: none, sono presenti nella pagina ma semplicemente non vengono visualizzati
- Questo significa che l'ingombro dell'elemento non muta, semplicemente al suo posto compare uno spazio vuoto avente le stesse dimensioni dell'elemento nascosto.

CLEAR E FLOAT: LAYOUT TABLE-LESS

- Queste due proprietà (che non vengono ereditate dagli elementi discendenti) assumono un ruolo assolutamente centrale nell'attività quotidiana di web design in quanto sono le protagoniste indiscusse di buona parte dei nuovi layout table-less.
- La tecnica che prevede l'utilizzo di queste proprietà è detta table-less ("senza tabelle") per contrapporla alla ormai obsoleta abitudine di molti web-designer di impostare il layout di un sito web utilizzando le tabelle di HTML.
- Fino a qualche anno fa, infatti, era abitudine diffusa racchiudere i vari elementi dei template web all'interno di tabelle annidiate attraverso le quali veniva definita la griglia dei contenuti. Una simile prassi, tuttavia, è da considerarsi deprecabile in quanto la finalità delle tabelle di HTML non è certamene quella di fungere da griglia per le pagine web, ma piuttosto quella di essere la base strutturale per la creazione di rapporti ed elenchi di dati.
- Ogni web designer che si rispetti, quindi, deve essere oggi in grado di creare layout (anche estremamente complessi) utilizzando esclusivamente elementi contenitore come DIV e liste opportunamente stilizzati mediante i fogli di stile. Vediamo quindi le caratteristiche di queste due proprietà, partendo da float.

FLOAT

- Questa proprietà ha lo scopo di estrarre un elemento dal normale flusso del documento per spostarlo su uno dei lati del suo contenitore; così facendo gli altri elementi della pagina non andranno a capo ma gli "scivoleranno" attorno posizionandosi sul lato opposto a quello dichiarato nella proprietà float.
- Questa proprietà, come avrete già intuito, ammette solo tre valori:
  - **none** - (default) l'elemento mantiene il suo posizionamento naturale;
  - **left** - l'elemento viene spostato a sinistra e gli altri contenuti della pagina gli scorrono a destra;
  - **right** - l'elemento viene spostato a destra e gli altri contenuti della pagina gli scorrono a sinistra.

FLOAT

- Un elemento flottante è sempre (ed automaticamente) un elemento di tipo block e pertanto è idoneo a possedere dimensioni esplicite, margini, padding e bordi
- Se all'elemento non vengono assegnati valori espliciti per le proprietà width ed height, le sue dimensioni saranno date dall'ingombro del suo contenuto sia in larghezza che in altezza (ovviamente non potrà essere superata la larghezza massima del suo contenitore).
- Un "problema" tipico di quanto si lavora con float è dato dall'atteggiamento del contenitore: come gli elementi posizionati in modo assoluto, anche quelli flottanti (essendo estratti dal normale flusso della pagina) perdono consistenza nei confronti del loro contenitore il quale, in un certo senso, non ne rileva la presenza.
- Al fine di risolvere tale problema è possibile:
    - impostare un altezza esplicita per il contenitore che sia almeno uguale a quella degli elementi flottanti in esso contenuti;
    - oppure utilizzare la proprietà clear per interrompere l'affiancamento e "ridare consistenza" al contenuto: in questo modo il contenitore si adatterà automaticamente all'altezza degli elementi flottanti e non sarà necessario assegnargli un'altezza arbitraria.

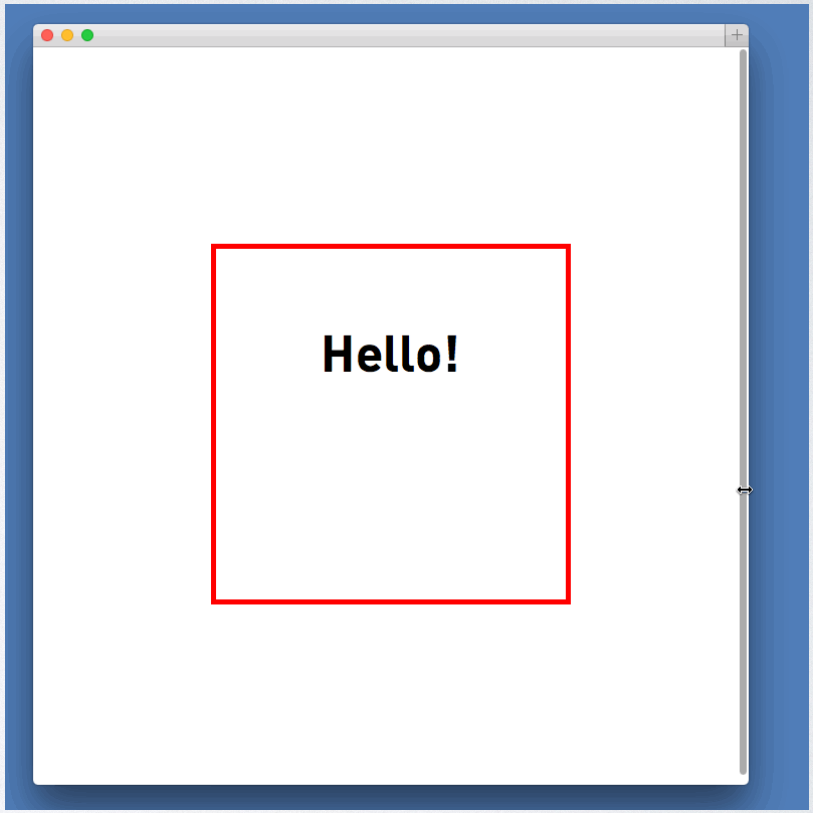
CLEAR

- La proprietà clear accompagna sempre la proprietà float: attraverso clear, infatti, si vuole impedire che ad un dato elemento si affianchino elementi flottanti su un lato o su entrambi.
- tale proprietà ammette i seguenti valori:
  - **none** - (default) gli elementi con float possono circondare l'elemento sia a destra che a sinistra;
  - **left** - l'elemento non può avere elementi flottanti affiancati a sinistra;
  - **right** - l'elemento non può avere elementi flottanti affiancati a destra;
  - **both** - l'elemento non può avere elementi flottanti affiancati né a destra né a sinistra;
- Tip: nei layout table-less vengono spesso utilizzati dei DIV vuoti a cui viene assegnata la proprietà clear: both al solo fine di "chiudere" un gruppo di elementi affiancati.

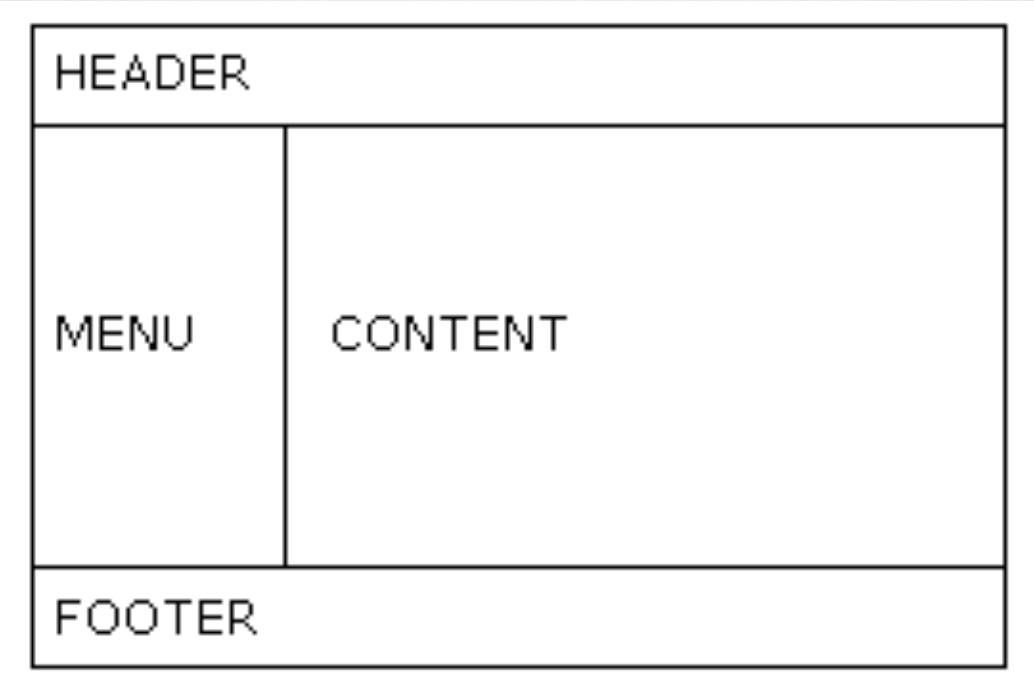
DIV RESPONSIVE

```
.square {
  position: relative;
  width: 50%;
}
.square:after {
  content: "";
  display: block;
  padding-bottom: 100%;
}
.content {
  position: absolute;
  width: 100%;
  height: 100%;
}

<div class="square">
  <div class="content">
    Hello!
  </div>
</div>
```



```
<style>
    * {
        box-sizing: border-box;
    }
.row::after {
    content: "";
    clear: both;
    display: table;
}
[class*="col-"] {
    float: left;
    padding: 15px;
}
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
.menu ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
.menu li {
    padding: 8px;
    margin-bottom: 7px;
    background-color: #33b5e5;
    color: #ffffff;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
    background-color: #0099cc;
}
</style>
```


HEADER
MENU
CONTENT
FOOTER

```html
<html>
  <head>
    <title>Layout table-less con i CSS</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <div id="MAIN">
      <div id="HEADER">logo...</div>
        <div id="MENU">
          menu laterale...
        </div>
        <div id="CONTENT">
          contenuto della pagina...
        </div>
        <div id="FOOTER">
          copyright e info di chiusura pagina...
        </div>
     </div>
  </body>
</html>
```

```
body{
    background: #FFFFFF;
    color: #000000;
    margin: 0;
}
div{
    font-size: 11px;
    font-family: verdana;
}
#MAIN{
    margin: 0 auto;
    width: 750px;
    border: solid 1px #000000;
}
#HEADER{
    border-bottom: solid 1px #000000;
}
#MENU{
    float: left;
    width: 149px;
}
#CONTENT{
    float: left;
    width: 600px;
    border-left: solid 1px #000000;
}
#FOOTER{
    clear: left;
    border-top: solid 1px #000000;
}
```

Dott. Antonio Giovanni Lezzi

PROGRAMMA DELLA GIORNATA

- Comunicazione protocollare in Internet
- Richieste e Mime
- Richieste al server
- Form
- Input con differenti Type
- Textarea e Select/Option
- Introduzione a HTML5

COMUNICAZIONE INTERNET

- I computer in Internet comunicano scambiandosi pacchetti di dati, chiamati pacchetti IP (Internet Protocol)
- Questi pacchetti partono da un computer, attraversano vari nodi della rete (Server di rete) e arrivano a destinazione
- Per stabilire il percorso intermedio tra i due computer che vogliono comunicare si esegue un algoritmo di routing (ne esistono di vari tipi, a seconda delle esigenze e a seconda del tipo di rete).

RETE DI CALCOLATORI

- Una rete di calcolatori è un sistema che permette la condivisione di dati informativi e risorse (sia hardware sia software) tra diversi calcolatori.

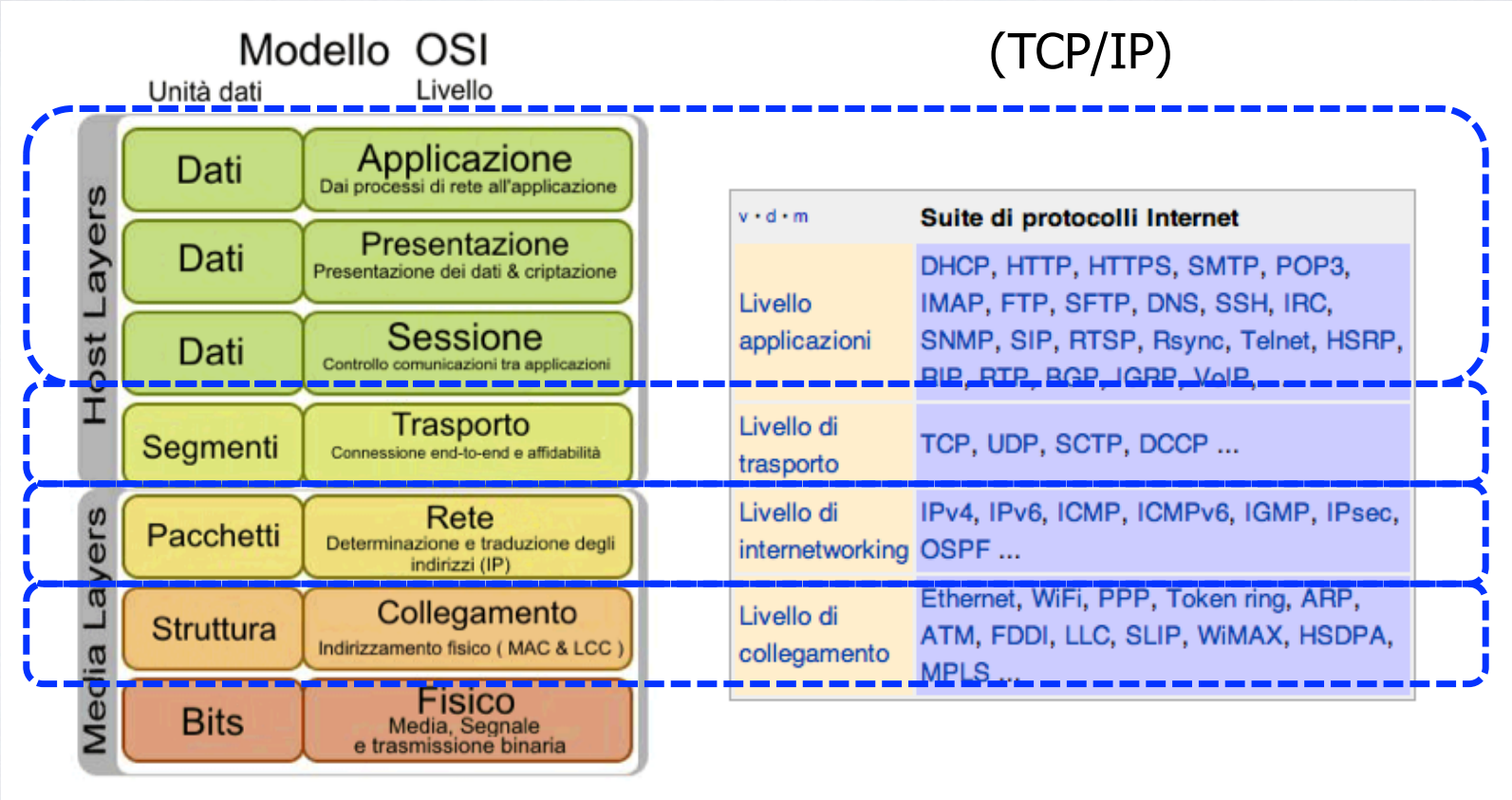
- Lo scopo è di fornire dei servizi aggiuntivi per gli utenti: Condivisione di informazioni e risorse.

- Può essere privata o pubblica:
  Internet è la più grande rete costituita da miliardi di dispositivi in tutto il mondo.

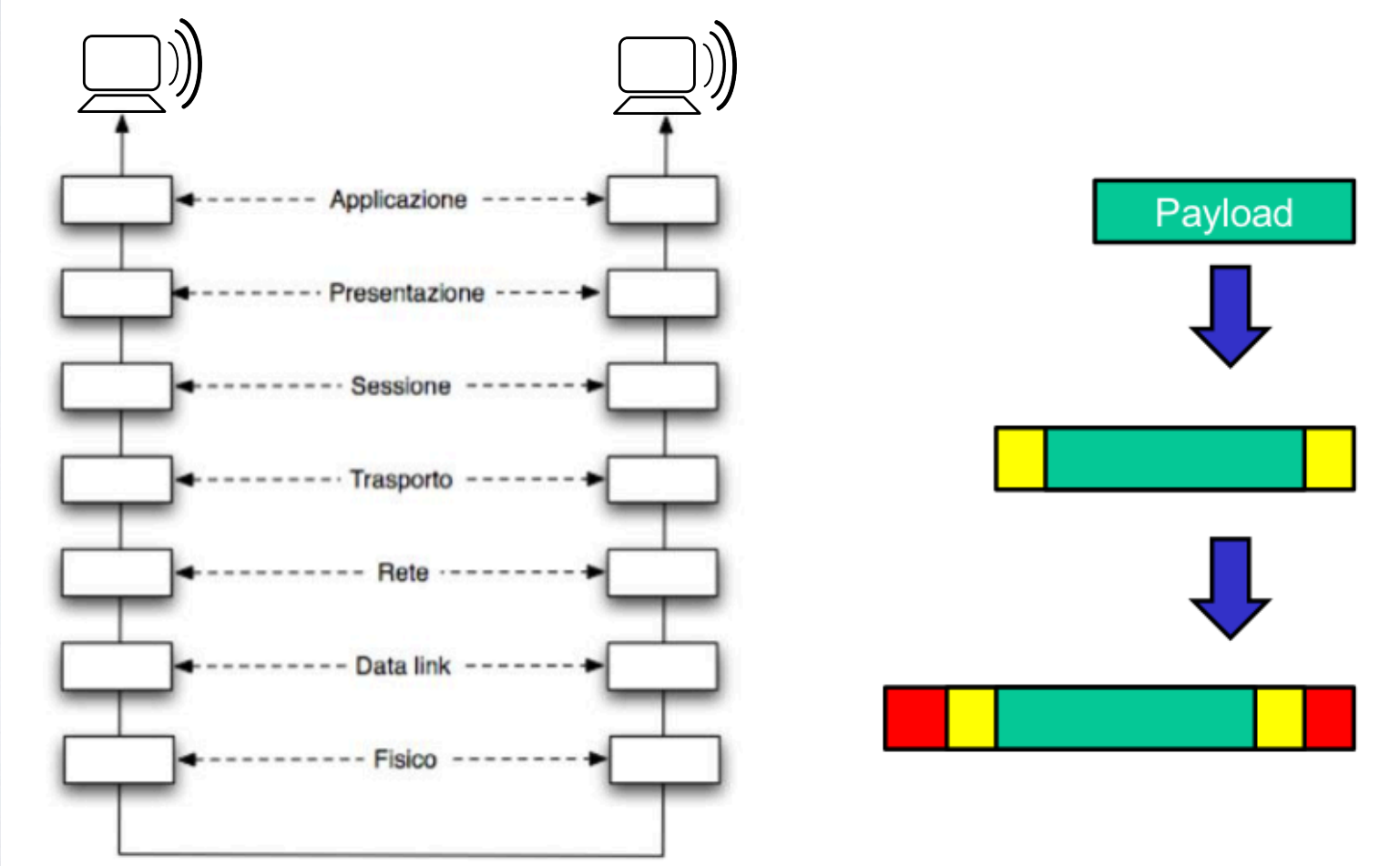
COMUNICAZIONE

- All'interno di una rete di calcolatori la comunicazione avviene tramite scambio di informazioni (messaggi)
- Per comunicare sono necessari:
  – Un canale fisico;
  – Un linguaggio comune: protocollo di comunicazione.
- Canale fisico: Cavo telefonico, fibra ottica, onde radio, ...
- Protocollo: Insieme di regole formalmente descritte, definite al fine di favorire la comunicazione tra una o più entità.

PILA PROTOCOLLARE



COMUNICAZIONE A LIVELLI



PARADIGMI DI COMUNICAZIONE

- **Client-Server**: Applicazioni di rete formate da due programmi distinti che possono essere in esecuzione in due elaboratori (host) diversi: un server e un client
  - **Server**: si mette in attesa di una richiesta da servire.
  - **Client**: effettua tale richiesta.
  - Tipicamente il client comunica con un solo server, mentre un server solitamente comunica con più client contemporaneamente.
- **Peer-to-peer**: Più host diversi comunicano tra loro comportandosi sia come client che come server.

NETWORKING PER LE APPLICAZIONI

- Le applicazioni più diffuse per utilizzare le funzionalità di rete si appoggiano direttamente sui protocolli TCP o UDP della suite di protocolli Internet.
  – I livelli (protocolli) sottostanti sono mascherati.

- Alcuni concetti chiave:

  - Hosts

  - Internet Addresses

  - Ports

  - Protocols

HOSTS

- i dispositivi connessi a Internet prendono il nome di *hosts*
- Molti hosts sono computer, ma possono anche esser considerati host router, printer, fax, macchina espresso, ecc

INDIRIZZO IP

- Per stabilire il percorso quindi: il mittente di una comunicazione, i destinatari, i nodi intermedi, occorre che ogni computer collegato in rete abbia un nome che lo identifica univocamente, questo nome, è un numero, e si chiama indirizzo IP.

INDIRIZZI INTERNET

- Ogni host su Internet è identificato con un unico indirizzo costituito da 4 byte definito come indirizzo: Internet Protocol cioè IP address.
- Questo indirizzo è scritto in formato di quattro cifre puntate (*dotted quad* format) come ad esempio 199.1.32.90 dove ogni byte è un intero senza segno compreso tra 0 e 255
- Ci sono circa 4 miliardi possibili indirizzi IP ma non sono allocati/assegnati efficientemente

DOMAIN NAME SYSTEM (DNS)

- Gli indirizzi numerici sono mappati con dei nomi come *www.apple.com* o *developer.apple.com* dal DNS
- Ogni sito ha un suo software domain name server software (server) che traduce i nomi in indirizzi IP e viceversa
- DNS è un sistema distribuito

PROTOCOLLO

- Un insieme di regole formalmente descritte, definite al fine di favorire la comunicazione tra una o più entità
- Tutte queste regole sono definite mediante specifici protocolli, dalle tipologie più varie e ciascuno con precisi compiti/finalità, a seconda delle entità interessate e il mezzo di comunicazione
- Se le due entità sono remote, si parla di protocollo di rete

ESEMPIO PROTOCOLLO

Determinati protocolli sono gestiti da organismi quali il World Wide Web Consortium (W3C)che rispettano gli standard ISO/OSI.

Di seguito si citano i protocolli più utilizzati.

- HTTP e HTTPS - principale sistema per la trasmissione di informazioni sul web.
- Transmission Control Protocol (TCP) è uno dei principali protocolli Internet su cui si appoggiano gran parte delle applicazioni web è usato di solito in combinazione con il protocollo IP
- User Datagram Protocol (UDP) è usato di solito in combinazione con il protocollo IP.
- Internet Protocol (IP) - un protocollo di rete a pacchetto;

Esistono tuttavia anche protocolli non standardizzati ovvero di tipo proprietario, tipicamente adibiti a funzionalità di tecnologie altrettanto proprietarie.

TCP (TRANSFERT CONTROL PROTOCOL)

- È un protocollo di alto livello:
  - È orientato alla connessione;
  - Esegue controlli sugli errori, congestione e flusso di comunicazione

- Un destinatario TCP è identificato da un indirizzo IP (Internet Protocol) e da una porta di destinazione.

  - L'indirizzo IP identifica la macchina alla quale vogliamo collegarci.

  - Il numero della porta identifica l'applicativo (servizio) al quale vogliamo connetterci.

- Esempio: Un server web all'indirizzo 131.114.11.100 può offrire un servizio di posta elettronica e un servizio http. Il primo risponde sulla porta 110 e il secondo alla porta 80.

URL

- Un URL o brevemente (Uniform Resource Locator) è un metodo per identificare univocamente la locazione della risorsa su internet
- Una risorsa web può essere qualsiasi cosa, una directory, un file, un oggetto in rete come ad esempio una interfaccia per fare delle query ad un database remoto, o per un motore di ricerca.
- Tramite un URL (Uniform Resource Locator) è possibile riferirsi alle risorse di Internet in modo semplice e uniforme. Si ha così a disposizione una forma intelligente e pratica per identificare o indirizzare in modo univoco le informazioni su Internet
- In alcuni web browser, l'http:// davanti all'indirizzo è indispensabile, perché individua il protocollo dell'URL, protocollo che il Navigator e l'Explorer intuiscono anche se omesso.

FORMATO DI UN URL

Un URL consiste di 4 componenti:

1. il protocollo separato dal resto dai due punti (esempi tipici di protocolli sono http, ftp, news, file, ecc.);
2. il nome dell'host, o l'indirizzo IP dell'host, che è delimitato sulla sinistra da due barre (//), e sulla destra da una sola (/), oppure da due punti (:)
3. il numero di porta, separato dal nome dell'host sulla sinistra dai due punti, e sulla destra da una singola barra. Tale componente è opzionale, in quanto, come già detto, ogni protocollo ha una porta di default
4. il percorso effettivo della risorsa che richiediamo. Il percorso viene specifica- to come si specifica un path sotto Unix. Se non viene specificato nessun file, la maggior parte dei server HTTP aggiunge automaticamente come file di default index.html.

ESEMPI DI URL

```
http://java.sun.com/
file:///Macintosh%20HD/Java/Docs/JDK%201.1.1%20docs/api/
  java.net.InetAddress.html#_top_
http://www.macintouch.com:80/newsrecent.shtml
ftp://ftp.info.apple.com/pub/
mailto:elharo@metalab.unc.edu
telnet://utopia.poly.edu
ftp://mp3:mp3@138.247.121.61:21000/c%3a/stuff/mp3/

http://elharo@java.oreilly.com/
http://metalab.unc.edu/nywc/comps.phtml?category=Choral+Works
```

MESSAGGI HTTP

- I web server permettono di ottenere informazioni come risultato di una query (interrogazione). Invece di richiedere un normale documento, si specifica nell'URL il nome di un programma, passandogli alcuni parametri che rappresentano la query vera e propria.

- In effetti tramite i programmi Browser viene eseguito l'HTML lato client come applicazione front end. Con lo sviluppo lato backend è possibile implementare dei servizi di connessione al database per fornire dei dati da utilizzare e creare pagine dinamiche.

- Una query ad un particolare servizio è costituita quindi da un normale URL, con in coda alcuni parametri. La parte dell'URL che specifica i parametri inizia con un punto interrogativo (?). Ogni parametro è separato da una "e commerciale" (&), e i valori che si assegnano ai parametri sono specificati in questo modo: nome = valore (il valore è facoltativo).

MIME

- MIME è un acronimo di "Multipurpose Internet Mail Extensions".
- inizialmente utilizzato per inviare messaggi di email ma adottato in seguito anche per HTTP
- Quando un browser invia una richiesta al server web, si invia anche un header MIME

HEADER DI RICHIESTE HTTP

```
GET /javafaq/images/cup.gif HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (Macintosh; I; PPC)
Host: www.oreilly.com:80
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
```

RISPOSTA DEL SERVER

- Quando un server web risponde a una richiesta di un browser un header ed un messaggio MIME viene inviato, ecco un esempio di risposta:

```
HTTP/1.0 200 OK
Server: Netscape-Enterprise/2.01
Date: Sat, 02 Aug 1997 07:52:46 GMT
Accept-ranges: bytes
Last-modified: Tue, 29 Jul 1997 15:06:46 GMT
Content-length: 2810
Content-type: text/html
```

RISPOSTA COMPLETA

```
GET javafaq/images/index.html HTTP/1.0

HTTP/1.1 302 Moved Temporarily
Date: Mon, 04 Aug 1997 14:21:27 GMT
Server: Apache/1.2b7
Location: http://www.java.com/javafaq/images/index.html
Connection: close
Content-type: text/html

<HTML><HEAD>
<TITLE>302 Moved Temporarily</TITLE>
</HEAD><BODY>
<H1>Moved Temporarily</H1>
The document has moved <A HREF="http://www.macfaq.com/macfaq/index.html">here</A>.<P>
</BODY></HTML>
```

USER AGENT

- Lo user agent è un'applicazione installata sul computer dell'utente che si connette ad un processo server
- Esempi di user agent sono i browser web, i lettori multimediali e i programmi client (Mail User Agent) come Outlook
- Oggi il termine è utilizzato soprattutto in riferimento ai client che accedono al World Wide Web
- Oltre ai browser, gli user agent del web possono essere i crawler dei motori di ricerca, i telefoni cellulari, i lettori di schermo e i browser braille usati da persone non vedenti

USER AGENT

- Quando gli utenti di Internet visitano un sito web, una stringa di testo è solitamente inviata per fare identificare al server lo user agent
- Questo fa parte della richiesta HTTP, con prefisso "User-agent:" o "User-Agent:" e tipicamente include informazioni come il nome dell'applicazione client, la versione, il sistema operativo e la lingua

- *3.01 su Windows 98 - User-Agent: Mozilla/2.0 (compatible; MSIE 3.01;Windows 98)*
- *5.0 su SunOS - User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; SunOS 5.9 sun4u; X11)*
- *5.1.7 su Mac OS 9 - User-Agent:Mozilla/4.0 (compatible; MSIE 5.17; Mac_PowerPC)*
- *5.2.3 su macOS - User-Agent:Mozilla/4.0 (compatible; MSIE 5.23; Mac_PowerPC)*

SERVIZI WEB E RICHIESTE REST

- I servizi web sono essenzialmente funzionalità rese disponibili in rete da servizi remoti dai quali, client distribuiti nel mondo possono recuperare informazioni, richiedere elaborazioni e altro
- Le richieste sul protocollo HTTP possono esser di diverso tipo e per il servizio REST avremmo:
- GET: per leggere dati da remoto senza apportare modifiche
- PUT: per richiedere la modifica dei dati pre-esistenti
- POST: per inviare dati verso il servizio con lo scopo di richiederne l'inserimento nella base dati
- DELETE: per cancellare il dato

CODICI DI STATO

- i codici di stato contenuti nella risposta e forniti dal server dobbiamo sapere:
- 200: OK
- 400: Bad Request
- 403: Forbidden
- 404: Not found
- 500: Internal Server error

FORM

- Il WEB è utilizzato soprattutto per visualizzare informazioni, ma attraverso di esso è possibile anche raccogliere informazioni e dati
- I moduli sono elementi che permettono un buon grado di interazione con l'utente e si rivelano particolarmente utili per la raccolta di informazioni.
- Un form normalmente è composto da vari tipi di controlli che consentono di inserire testo, selezionare delle opzioni ed effettuare altre operazioni disposte dal WebMaster
- L'utente conferma le sue scelte effettuando l'invio dei dati immessi, premendo l'apposito bottone a cui normalmente si da come etichetta Submit o Invia

FORM

- In seguito a questa operazione viene eseguito un programma residente sul server e a questo vengono passati i dati inseriti; il programma elabora i dati immessi ed effettua alcune operazioni per le quali è stato realizzato (ad esempio la visualizzazione di una pagina, l'invio di una E-Mail o altro).
- I programmi sono scritti solitamente in Perl o in C, ed utilizzano l'interfaccia CGI (Common Gateway Interface) per ricevere i dati dalla pagina web
- I metodi per passare i dati allo script sono sostanzialmente due: tramite una richiesta POST o tramite una richiesta GET; se non diversamente specificato è utilizzata la seconda possibilità

FORM

- Tramite il metodo GET i dati vengono passati allo script tramite l'URL della richiesta e risultano visualizzati nella barra degli indirizzi; in tale evenienza i parametri sono assegnati alla variabile speciale di ambiente dello script QUERY_STRING. Questa deve essere analizzata dallo script e solitamente non può contenere più di 255 caratteri.

- La variabile QUERY_STRING è separata dall'URL dello script (contenuto nella variabile SCRIPT_NAME) da un segno di punto interrogativo '?', mentre i parametri in essa contenuti sono indicati nella forma "nome=valore"e separati dal segno di ampersen '&'.

FORM

- Il metodo POST è invece molto più versatile perché non pone limiti alla dimensione dei dati passati allo script, in quanto questi vengono infatti inviati attraverso un canale di comunicazione separato (lo standard output)
- Questo metodo si sta diffondendo sempre di più ed è attualmente quello più utilizzato
- Con tale metodo per segnalare la fine dell'input per lo script viene utilizzata la variabile CONTENT_LENGTH che indica la dimensione in byte del flusso di dati
- Altra informazione passata tramite lo stream di output è una descrizione del tipo di dati, questa viene copiata nella variabile di ambiente CONTENT_TYPE

FORM

- Il tag che permette l'introduzione di un modulo è <FORM>, e tra esso ed il suo tag di chiusura può essere inserito qualunque contenuto HTML valido, tranne altri tag FORM
- Riconosce due attributi: ACTION="url" e METHOD="get | post''
- L'attributo ACTION è un po' come l'attributo SRC per le immagini; specifica l'indirizzo del programma che dovrà ricevere i dati immessi nel modulo
- METHOD specifica invece il modo in cui questi parametri vanno spediti al programma cgi, mediante l'url oppure mediante lo standard output

FORM

- La sintassi corretta del tag è la seguente:

```
<FORM ACTION="www.server.com/directory/programma.cgi" METHOD="post">
.... (modulo)
</FORM>
```

- I moduli vengono costruiti specificando i campi che devono contenere, ad esempio degli spazi in cui scrivere, dei pulsanti da cliccare, dei bottoni da selezionare etc... opportunamente disposti sullo schermo per dare una visualizzazione compatta e gradevole

- Verrà realizzato mediante appositi tag, che sono: <INPUT>, <SELECT> e <TEXTAREA>

FORM: RACCOGLIERE INFO

- Abbiamo visto come introdurre un modulo in una pagina web; però un modulo così introdotto non serve a niente, non è neanche in grado di visualizzare qualcosa sullo schermo, occorre inserire degli elementi adatti a raccogliere le informazioni inserite dall'utente.
- Questi elementi possono essere campi in cui l'utente possa scrivere, pulsanti da selezionare, bottoni da cliccare o altri elementi su cui effettuare una scelta.
- Il tag principale e quello più utilizzato è <INPUT> che specifica un dato in ingresso per il modulo
- Il tag <INPUT> riceve alcuni attributi che indicano la natura della informazione introdotta
- L'attributo NAME="testo" specifica l'identificativo di quell'elemento introdotto, in pratica il nome della variabile a cui l'utente fornisce il valore.

FORM: RACCOGLIERE INFO

- Infine l'attributo TYPE="tipo" che indica il tipo di informazione da raccogliere. Può assumere i valori
  - “text"
  - “password"
  - “hidden"
  - “checkbox"
  - “radio"
  - “submit"
  - “reset"
  - "image"

FORM: RACCOGLIERE INFO

- . Esaminiamo in dettaglio i valori di questo tipo di dato:
  - TYPE="text" è utilizzato per ricevere del testo come input dall'utente, infatti esso visualizza una casella in cui scrivere; esso utilizza anche alcuni attributi supplementari
    - MAXLENGHT="n" indica il numero massimo di caratteri che è possibile inserire (se non specificato, non viene posto alcun limite)
    - SIZE="n" indica in caratteri la dimensione della casella di testo
    - VALUE="testo" indica il valore predefinito iniziale a cui è impostata la variabile, se l'utente non scrive nulla sarà questo ad essere trasmesso

FORM: RACCOGLIERE INFO

- TYPE="password" si comporta allo stesso identico modo di "text" con la sola eccezione che i caratteri vengono visualizzati come asterischi '*'; in ogni caso il valore è passato al programma senza alcuna criptazione e quindi nella forma originale inserita dall'utente

- TYPE="hidden" si comporta alla stregua di "text", ma non visualizza alcunché sullo schermo. È utilizzato per passare dei valori allo script e/o per contenere valori temporanei acquisiti in moduli che si sviluppano su più pagine

FORM: RACCOGLIERE INFO

- TYPE="checkbox" visualizza sullo schermo delle caselle quadrate spuntabili per indicare delle situazioni di tipo vero/falso
  - un attributo che si può usare con questo tipo di dato è VALUE="testo" ed indica il valore da ritornare allo script nel caso in cui la casella venga spuntata; se non specificato viene ritornato il valore 'on'
  - altro attributo è CHECKED, senza valori, che se utilizzato rende la casella già spuntata al momento della visualizzazione.

FORM: RACCOGLIERE INFO

- TYPE="radio" definisce delle caselle di selezione a mutua esclusione, ossia è possibile selezionare solo una voce alla volta tra quelle disponibili; graficamente è rappresentato da un cerchio che se pieno contrassegna la voce come selezionata
  - per definire un gruppo di voci a mutua esclusione devono essere definiti tutti i pulsanti con <INPUT TYPE="radio" NAME="nome"> e avere tutti lo stesso nome, ossia lo stesso valore assegnato all'attributo NAME di ogni pulsante
  - a questo tipo si applica anche l'attributo VALUE="testo" che è il valore ritornato al programma e può essere diverso da casella a casella, e l'attributo CHECKED, senza valori, per selezionare un valore iniziale (si consiglia, pena comportamenti imprevedibili da parte del browser, di settare come CHECKED una sola voce per ogni gruppo)

FORM: RACCOGLIERE INFO

- TYPE="submit" crea il pulsante per l'invio dei dati raccolti; mediante l'attributo VALUE="testo" è possibile definire un'etichetta diversa da quella di default che è “Submit".

- TYPE="reset" crea un pulsante per annullare tutti i valori inseriti dall'utente, in modo che possa ricominciare da capo; mediante l'attributo VALUE="testo" è possibile definire un'etichetta diversa da quella di default che è "Reset".

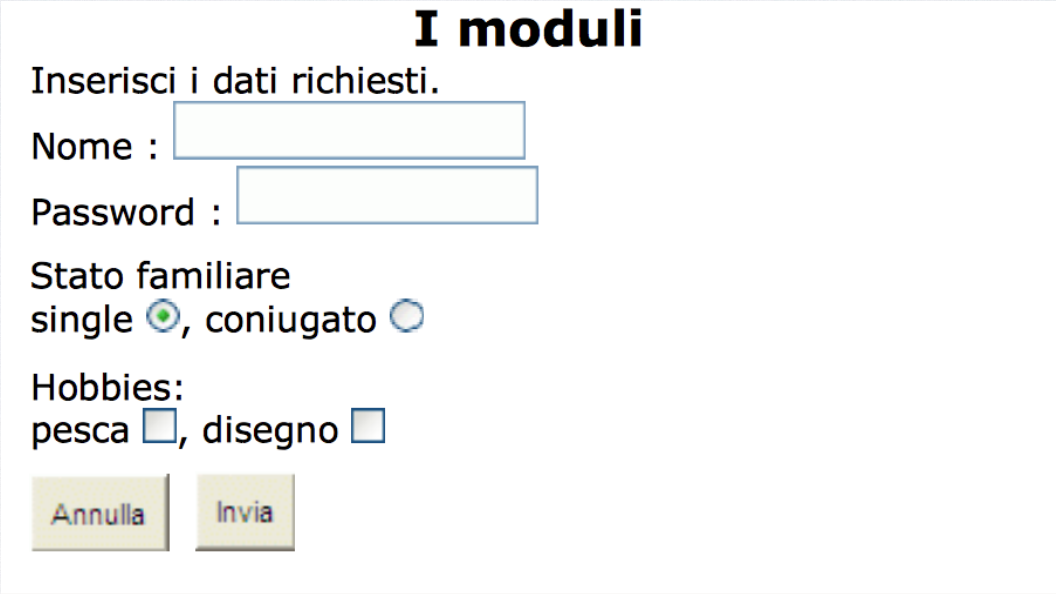
FORM: RACCOGLIERE INFO

- TYPE="image" si comporta esattamente come il pulsante "Submit" ma visualizza un immagine al posto del pulsante di invio
  - l'immagine è specificata dall'attributo aggiuntivo SRC="immagine.gif"
  - Assegnando all'immagine anche l'attributo NAME="nome", si fa sì che al programma vengano passate anche le informazioni sulla posizione del cursore del mouse al momento del clic; queste vengono passate nella forma nome.x=CoordX&nome.y=CoordY; nome indica il nome che si è assegnato all'immagine tramite l'attributo name, mentre CoordX e CoordY sono le coordinate x e y del cursore all'interno dell'immagine
- Vediamo adesso un esempio di modulo, in cui utilizziamo alcuni dei controlli del tag <INPUT>; sarà solo un esempio di visualizzazione delle caselle, non sarà un modulo vero e proprio, perché non possiamo progettare il programma che riceverà ed elaborerà i dati.
- Occorrerà conoscere linguaggi di programmazione come Java, oppure di scripting come il Perl

FORM: RACCOGLIERE INFO

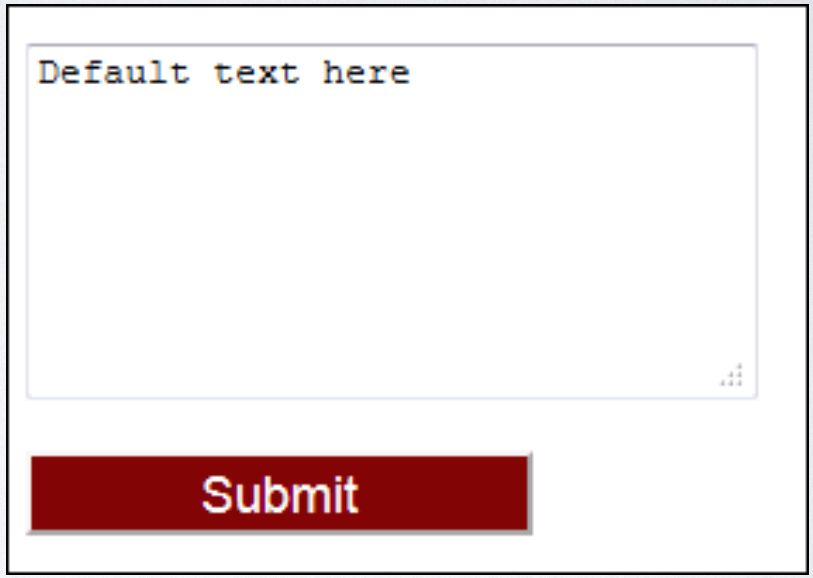
```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
    </HEAD>
    <BODY BGCOLOR="#BBFFBB">
        <H2 ALIGN="center">I moduli</H2>
        <P ALIGN="justify">Inserisci i dati richiesti.
        <FORM ACTION="..url.." METHOD="post">
            Nome : <INPUT TYPE="text" NAME="nome" SIZE="18"><BR>
            Password : <INPUT TYPE="password" NAME="pwd" SIZE="15"><BR>
            Stato familiare<BR>
            single <INPUT TYPE="radio" NAME="sfam" VALUE="single" CHECKED>
            , coniugato <INPUT TYPE="radio" NAME="sfam" VALUE="coniugato"><BR>
            Hobbies<BR>
            pesca <INPUT TYPE="checkbox" NAME="pesca">, disegno <INPUT TYPE="checkbox" NAME="disegno"><BR>
            <INPUT TYPE="reset" VALUE="Annulla">
            <INPUT TYPE="submit" VALUE="Invia">
            <BR>
        </FORM>
    </BODY>
</HTML>
```

FORM: RACCOGLIERE INFO



FORM: RACCOGLIERE INFO

- Il tag <INPUT> non è il solo utilizzato per reperire informazioni, ne esistono altri due, <TEXTAREA> e <SELECT>
- Il tag <TEXTAREA> si comporta come il tag <INPUT TYPE="text"> solo che il contenuto della casella può essere inserito su più righe
- Come <INPUT> esso infatti rileva l'attributo NAME="nome" con cui associare il nome della variabile, mentre ignora gli attributi MAXLENGHT, SIZE e VALUE
- è possibile però definire le dimensioni della casella mediante gli attributi cols="n" e rows="n" che indicano la larghezza in numero di caratteri e l'altezza in numero di righe.



FORM: RACCOGLIERE INFO

- Il tag <TEXTAREA> necessita del tag di chiusura </TEXTAREA> ed in mezzo ai due tag ci va l'eventuale testo che vorremmo inserire di default all'interno della casella; sarà l'utente a cancellarlo se vorrà cambiarlo

- Non è possibile dare una formattazione al testo da inserire nell'area, e quindi non è possibile inserire del codice HTML tra questi due tag; tutto quello che sarà scritto nel documento HTML apparirà all'interno della casella.

FORM: RACCOGLIERE INFO

- Il tag <SELECT> che permette di inserire una casella di selezione a discesa. Una casella di selezione a discesa è una casella contenente alcune voci, visualizzate una sopra l'altra e tra le quali è possibile selezionarne una voce

- Tra il tag di apertura e di chiusura vanno messe le varie opzioni selezionabili, mediante il tag <OPTION>.

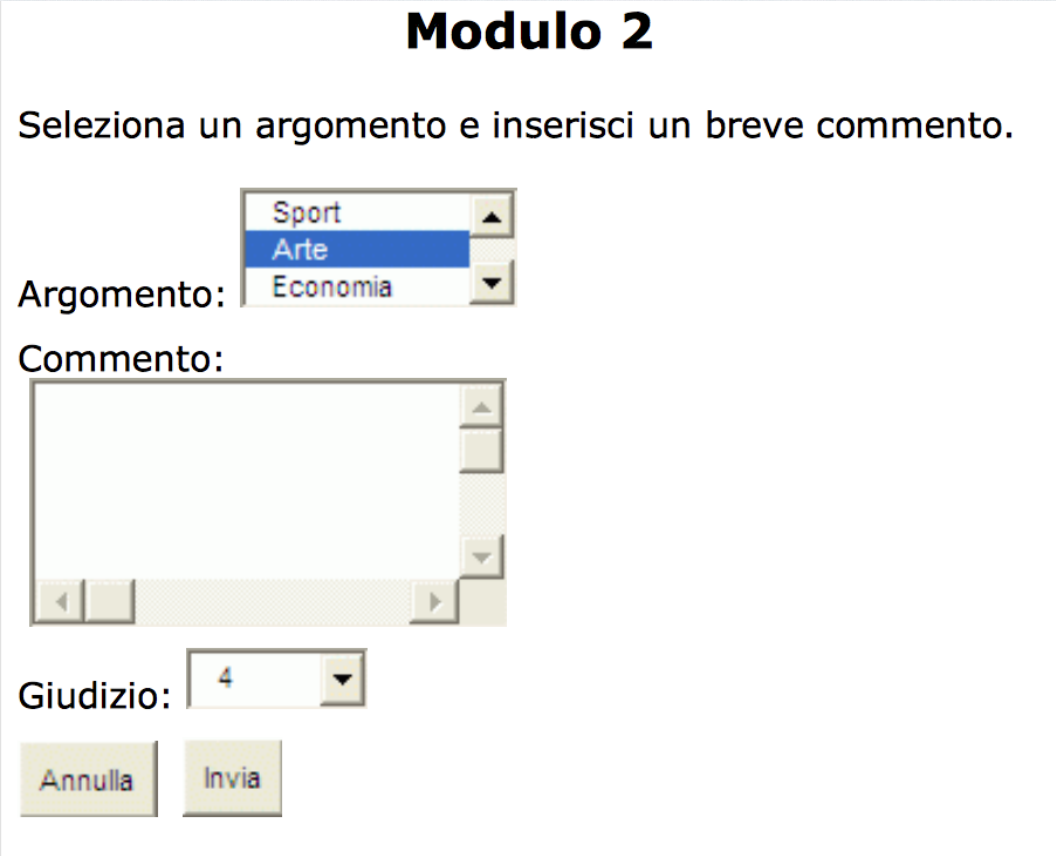
FORM: RACCOGLIERE INFO

- <SELECT> ammette tre attributi, NAME="nome", SIZE="n" e MULTIPLE
- l'attributo NAME serve per dare un nome alla variabile in questione
- SIZE serve per indicare il numero di voci da visualizzare sullo schermo
- MULTIPLE che se usato consente di selezionare più voci simultaneamente.
- Il tag <OPTION>, è usato per indicare una ad una le varie opzioni della casella; esso riconosce 2 attributi, VALUE="testo" e SELECTED
- Value serve per indicare il valore che sarà passato al programma legato alla variabile "nome"
- SELECTED, indica che quella voce è inizialmente selezionata per default; per evitare comportamenti imprevedibili da parte del browser è bene che questo attributo sia associato ad una sola opzione

FORM: RACCOGLIERE INFO

```
<HTML>
    <HEAD>
        <TITLE>Manuale di HTML</TITLE>
        .... </HEAD>
    <BODY BGCOLOR="#FFEBCD">
    <FORM  ACTION="..url.." METHOD="post">
        <H2 ALIGN="center">Modulo 2</H2>
        <P>Seleziona un argomento e<BR>
        inserisci un breve commento.</P>
        Argomento : <SELECT NAME="argom" SIZE="3" ALIGN="middle">
            <OPTION VALUE="sport">Sport</OPTION>
            <OPTION VALUE="arte" SELECTED>Arte</OPTION>
            <OPTION VALUE="econ">Economia</OPTION>
            <OPTION VALUE="cultura">Cultura</OPTION>
            <OPTION VALUE="storia">Storia</OPTION>
            <OPTION VALUE="tecno">Tecnologia</OPTION>
        </SELECT><BR>
        Commento:<BR>
        <TEXTAREA NAME="comment" rows="5" cols="20"></TEXTAREA><BR>
        Giudizio : <SELECT NAME="giudizio"> <OPTION VALUE="1">1</OPTION>
            <OPTION VALUE="2">2</OPTION>
            <OPTION VALUE="3">3</OPTION>
            <OPTION VALUE="4" SELECTED>4</OPTION> <OPTION VALUE="5">5</OPTION>
            <OPTION VALUE="6">6</OPTION> </SELECT><BR>
        <INPUT TYPE="reset" VALUE="Annulla"> <INPUT TYPE="submit" VALUE="Invia"> <BR>
    </FORM>
    </BODY>
</HTML>
```

FORM: RACCOGLIERE INFO



HTML5

- Con l'evoluzione di HTML, acronimo di Hyper Text Markup Language (linguaggio di marcatura per ipertesti) e di CSS (fogli di stile o Cascading Style Sheets), il World Wide Web si sta trasformando in un ambiente dove i documenti caricati sono associati ad informazioni e dati facilitando così l'interrogazione e l'interpretazione dei contenuti
- Questa evoluzione s'indica con il termine web semantico coniato da Tim Berners-Lee fondatore del www assieme a Robert Cailliau.
- A causa delle divergenze ideologiche tra il W3C e il WAHTWG (i due enti coinvolti nella definizione della specifica) e i diversi interessi delle aziende coinvolte nello sviluppo dei principali browser, HTML 5 adotta alcune scelte che possono sembrare discutibili ma sono frutto di sano pragmatismo

STORIA

- Il 4 giugno 2004, in una notte buia e tempestosa, Ian Hickson annuncia la creazione del gruppo di ricerca **WHAT**, acronimo di Web Hypertext Application Technology in un **sintetico ma ricco messaggio**.
- L'obiettivo del gruppo è quello di elaborare specifiche per aiutare lo sviluppo di un web più orientato alle applicazioni che ai documenti; tra i sottoscrittori di questa iniziativa si annoverano aziende del calibro di Apple, Mozilla e Opera.
- Questa piccolo scisma dal W3C è determinato dal disaccordo in merito alla rotta decisa dal consorzio durante un famoso convegno del 2004 dove, per un pugno di voti, prevalse la linea orientata ai documenti di XHTML2.
- "XHTML 2.0 considered harmful" è il titolo di un messaggio inviato alla mailing list ufficiale del W3C datato 14 gennaio 2003 che ben riassume i sentimenti contrastanti che all'epoca si respiravano in merito a queste specifiche.

STORIA

- La principale causa di perplessità è da ricercarsi nella decisione azzardata di non mantenere la retro-compatibilità con la versione 1.1 eliminando alcuni tag e imponendo agli sviluppatori web un controllo rigoroso nella creazione delle pagine, pena lo stop del processo di parsing e la visualizzazione a schermo degli errori di interpretazione
- Nei due anni successivi i gruppi XHTML2 e WHAT proseguono i lavori sulle proprie specifiche in modo indipendente e parallelo, sollevando dubbi e confusione sia da parte dei produttori di browser che degli sviluppatori web.
- Emblematico in tal senso è un articolo firmato da Edd Dumbill nel dicembre 2005 intitolato The future of HTML. Il 27 ottobre 2006 in un post sul proprio blog intitolato Reinventing HTML, Tim Berners-Lee annuncia la volontà di creare un nuovo gruppo di ricerca che strizzi l'occhio al WHAT e ammette alcuni sbagli commessi seguendo la filosofia XHTML2:
- Some things are clearer with hindsight of several years. It is necessary to evolve HTML incrementally. The attempt to get the world to switch to XML, including quotes around attribute values and slashes in empty tags and namespaces all at once didn't work. The large HTML-generating public did not move, largely because the browsers didn't complain. Some large communities did shift and are enjoying the fruits of well-formed systems, but not all. It is important to maintain HTML incrementally, as well as continuing a transition to well- formed world, and developing more power in that world. Tim Berners-Lee.

STORIA

- Dovranno passare quasi altri due anni di competizione tra le due specifiche, questa volta entrambe interne al W3C, prima che nel luglio del 2009 lo stesso Tim sancisca di non voler riconfermare il gruppo XHTML2 per l'anno successivo preferendo di fatto la direzione intrapresa dall'HTML5. Frattanto il gruppo di ricerca, formato da una commistione di elementi del W3C e del WHAT, sotto la guida di Ian Hickson, è giunto alla 4° versione delle specifiche.

- Nonostante il continuo e solerte lavoro e il grande intervallo di tempo speso nella stesura di questo documento, i passi residui necessari ad eleggere questa tecnologia al rango di 'W3C Reccomandation', relegandola così tra gli standard riconosciuti, sono ancora molti, al punto che si prevede di raggiungere l'agognato traguardo soltanto attorno al 2020.

- Ricordiamo però che il consorzio si riferisce sempre all'intero universo inscritto nelle specifiche mentre alcune feature ritenute peculiari ed importanti, ad esempio il tag <video>, sono già implementate da tempo dalla totalità (o quasi) dei browser in commercio.

HTML5: COSA È

- Domanda semplice solo all'apparenza; siamo sicuramente di fronte alla quinta revisione delle specifiche HTML ma anche di un vastissimo elenco di funzionalità che si sviluppano attorno al tema del linguaggio di markup.
- È importante ricordare che, anche in virtù della storia della sua nascita, all'interno dell'HTML5 convivono in armonia due anime: la prima, che raccoglie l'eredità semantica dell'XHTML2, e la seconda che invece deriva dallo sforzo di aiutare lo sviluppo di applicazioni Web.
- Il risultato di questo mix di intenti è più articolato di quanto si immagini; in prima istanza si assiste ad una evoluzione del modello di markup, che non solo si amplia per accogliere nuovi elementi, ma modifica in modo sensibile anche le basi della propria sintassi e le regole per la disposizione dei contenuti sulla pagina.

HTML5: COSA È

- A questo segue un rinvigorimento delle API JavaScript che vengono estese per supportare tutte le funzionalità di cui una applicazione moderna potrebbe aver bisogno:
    - salvare informazioni sul device dell'utente e accedere all'applicazione anche in assenza di una connessione Web;
    - comunicare in modo bidirezionale sia con il server sia con altre applicazioni;
    - eseguire operazioni in background;
    - pilotare flussi multimediali (video, audio);
    - editare contenuti anche complessi, come documenti di testo;
    - pilotare lo storico della navigazione;
    - usare metafore di interazione tipiche di applicazioni desktop, come il drag and drop;
    - generare grafica 2D e grafica 3D in tempo reale;
    - accedere e manipolare informazioni generate in tempo reale dall'utente attraverso sensori multimediali quali microfono e webcam.

HTML5: COSA È

- Anche l'aspetto semantico a contorno del markup non è dimenticato
- accedere alle informazioni geografiche del device dell'utente (posizione, orientamento,…);
- mantenere un database sul device dell'utente;
- generare grafica 3D in tempo reale;
- E non dimentichiamo che l'evoluzione dell'HTML viaggia di pari passo con quella dei CSS, che si avvicinano alla terza versione, e di altri importanti standard come SVG e MathML, e che ognuno di questi componenti è progettato nella sua versione più recente per recare e ricevere beneficio dagli altri.

HTML5: PERCHÉ

- Quando sono state definite le precedenti versioni, il web era strettamente collegato al concetto di ipertesto: gli utenti lo utilizzavano con il principale obiettivo di ottenere informazioni tipicamente in forma testuale, la velocità di connessione era mediamente bassa e di conseguenza applicazioni web quali quelle odierne erano impensabili. L'obbiettivo principale era quello di pubblicare semplici documenti testuali collegati fra loro.

- Negli anni successivi il numero di utenti è cresciuto molto rapidamente e questo ha permesso investimenti per aumentare le velocità di connessione e quindi lo sviluppo di applicazioni web più complesse. Le specifiche però non erano adatte a questo nuovo utilizzo del web, quindi molti sviluppatori si sono visti costretti ad aggirare le limitazioni imposte con metodologie spesso complesse e poco eleganti.

- Queste due nuove tecnologie nascono per offrire un supporto a questo nuovo mondo del web sia per quanto riguarda la strutturazione del contenuto, sia per lo sviluppo di vere e proprie applicazioni web. Il World Wide Web Consortium ha annunciato che la prima versione dello standard HTML 5 è pronta per il luglio 2014

HTML5: PRIVACY

- Con l'avvenire di HTML 5 e delle novità ad esso collegate, come il Web Storage e la geolocalizzazione, il tema della privacy è sempre più sensibile
- Gli utenti che utilizzeranno questi strumenti condivideranno informazioni riguardanti i loro interessi e le loro attitudini in modo nettamente superiore rispetto alle informazioni che lasciamo in rete oggi.

FLASH VS HTML5

- Dopo numerose battaglie tra le due tecnologie, Adobe ha annunciato ufficialmente che non svilupperà più Flash per dispositivi mobili, Flash Player 11 sarà l'ultima versione ufficiale rilasciata per dispositivi mobili e il futuro sarà HTML 5.
- Steve Jobs, in una lettera aperta intitolata "Thoughts on Flash" del 2010, spiega quali sono i motivi che hanno portato Apple ad optare per HTML 5 preannunciando un futuro senza il software di Adobe per quanto riguarda le animazioni nel campo mobile. Forse il contributo al decadimento di questa tecnologia, per quanto riguarda i dispositivi mobili, in parte è da attribuire proprio alla campagna di pubblicità negativa dettata da Apple. Basti pensare che iPhone e iPad non supportano tale tecnologia. Inoltre anche Microsoft, YouTube, Amazon e molte altre aziende stanno migrando verso la nuova tecnologia HTML.
- Oltre alla pubblicità negativa, HTML 5, rispetto a Flash, offre un linguaggio per le animazioni molto più leggero ed è supportato da tutti i browser moderni, tramite l'utilizzo di librerie JavaScript, HTML 5 sarà in grado di rimpiazzare completamente Flash.

Dott. Antonio Giovanni Lezzi

PERCHÉ PASSARE A HTML5

- Il percorso di crescita del web ha fatto emergere alcune strutture di contenuto ricorrenti, ben caratterizzate dal fenomeno dei blog: informazioni di testata, menu di navigazione, elenchi di articoli, testo a pie' di pagina, ed altri
- La parte dedicata al singolo articolo presenta anch'essa solitamente lo stesso set di informazioni quali autore, data di pubblicazione, titolo e corpo del messaggio. Anche in questo caso l'HTML4 non ha saputo fornire gli strumenti adatti a consentire una corretta gestione e classificazione del contenuto obbligando gli sviluppatori web a ripiegare su strutture anonime, quali e <p>, arricchite di valore semantico con l'utilizzo di attributi quali class e id.
- L'HTML5 nasce per risolvere questi problemi offrendo agli sviluppatori web un linguaggio pronto ad essere plasmato secondo le più recenti necessità, sia dal lato della strutturazione del contenuto che da quello dello sviluppo di vere e proprie applicazioni.

ELEMENTI E ATTRIBUTI NON PRESENTI

- Le **specifiche HTML5** sanciscono definitivamente la fine di tutta una serie di elementi e attributi che mantengono validità formale solo per preservare la retrocompatibilità di pagine datate ma sono espressamente vietati nella creazione di nuovi documenti.
- I primi a subire questo esilio sono tutti quei costrutti funzionali alla parte di presentazione e caduti ampiamente in disuso con l'introduzione dei fogli di stile. Stiamo parlando di elementi come: basefont, big, center, font, s, strike, tt e u.
- Ad essi si aggiunge una moltitudine di attributi più o meno noti, tra i quali ricordiamo: align e valign, background, bgcolor, cellpadding, border, cellspacing e molti altri
- Interessante notare come si sia deciso invece di mantenere elementi come i e b; trasformandoli, però, da modificatori tipografici a semplici indicatori di porzioni di testo particolari e consentendone l'uso solo come ultima risorsa.
- Cadono, infine, anche tutti i tag che gravitano intorno al concetto dei frame, ritenuti dannosi per usabilità e accessibilità: frame, frameset e noframes

SEMANTICA

- Con semantica di un tag s'intende il significato che l'elemento attribuisce al suo contenuto
- Con le nuove specifiche di HTML 5 gli attributi stilistici vengono gestiti solamente tramite fogli di stile in modo tale da separare il contenuto di un documento dalla sua visualizzazione.

DIFFERENZE TRA TAG HTML

- Abbiamo osservato che esistono tre distinte versioni di un Tag, ognuna di esse si applica selettivamente solo ad alcuni elementi:

- Tag 'classico'

```
<p> bla bla bla bla ... </p>
```

- Tag 'vuoto'

```
<img src="immagine_interessante.jpg" alt="Una immagine interessante">
```

- Tag 'autochiudente'

```
<rect x="150" y="40" width="60" height="30" fill="black" stroke="black"/>
```

MAIUSCOLO MINUSCOLO

- Una delle differenze principali rispetto alle regole XHTML riguarda l'uso del maiuscolo e del minuscolo per definire un tag
- In XHTML è obbligatorio usare il minuscolo
- In HTML5 è consentito scrivere un tag usando anche il maiuscolo:
- <IMG src="immagine.png" alt="testo">

SEMPLIFICAZIONI

- In direzione della semplificazione vanno anche altre indicazioni. Ci soffermiamo su quelle riguardanti due elementi fondamentali come style e script

- La sintassi tipica di XHTML prevede la specificazione di attributi accessori come type:
  <style type="text/css"> regole CSS... </style>
  <script type="text/javascript" src="script.js"> </script>

- In HTML5 possiamo farne a meno, scrivendo dunque:
  <style> regole CSS... </style>
  <script src="script.js"> </script>

DOCTYPE

- Il doctype è il primo elemento che dovrebbe comparire in un documento HTML. La sua funzione è di validare il documento e, per alcuni browser, serve ad abilitare la modalità di interpretazione standard.

- Doctype HTML 5
  ```
  <!DOCTYPE html>
  ```

- Doctype HTML 4.01 strict
  ```
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
  ```

- Inserire il doctype di HTML 4 come si può ben vedere non era per niente semplice. Fortunatamente con la nuova specifica la sintassi di questo elemento è molto più snella.

CODIFICA DEI CARATTERI

- Nell'intestazione di un documento HTML (all'interno dei tag <head></head>) sono definiti i tag <meta> tra i quali uno è utilizzato per definire la codifica dei caratteri.

- Codifica caratteri HTML 5
  ```
  <META charset=UTF-8">
  ```

- Codifica caratteri HTML 4.01
  ```
  <META http-equiv="Content-Type" content="text/html;
  charset=UTF-8">
  ```

- Anche in questo caso si può rappresentare lo stesso attributo con una sintassi più agile.

ATTRIBUTI

- Anche rispetto alle definizione degli attributi HTML5 consente una libertà maggiore rispetto a XHTML, segnando di fatto un ritorno alla filosofia di HTML 4
- In sintesi: non è più obbligatorio racchiudere i valori degli attributi tra virgolette.
- I casi previsti nella specifica sono 4.
- Attributi 'vuoti': non è necessario definire un valore per l'attributo, basta il nome, il valore si ricava implicitamente dalla stringa vuota. Un caso da manuale:
- Secondo le regole XHTML:
  <input checked="checked" />
- In HTML5:
  <input checked>

ATTRIBUTI

- Attributi senza virgolette: è perfettamente lecito in HTML5 definire un attributo senza racchiudere il valore tra virgolette. Esempio:

- Attributi con apostrofo: il valore di un attributo può essere racchiuso tra due apostrofi (termine più corretto rispetto a 'virgoletta singola'). Esempio:

- Attributi con virgolette: per concludere, è possibile usare la sintassi che prevede l'uso delle virgolette per racchiudere il valore di un attributo. Il codice:

DATA ATTRIBUTI

- Lo standard HTML5 ha introdotto i cosiddetti "data-attributes", attributi inseriti nei tag allo scopo di ritenere alcune informazioni utili alla gestione del layout o a supporto dell'interfaccia utente.

- Possiamo definire un attributo con il prefisso data-, seguito dal nome (es. data-cognome) e assegnare ad esso un valore proprio come facciamo con gli altri attributi.

- Mettiamo ad esempio di avere un'agenda e di voler mostrare l'indirizzo e il numero di telefono quando passiamo vicino ai nomi in un elenco. Possiamo associare tutte le informazioni che ci servono ad ogni elemento lista (<li>) grazie agli attributi "data-" e raggiungerli poi facilmente via JavaScript.

```
<ul id="elenco">
<li data-indirizzo="Piazzale Cantore" data-telefono="3204256969">Antonio</li>
<li data-indirizzo="Via le mani dal naso" data-telefono="321254685">Paolo</li>
<li data-indirizzo="Largo Augusto" data-telefono="3213344314">Umberto</li>
</ul>
```

- L'inserimento di questi attributi personalizzati non avrà alcun effetto sulla resa della pagina. Il browser si limiterà ad ignorarli.

CASI PARTICOLARI

- Esistono alcune casistiche per le quali un tag classico può mancare della sua particella di apertura o di chiusura; questo succede quando il browser è comunque in grado di determinare i limiti di operatività dell'elemento
- Gli esempi più eclatanti riguardano i tag 'contenitori', come head, body e html, che possono essere omessi in toto a meno che non contengano un commento o del testo come istruzione successiva
- È quindi sintatticamente corretto scrivere un documento come segue:

```
<meta charset="utf-8">
<title>Pagina HTML5 Valida</title>
<p>Un paragrafo può non avere la particella di chiusura
<ol>
  <li>e anche un elemento di lista
</ol>
```

- I tag p e li possono essere scritti omettendo la particella di chiusura, a patto che l'elemento successivo sia all'interno di una cerchia di elementi definita dalle specifiche. A fronte di queste opzioni di semplificazione è però errato pensare che la pagina generata dal codice di cui sopra manchi, ad esempio, dell'elemento html; esso è infatti dichiarato implicitamente ed inserito a runtime dallo user-agent.

ELEMENTI DI SCRITTURA

- Gli elementi di struttura sono i tag che specificano la suddivisione in aree logiche di una pagina web.
- Inizialmente la struttura di una pagina web era composta di tabelle suddivise in righe e colonne nelle quali erano presenti le varie sezioni del documento HTML. I principali svantaggi di questa metodologia riguardavano: i motori di rendering, poiché la presenza di più tabelle annidiate tra loro rallentava lo scaricamento della pagina web, e il problema della mescolanza tra elementi usati per marcare il contenuto e quelli usati per la visualizzazione. Inoltre per eseguire una sostanziale modifica sulla visualizzazione della pagina i tempi non erano di certo brevi, data la complessità della struttura che spesso veniva a crearsi.
- In seguito le tabelle vennero sostituite dai , contenitori generici, che consentono la creazione di pagine web meno complesse e di facile manutenibilità. Anche in questo caso però lo stesso elemento era finalizzato a descrivere contenuto e visualizzazione. Per marcare il contenuto spesso si utilizzavano le classi di stile (un classico esempio è).
- Sulla base di queste esperienze HTML 5 introduce nuovi elementi per poter migliorare la semantica dei documenti.

HTML DEL PASSATO

- All'inizio erano tabelle; ricordiamo tutti bene il tedio provocato dal dover costruire strutture complesse spezzandole all'interno di una griglia fatta da infiniti <tr> e <td>: un attività noiosa, resa ancora peggiore dalla scarsa qualità e flessibilità del risultato

- Poi arrivarono i e fu una vera e propria rivelazione; finalmente un modello di costruzione del documento pensato per separare in modo chiaro i contenuti tipici di una pagina web.

HTML DEL PASSATO

```
<html>
<head> </head>
  <body>
    <div id="header">
      --- Titolo e Testata ---
    </div>
    <div id="body">
       <div id="menu">
       --- Voci di Menu ---
       </div>
       <div id="main_content">
          <div class="post">
          --- Un Post ---
          </div>
          <div class="post">
          --- Un altro Post ---
          </div>
      </div>
    </div>
  </body>
</html>
```

HTML DEL PASSATO

- All'interno di un costrutto come questo è tutto molto semplice da interpretare: basta seguire il flusso dei rientri di pagina facendosi guidare dai valori semantici attribuiti a id e class.
- Successivamente ci si accorse che né il browser né tantomeno i motori di ricerca avrebbero mai potuto beneficiare di questa divisione semantica, proprio per colpa di quell'arbitrarietà che permetteva a milioni di siti web di essere organizzati in strutture simili ma sempre leggermente diverse tra loro e per questo non raggruppabili secondo schemi automatici
- Emerse in questo modo uno dei più grandi problemi dell'HTML4: l'incapacità di descrivere il significato delle informazioni di una pagina web in un formato interpretabile da altri software
- In un mondo sempre più caratterizzato dall'interconnessione e dall'aggregazione su base semantica dei contenuti ci si adattò inventando e condividendo convenzioni studiate per rappresentare eventi di calendario, persone e quant'altro; nacquero così diversi microformati

MICROFORMATI

- Un microformat (abbreviato con µF o uF) è una parte di mark up che consente espressioni semantiche in una pagina web HTML (o XHTML) ed è possibile sfruttando gli attributi (X)HTML class, rel, rev.
- Tramite i microformat, i programmi possono estrarre i dati semantici presenti in una pagina web. I microformat permettono infatti di creare codice (X)HTML leggibile dai programmi ma continuando a garantire un'elevata comprensibilità da parte delle persone
- Le pagine web create sfruttando i microformat permettono ai programmi di esaminarne i contenuti e di utilizzare le informazioni in esse contenute
- Ad esempio, attraverso il Microformat hCard, specifico per la descrizione delle persone, il browser può facilmente riconoscere, all'interno di una pagina web, l'indirizzo e-mail o il numero di telefono dei contatti presenti nella pagina, in modo da poterle trasferire velocemente nella rubrica.
- http://microformats.org/wiki/Main_Page

MICROFORMATI

```
<div>
  <div>Joe Doe</div>
  <div>The Example Company</div>
  <div>604-555-1234</div>
  <a href="http://example.com/">http://example.com/</a>
</div>
```

Con il microformat hCard, diventa:

```
<div class="vcard">
  <div class="fn">Joe Doe</div>
  <div class="org">The Example Company</div>
  <div class="tel">604-555-1234</div>
  <a class="url" href="http://example.com/">http://example.com/</a>
</div>
```

MICRODATA

- La specifica dei microdati HTML 5 è un modo per assegnare etichette ai contenuti al fine di descrivere un tipo specifico d'informazioni (ad esempio recensioni, informazioni su persone o eventi). Ogni tipo d'informazione descrive uno specifico tipo di elemento, come una persona, un evento o una recensione. Ad esempio, un evento ha proprietà quali il luogo, l'ora d'inizio, il nome e la categoria.

- L'obbiettivo è quello di esaltare le proprietà semantiche per dare la possibilità a programmi come crawler dei motori di ricerca o screen reader di comprendere il significato del testo. Queste informazioni sono accessibili da questi programmi e rimangono (per ora) invisibili per l'utente.

MICRODATA

- Sintassi
- Marcare i dati pubblicati con il formato microdata è semplice, è sufficiente applicare i seguenti attributi ai tag HTML in modo tale da definire gli oggetti semantici.
- **Itemscope:** Questo è un attributo globale che contrassegna un insieme di proprietà logicamente legate tra loro, inoltre si precisa che all'interno dell'elemento che lo contiene sono presenti informazioni dell'oggetto che si vuole descrivere.
- **Itemprop:** Definisce il nome della proprietà associata all'oggetto che si vuole descrivere. L'attributo può essere definito, ad esmpio, in un tag , in questo caso il valore della proprietà sarà il contenuto dell'elemento ma se itemprop è definito, ad esempio, in un tag <img> il valore della proprietà va ricercato all'interno dell'elemento. Nel caso dell'immagine, il valore della proprietà definita sarà il valore dell'attributo src.

MICRODATA

- **Itemtype:** permette la "tipizzazione" degli elementi mircodata e serve a distinguere una proprietà con un determinato nome associata a un determinato oggetto, da una proprietà con lo stesso nome ma associata a un oggetto da descrivere differente

- Il valore di itemtype delinea il contesto delle proprietà definendo così un vocabolario. Conviene utilizzare vocabolari che siano ampiamente diffusi e condivisi in modo tale da favorire la condivisione delle informazioni. Il valore di itemtype può quindi essere un collegamento ipertestuale che fa riferimento ad un vocabolario condiviso in rete.

MICRODATA: ESEMPIO

```
<div itemscope itemtype="http://data-vocabulary.org/Person"> Il mio nome è
<span itemprop=‘‘name">Antonio Lezzi</span> e il mio nickname in alcuni siti web è
<span itemprop="nickname">Letus</span>. Questo è il mio account twitter:
<a href="http://www.example.com" itemprop="url">http://www.example.com</a> Vivo a
Milano in Italia e sono uno
<span itemprop="role">professionista</span>.
</div>
```

RICH SNIPPET

- I microdati stanno alla base dei rich snippet di Google ovvero le informazioni aggiuntive visualizzate dopo una ricerca
- Google ha rilevato come i collegamenti che offrono una sintesi ben strutturata delle proprie informazioni abbiano sperimentato un incremento di traffico rilevante. Ecco un esempio:
-

RICH SNIPPET

- Per definire i microdata in modo tale che siano interpretabili dai rich snippet visitare la pagina web nella quale sono presenti i vocabolari di Google: http://www.data-vocabulary.org/ .
- I vocabolari più popolari, supportati da Google, sono:
- **Breadcrumbs**: serve a rappresentare un insieme di collegamenti che possano aiutare un utente a comprendere e navigare all'interno del sito.
- **Businesses** and organizations: definisce una società, un negozio e più in generale un luogo.
- **People**: definisce una persona.
- **Address**: definisce un indirizzo.
- **Events**: definisce un evento (con informazioni come il titolo, la data e il luogo).

RICH SNIPPET

- **Product**: definisce un prodotto.
- **Offer**: definisce un'offerta.
- **Offer-aggregate**: definisce un aggregato di offerte (con prezzo minimo, prezzo massimo, etc).
- **Recipes**: definisce una ricetta.
- **Review**: definisce una singola recensione.
- **Review-aggregate**: definisce una recensione aggregata.
- **Rating**: definisce una valutazione.

STRUTTURA

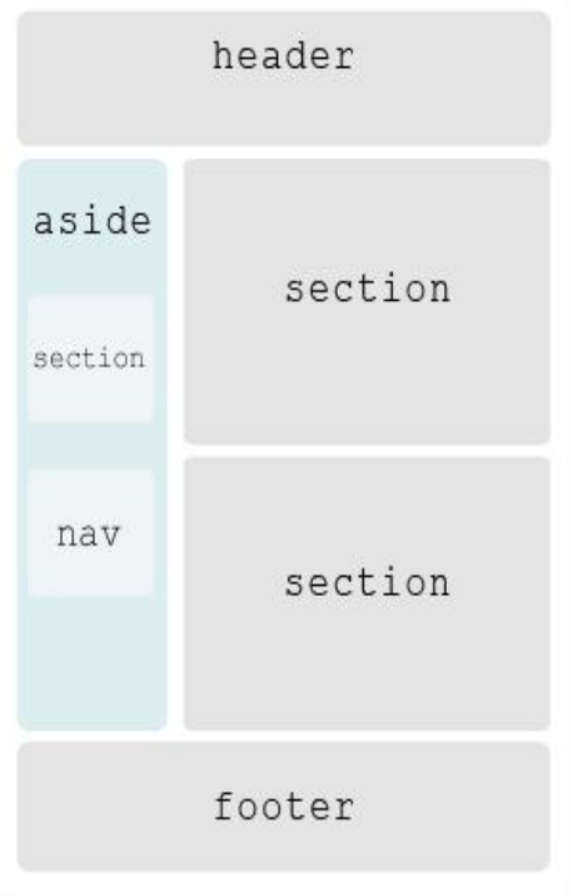


Figura 1: struttura di una pagina

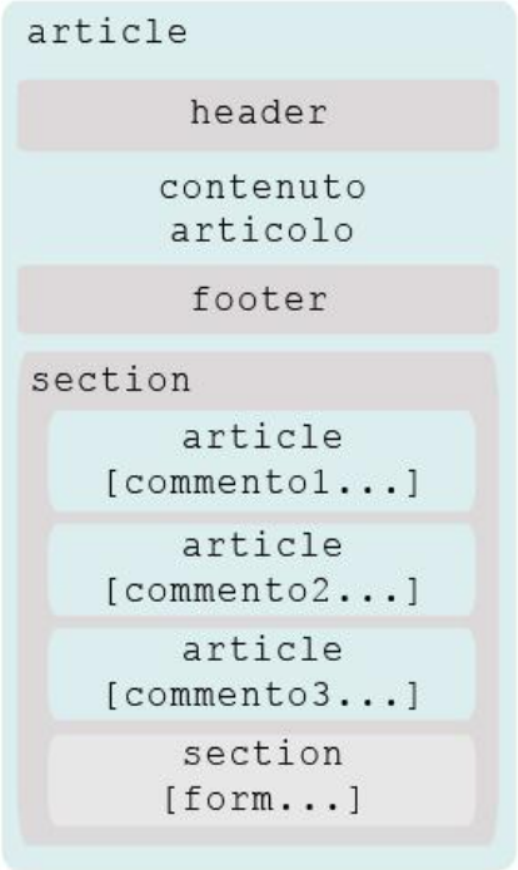


Figura 2: suddivisione semantica degli articoli

HEADER

- La specifica impone che questo elemento debba delimitare l'intestazione di una sezione del documento e possono essere molteplici all'interno di un documento

- Esempio <header>

```
<header>
        <h1>Algoritmo Pippo</h1>
        <p>Scritto da Marco Rossi</p>
        <p>Pubblicato il 29-05-2017</p>
</header>
```

ELEMENTO <NAV>

- Nell'intestazione di una sezione possono essere presenti degli elementi con il fine di abilitare la navigazione del sito web, spesso sono inseriti anche a piè di pagina. Questi elementi dovrebbero essere racchiusi tra i tag <nav> ma non necessariamente qualsiasi gruppo di elementi di navigazione dev'essere delimitato da questo nuovo tag.

- Esempio <nav>

```
<nav>
    <ul>
        <li><a href="ricerche.html">Ricerche</a></li>
        <li><a href="chisiamo.html">Chi siamo</a></li>
        <li><a href="contatti.html">Contatti</a></li>
    </ul>
</nav>
```

ELEMENTO <SECTION>

- Questo elemento rappresenta una sezione generica di un documento o applicazione
- Individua un raggruppamento tematico di contenuti, ed in genere contiene un titolo introduttivo
- Esempi d'uso potrebbero essere le sezioni di una homepage o i capitoli di un libro
- Non è da intendersi come un elemento la cui finalità è l'attribuzione di un determinato stile per gli elementi che racchiude, per questo scopo è sempre valido il classico
- In oltre se l'area da racchiudere è destinata a un utilizzo via RSS questo non è l'elemento appropriato per tale impiego, il tag <article> è più opportuno

ELEMENTO <ARTICLE>

- L'elemento <article> dev'essere utilizzato quando s'inserisce informazioni indipendenti dal resto del documento o della sezione in cui è racchiuso
- Può trattarsi ad esempio di un articolo di giornale o una pubblicazione in un blog o un qualsiasi altro elemento indipendente dal contenuto.

ELEMENTO <ASIDE>

- Se si vogliono aggiungere informazioni aggiuntive a un elemento, questo è il tag appropriato
- Le informazioni che racchiude, se rimosse, non devono incidere negativamente nella completezza dell'informazione contenuta nell'elemento a cui è associato
- Può essere utilizzato ad esempio per racchiudere elementi <nav> aggiuntivi, barre laterali o annunci.

ELEMENTO <FOOTER>

- Analogamente alle intestazioni anche i piè di pagina possono essere molteplici all'interno di una pagina web
- Contrariamente a quanto si potrebbe pensare non è detto che questo tag debba essere l'ultimo elemento di una determinata sezione
- Esempio <footer>

```
<footer>
    <nav>
        <ul>
            <li><a href="ricerche.html">Ricerche</a></li>
            <li><a href="chisiamo.html">Chi siamo</a></li>
            <li><a href="contatti.html">Contatti</a></li>
        </ul>
    </nav>
    <p>Corso di perfezionamento</p>
</footer>
```

JAVASCRIPT

- Durante questa fase di transazione rivolta verso le nuove specifiche, si sono diffuse alcune librerie JavaScript per dare un supporto alle nuove tecnologie:
  - Testare la capacità o meno dei vari browser di elaborare correttamente le nuove funzionalità: Modernizr è una di queste, lo scopo è verificare il funzionamento di determinate funzioni indipendentemente dal browser utilizzato.
  - Sostituire le funzionalità del browser mancanti con altre funzioni: la libreria excanvas creata da Google, ad esempio, permette l'utilizzo dell'elemento <canvas> sulle versioni di Internet Explorer precedenti alla 9.
  - Effetti grafici: queste librerie evidenziano le potenzialità di questa innovazione, un esempio potrebbe essere la libreria close-pixelate di David De Sandro.

RETROCOMPATIBILITÀ

- Innanzitutto per utilizzare le nuove specifiche bisogna adottare una strategia di retrocompatibilità nel caso in cui queste non siano supportate.
- Una strategia potrebbe essere quella di identificare il browser utilizzato e la sua versione nel momento in cui si visualizza una pagina web e prendere decisioni in funzione del risultato.
- Una strategia migliore prevede l'utilizzo della libreria JavaScript Modernizr che esegue rapidamente una serie di test per stabilire se una data funzionalità sia supportata o meno.

```
<script>
    if(Modernizr.video()) {
        /*supportato*/
    }
   else {
        /*non supportato: strategie alternative*/
   } </script>
```

Video è una proprietà booleana dell'oggetto Modernizr, se il suo valore è true allora il browser supporterà l'elemento <video>

RETROCOMPATIBILITÀ

- Le versioni di Internet Explorer precedenti alla 9 quando individuano elementi sconosciuti non applicano gli stili associati. Per aggirare questo problema si può creare in memoria questi elementi con l'utilizzo di JavaScript.

- Elementi di struttura in memoria

```
<script>
    document.createElement("header");
    document.createElement("footer");
    document.createElement("nav");
    document.createElement("section");
    document.createElement("article");
    document.createElement("aside");
</script>
```

Home | News | Apr 25, 2017 - New feature added: CSS display: contents | Compare browsers | About

Know when and why code breaks — Rollbar: Users finding bugs? Searching logs for errors? Find + fix broken code fast! AD

Can I use _____ ? Settings

Detected your country as "Italy". Would you like to import usage data for that country?

Import | No thanks

Index of features

Latest features

- CSS display: contents
- Link type "noreferrer"
- DOMMatrix
- Rebeccapurple color
- navigator.hardwareConcurrency

Most searched features

1. Flexbox
2. CSS Grid
3. CSS transforms
4. SVG
5. CSS calc()

Did you know?

If a feature you're looking for is not available on the site, you can vote to have it included. Better yet, if you've done the research you can even submit it yourself!

Next

Browser scores

Current version | Dev version

Chrome 58: 288
Firefox 53: 285
Safari 10.1: 243
Edge 15: 223

Third party tools

- The CanIUse Embed — Add support tables to your site
- Caniuse Component — Add support tables to your presentations
- Caniuse command line tool
- Doiuse...? — Lint your CSS to check what features work
- I want to use — Select multiple features and see what % of users can use them

HTTP://CANIUSE.COM

NUOVI ELEMENTI

- I tag presentati finora concernevano le novità riguardanti la nuova struttura delle pagine web
- Oltre a questi sono stati introdotti nuovi elementi con il fine di fornire un maggior dettaglio dei contenuti
- Alcuni di questi tag oggi non sono ancora supportati dalle più recenti versioni dei browser più popolari.

ELEMENTI <FIGURE> E <FIGCAPTION>

- Secondo la specifica l'elemento <figure> dovrebbe essere utilizzato per contenere elenchi di codice, immagini, diagrammi, ecc... Con la condizione che non siano parte integrante del contesto in cui sono inseriti

- L'elemento <figcaption> ha il compito di arricchire la descrizione dell'elemento posto all'interno del tag <figure>.

- Esempio <figure> e <figcaption>

```
<figure>
    <img src="dei.jpg" alt="Laboratorio sviluppo siti web">
    <figcaption>La sede di questa scuola vista dall'esterno.</
figcaption>
</figure>
```

ELEMENTO <HGROUP>

- Questo elemento fa parte dell'intestazione di una sezione di una pagina, ma a differenza del tag <header>, raggruppa solamente elementi d'intestazione da<h1>a<h6>
- La vera importanza del tag <hgroup> è che maschera l'outline dell'elemento padre che lo contiene; infatti, l'algoritmo dell'outliner riconosce come un titolo solamente l'heading con il valore più alto e considera tutti gli altri elementi sottotitoli
- Questo consente di utilizzare ad esempio un <h2> come sottotitolo ma di escluderlo da un insieme di titoli <h2> associati ad una determinata sequenza logica.
- Esempio <hgroup>

```
<hgroup>
    <h1>Unipd Lab</h1>
    <h2>Il laboratorio dell'università di Padova</h2>
</hgroup>
```

ELEMENTO <TIME>

- Questo elemento permette di rappresentare una data in un formato leggibile sia dagli utenti sia dalle macchine.
- Esempio <time>
  Arrivò a Padova alle <time datetiame="2017-01-21T09:00+1:00"> 9 del mattino del 21 Gennaio 2017</time> a causa di un ritardo del treno.
- L'attributo datetime indica data e ora nel formato ISO. Il fuso orario può anche essere omesso, in tal caso la data sarà stabilita dal software utilizzato per la visualizzazione. Si potrebbe anche omettere l'intero attributo datetime con la condizione di inserire all'interno dell'elemento <time> una data conforme al formato ISO.
- Un altro attributo che si potrebbe aggiungere al tag è pubdate: un booleano che serve a indicare se la data riportata fa riferimento alla pubblicazione dell'articolo in cui è contenuto il tag <time>. Se non è presente nessun elemento <article> la data fa riferimento all'intero documento.

ELEMENTO <PROGRESS>

- Indica lo stato di completamento di un'attività
- Un esempio d'uso potrebbe essere il caricamento di un file
- Se l'intervallo di completamento è noto, allora il browser dovrebbe visualizzare una barra di completamento che indichi lo stato dell'esecuzione di una determinata attività
- Nel caso in cui questo intervallo non sia noto, il browser dovrebbe visualizzare un oggetto che indichi che l'esecuzione dell'attività richiesta è in corso.

ELEMENTO <PROGRESS>

```
<section>
    <p>Caricamento: <progress id="mioLoader" max="100" value="10"><span>0</span>%</progress></p>
</section>
<input id=''start'' type="button" value="Start" onclick="updateProgress()">
<script type="text/javascript">
    var progressBar = document.getElementById('mioLoader'); var Value = 0;
    function updateProgress(newValue) {
        setTimeout('getValue()',10);
        progressBar.value = newValue; progressBar.getElementsByTagName('span')[0].textContent = newValue;
    }

    function getValue(){
        if (Value < 100){
            Value ++;  updateProgress(Value)
        } else{ Value = 0; }
   }
</script>
```

ELEMENTO <METER>

- Rappresenta una grandezza scalare all'interno di un intervallo noto, o un valore frazionario. Esempi d'uso potrebbero essere il Google rank o lo spazio su disco rimanente.
- Esempio <meter>
- <meter value="5" min=''0'' max="8">5 GB utilizzati su 8 GB</meter>
- In caso di mancato supporto, è opportuno inserire all'interno del tag del testo in modo tale da poter indicare comunque l'informazione.

ELEMENTO <MARK>

- Rende evidente una porzione di testo all'interno di un documento
- Un possibile utilizzo potrebbe essere quello di evidenziare le parole cercate da un utente in un testo.
- Esempio <mark>
- Senza <mark>plug in</mark> di terze parti il web potrebbe diventare per noi sviluppatori più democratico; con le nuove API HTML5 non abbiamo più bisogno di diversi <mark>plug in</mark> di terze parti che sino ad ora erano indispensabili per i contenuti multimediali.

Nuovi tag semantici					
<header>	9.0+	3.0+	3.1+	4.0+	9.0+
<footer>	9.0+	3.0+	3.1+	4.0+	9.0+
<section>	9.0+	3.0+	3.1+	4.0+	9.0+
<article>	9.0+	3.0+	3.1+	4.0+	9.0+
<nav>	9.0+	3.0+	3.1+	4.0+	9.0+
<aside>	9.0+	3.0+	3.1+	4.0+	9.0+

Nuovi tag strutturali					
<figure>	9.0+	3.0+	3.1+	4.0+	9.0+
<figcaption>	9.0+	3.0+	3.1+	4.0+	9.0+
<hgroup>	9.0+	3.0+	3.1+	4.0+	9.0+
<time>	9.0+	3.0+	3.1+	4.0+	9.0+
<meter>	No	6.0+	5.2+	6.0+	11.0+
<progress>	10.0+	6.0+	5.2+	6.0+	10.6+
<mark>	9.0+	3.0+	3.1+	4.0+	9.0+

WEB FORM

- I form, prima della definizione della nuova specifica, utilizzavano una modalità di interazione efficace ma un po' troppo rudimentale
- Le principali novità di HTML 5 dei web forms riguardano principalmente la semantica e l'aggiunta di nuovi attributi agli elementi con lo scopo di strutturare meglio la pagina e facilitarne l'accessibilità.

ATTRIBUTO TYPE

- Finora l'attributo type dell'elemento <input> supportava dieci valori (text, button, radio, ecc..), la nuova specifica ne supporta 23
- Fortunatamente se il valore dell'attributo non è riconosciuto, sarà interpretato con il valore di default type="text", quindi si potranno utilizzare i nuovi attributi anche se non supportati da tutti i browser.

INPUT TYPE SEARCH

- Permette una diversa visualizzazione in funzione della piattaforma utilizzata, diversa dalla classica visualizzazione di un campo testo

- Ad esempio, utilizzando Safari o Google Chrome, se scriviamo qualcosa nel campo, compare una piccola X sulla destra che, se cliccata, svuota il campo

```
<form name="ricerca" method="post" action="/search">
  <label> Cerca:
    <input type="search" autocomplete="on" placeholder="HTML, CSS, ..." name=‘‘keyword" required
maxlength="50">
    </label>
  <input type="submit" value="Ricerca">
</form>
```

INPUT TYPE EMAIL

- L'indirizzo di posta elettronica è un'informazione spesso richiesta durante la compilazione di un form, questo input type sarà utilizzato per creare un campo in cui si dovrà inserire questa informazione.

- Se l'indirizzo non è valido, sarà visualizzato un messaggio di errore. Inoltre i dispositivi mobili, nel momento in cui si abilita l'inserimento del testo nel campo, potrebbero modificare la tastiera, ad esempio visualizzando la chiocciola come accade con IPhone.

- Esempio email
```
<form name="registrazione" method="post" action="/send">
<label> Email:
<input type="email" name="email" autocomplete=‘‘on" placeholder="email@domain.ext"> </label>
<input type="submit" value=‘‘Registra">
</form>
```

- Questo controllo inoltre accetta l'attributo booleano multiple che se impostato a true permette l'inserimento di più indirizzi email separati da una virgola.

INPUT TYPE URL

- Un altro dato spesso richiesto nella compilazione di un form è l'indirizzo web con controllo validazione dato e nei browser mobile visualizza una tastiera che include il punto, la slash e l'estensione .com.

- Esempio url
```
<form name="registrazione" method="post" action="/send">
<label> Indirizzo web:
<input type="url" name="url" autocomplete=‘‘on" placeholder="http://mywebsite.com"> </label>
<input type="submit" value="Registra">
</form>
```

- Come per il type email anche il type url supporta l'attributo booleano multiple per l'inserimento di più indirizzi web

INPUT TYPE TEL

- Questo input type è stato aggiunto per permettere la per creazione di un campo adatto all'inserimento di numeri di telefono. A differenza dei campi email e url non è imposto un particolare formato perché a livello internazionale tra i numeri di telefono ci sono sostanziali differenze riguardanti la struttura. Per i mobile offre una tastiera dedicata per questo elemento visualizzando un tastierino numerico.

- Esempio tel

```
<form name="registrazione" method="post" action="/send">
<label> Numero di telefono:
    <input type="tel" name="tel">
 </label>
<input type="submit" value="Registra">
</form>
```

INPUT TYPE NUMBER

- Questo input type è stato pensato per l'inserimento di valori numerici. La visualizzazione prevista di questo elemento, rispetto a un campo di testo, presenta due frecce sul lato sinistro con il fine di poter incrementare o diminuire il valore inserito. Sia Android sia iOS di IPhone offrono una tastiera alternativa per questo campo. Inoltre si possono impostare tre attributi all'interno di questo elemento:
    - min: specifica il valore minimo dell'intervallo, il valore predefinito è 0
    - max: specifica il valore massimo dell'intervallo, il valore predefinito è 100
    - step: specifica il salto tra un valore ed il successivo nell'intervallo scelto
- Dopo aver scelto il valore, è previsto un controllo con il fine di verificare l'appartenenza del numero inserito all'insieme preimpostato.
- Esempio number

```
<form name="registrazione" method="post" action="/send">
  <label> Peso:
    <input type="number" name="peso" min="10" max="200" step="0.1" value="70.5">
  </label>
  <input type="submit" value="Registra">
</form>
```

INPUT TYPE RANGE

- Questo campo, simile al precedente, permette l'inserimento di numeri ma tramite l'utilizzo di uno slider
- La specifica stabilisce che quando s'intende utilizzare questo controllo non è importante conoscere con esattezza il valore scelto ma è sufficiente ottenere un valore indicativo. Anche questo campo prevede gli attributi min max e step descritti prima.
- Esempio range

```
<form name="intervista" method="post" action="/send">
<label> Come valuti questo sito?
<input type="range" name="peso" min="0" max="10" step="1"></label>
<input type="submit" value="Inoltra">
</form>
```

INPUT TYPE DATE

- Prima della definizione della nuova specifica per gestire le date era necessario ricorrere a supporti esterni, ora è molto più semplice grazie ai nuovi elementi di seguito descritti:
  - datetime: gestisce sia la data che l'ora (con fuso orario);
  - datetime-local: gestisce sia la data che l'ora (senza fuso orario);
  - date: gestisce le date;
  - month: gestisce i mesi;
  - week: gestisce le settimane;
  - time: gestisce l'ora.

INPUT TYPE DATE

- Per tutti questi input type abbiamo degli attributi specifici che sono:
  - min: rappresenta il minimo valore accettato;
  - max: rappresenta il massimo valore accettato;
  - step: rappresenta la granulosità dei valori accettati.

INPUT TYPE DATE

```
<label> Datetime <input type="datetime" name="mydatetime"> </label>
<label> Datetime-local <input type="datetime-local" name="mydatetime"> </label>
<label> Date <input type="date" name="mydatetime"></label>
<label> Month <input type="month" name="mydatetime"> </label>
<label> Week <input type="week" name="mydatetime"></label>
 <label> Time <input type="time" name="mydatetime"> </label>
```

INPUT TYPE COLOR

- Lo scopo di questo elemento è di permettere la selezione di un colore in modo semplice, tramite l'utilizzo di una tabella, ottenendo una stringa esadecimale di sei cifre che rappresenta il colore in formato RGB.

- Esempio color

```
<form name="intervista" method="post" action="/send">
<label> di che colore è la tua macchina? <input type="color" name="mycar"></label>
<input type="submit" value="Inoltra">
</form>
```

INPUT DATALIST

- Con le precedenti versioni di HTML per permettere la scelta di un'opzione da un elenco preimpostato veniva utilizzata una combinazione dei due tag <select> e <option>. Nel caso in cui si fosse dovuto dare la possibilità di immettere un'informazione non compresa nell'elenco, l'unica soluzione era aggiungere un ulteriore campo testo

- Con HTML 5 si può fare entrambe le cose utilizzano un unico controllo <input> impostando l'attributo list di quest'ultimo con l'id della lista (rappresentata dal tag <datalist>) delle opzioni selezionabili (ognuna definita all'interno di un tag <option>).

- Esempio input con datalist

```
<form name="intervista" method="post" action="/send"> <label> Che sistema operativo preferisci?
<input type="text" name="mood" list="OS"> <datalist id="OS">
                    <option value="Ubuntu">
                    <option value="Debian">
                    <option value="Kubuntu">
                </datalist>
            </label>
<input type="submit" value="Inoltra"> </form>
```

- Per una corretta visualizzazione l'attributo value di <option> non deve essere vuoto ed inoltre <option> non deve contenere l'attributo disabled.

NUOVI ATTRIBUTI

- Oltre agli attributi relativi ad alcuni degli input type descritti (multiple, min, max, step e list) ne sono stati introdotti altri, applicabili a tutti gli elementi <input>.
- **Attributo autocomplete:** Di default il valore di questo attributo è impostato a on, ovvero i browser compilano i campi di un form in funzione di altre compilazioni già effettuate. Spesso risulta comoda questa funzionalità poiché velocizza il processo d'inserimento dei dati, se però i campi in questione sono relativi a dati sensibili (come le password) potrebbe convenire impostare il valore dell'attributo a off dato che i dati vengono memorizzati all'interno del browser con la conseguenza di una più facile accessibilità da parte di persone o software sgraditi.
- **Attributo autofocus:** L'attributo booleano autofocus serve ad impostare il focus su un determinato elemento di un form dopo il caricamento di una pagina. Nel caso in cui questo attributo sia stato applicato all'interno di più campi, il focus sarà assegnato all'ultimo elemento a cui è applicato. L'utilizzo di questo attributo all'interno di pagine lunghe potrebbe risultare scomodo in quanto la pressione della barra spaziatrice agirà sul controllo mentre alcuni utenti la utilizzano per scorrere velocemente la pagina
- Nel caso in cui autofocus non sia supportato, si potrebbe ottenere lo stesso effetto tramite un semplice script.

```
if(!Modernizr.input.autofocus){ Document.getElementById(''password").focus() }
```

NUOVI ATTRIBUTI

- **Attributo form:** Prima della definizione della nuova specifica un elemento di un form doveva necessariamente essere racchiuso tra i due tag <form></form>. Ora se a un elemento esterno a un form è aggiunto questo attributo impostando il suo valore con l'id di uno o più form (separati da virgola), continuerà comunque ad esserne parte integrante.

- **Attributo pattern:** Con questo attributo le espressioni regolari potranno essere gestite con HTML 5. Il suo valore dovrà quindi essere un'espressione regolare valida. All'interno dell'attributo title si potrebbe inserire una spiegazione del formato richiesto per la compilazione dell'elemento. Nel caso in cui i vincoli non siano rispettati, il browser blocca l'invio del form avvisando l'utente.

- **Attributo placeholder:** Il valore testuale di questo attributo è visualizzato all'interno di un input o di una textarea. Il testo scompare nel momento in cui il campo assume il focus oppure quando l'utente lascia il focus dell'elemento senza averlo compilato. Può essere utilizzato per fornire un suggerimento all'utente riguardante la compilazione, oppure può essere utilizzato come semplice etichetta.

NUOVI ATTRIBUTI

- **Attributo required:** Per rendere obbligatoria la compilazione di un campo contenuto in un form è sufficiente aggiungere questo attributo booleano. Se l'utente tenta di inviare i dati senza aver compilato tutti i campi contenenti questo attributo il browser segnala l'errore.

- **Attributo novalidate:** Questo attributo si applica all'elemento <form> e permette di saltare tutte le validazioni dei tag che da esso discendono.

```
    <form name="registrazione" method="post" action="/send">
        <fieldset>
            <legend>Registrati</legend>
            <p>
                <label for="nome">Nome*</label>
                <br>
                <input id="nome" type="text" name="nome" autofocus placeholder="inserisci nome e cognome"
size="40" required>
            </p>
            <p>
                <label for="email">Email*</label>
                <br>
                <input id="email" type="email" name="email" placeholder="tua@mail.it" size="40" required>
            </p>
            <p>
                <label for="pass">Password*</label>
                <br>
                <input id="pass" type="password" name="pass" placeholder="almeno 5 caratteri" size="40" required
pattern="(\s*(\S)\s*){5,} " autocomplete="off">
            </p>
            <p>
                <label for="telefono ">Telefono</label>
                <br>
                <input id="telefono " type="tel" name="telefono" placeholder="inserisci un recapito telefonico "
size="40 ">
            </p>
            <p>
                <input type="submit" name="registra" value="Registra">
            </p>
        </fieldset>
    </form>
```

VALIDAZIONE DATI E CONTROLLO

- Ovviamente tutti i controlli eseguiti lato client è necessario rieseguirli lato server dato che i controlli html sono facili da eludere
- La finalità dei controlli lato client è di migliorare la struttura e di facilitare l'accessibilità da parte degli utenti.

Dott. Antonio Giovanni Lezzi

CANVAS

- Il tag <canvas> riproduce l'effetto di una tela sulla quale disegnare con il "pennello"
- Esistono delle API del Canvas 2D, ovvero un insieme di funzioni JavaScript che permettono un rendering dinamico di immagini bitmap
- Saranno presentati solamente i metodi legati al contesto di disegno 'Canvas 2d', ma esiste già una versione sperimentale di contesto Canvas 3d che sottende un set di API completamente diverso, basato sulle OpenGL ES 2.0 e detto WebGL.

CANVAS

- Gli unici attributi specifici di questo elemento sono width e height che permettono di impostare le dimensioni della "tela". Se non impostati i valori sono 300 pixel di larghezza e 150 pixel di altezza

- Gli elementi all'interno dei tag <canvas></canvas> sono visualizzati solamente se il tag non è supportato dal browser, in questo caso si potrebbe inserire un'immagine come alternativa all'elemento.

- Prima di cominciare a disegnare è necessario ottenere un riferimento all'oggetto che rappresenta l'elemento con il metodo JavaScript getElementById() e verificare che il browser lo supporti. (Modernizr si può testare il supporto)

CANVAS TEST

- <script>
  var canvas = document.getElementById("tela")
  if(Modernizr.canvas){
  //si scrive sul canvas
  } </script>

- Dopo aver testato il supporto, utilizzando il metodo getContext(), si ottiene un riferimento al contesto bidimensionale nell'ambito del quale operare. L'unico argomento che, a oggi, accetta è 2d

- L'oggetto che si ottiene si può immaginare come un piano cartesiano dove l'origine è l'angolo in alto a sinistra e il valore dell'asse delle ascisse e delle ordinate aumenta man mano che ci si allontana dall'origine spostandosi rispettivamente verso destra e verso il basso.

CANVAS: METODI E PROPRIETÀ

- Le tabelle di seguito riportate contengono una breve descrizione dei metodi e delle proprietà dell'oggetto rappresentante l'elemento <canvas>
- I metodi possono accettare uno o più argomenti.

COLORI E STILI

Proprietà	*Descrizione*
`fillStyle`	Imposta o restituisce il colore, il gradiente o il modello del disegno.
`strokeStyle`	Imposta o restituisce il colore, il gradiente, o il modello dei tratti.
`shadowColor`	Imposta o restituisce il colore usato per le ombre.
`shadowBlur`	Imposta o restituisce il livello di sfocatura per le ombre.
`shadowOffsetX`	Imposta o restituisce la distanza orizzontale dell'ombra dalla forma.
`shadowOffsetY`	Imposta o restituisce la distanza verticale dell'ombra dalla forma.

Metodo	*Descrizione*
`createLinearGradient()`	Crea un gradiente lineare (da usare sul contenuto del Canvas).
`createPattern()`	Ripete un elemento specificato nella direzione specificata.
`createRadialGradient()`	Crea un gradiente radiale/circolare (da usare sul contenuto del Canvas).
`addColorStop()`	Specifica i colori e le posizioni di arresto in un oggetto gradiente.

COLORI E STILI

Stili delle linee

Proprietà	*Descrizione*
`lineCap`	Imposta o restituisce lo stile delle punte di una linea.
`lineJoin`	Imposta o restituisce il tipo di angolo creato, quando due linee si incontrano.
`lineWidth`	Imposta o restituisce la larghezza di una linea.
`miterLimit`	Imposta o restituisce l'angolazione oltre la quale la punta della giunzione ad angolo di un segmento viene troncata.

Rettangoli

Metodo	*Descrizione*
`rect()`	Creazione di un rettangolo.
`fillRect()`	Disegna un rettangolo "pieno".
`strokeRect()`	Disegna un rettangolo (senza riempimento).
`clearRect()`	Cancella i pixel specificati all'interno di un rettangolo dato.

COLORI E STILI

Percorsi

Metodo	*Descrizione*
`fill()`	Riempie il disegno corrente (percorso).
`stroke()`	Disegna il percorso che è stato definito.
`beginPath()`	Inizia un percorso, o reimposta il percorso corrente.
`moveTo()`	Consente di spostare il percorso in un punto specificato del Canvas.
`closePath()`	Crea un nuovo percorso dalla fine del punto del percorso corrente.
`lineTo()`	Aggiunge un nuovo punto e crea una linea da quel punto all'ultimo punto specificato del Canvas.
`clip()`	Taglia una regione di qualsiasi forma e dimensione dalla tela originale.
`quadraticCurveTo()`	Crea una curva quadratica di Bézier.
`bezierCurveTo()`	Crea una curva di Bézier cubica.
`arc()`	Crea un arco/curva (utilizzato per creare cerchi, o parti di cerchi).
`arcTo()`	Crea un arco/curva tra due tangenti.
`isPointInPath()`	Restituisce `true` se il punto specificato è nel percorso corrente, altrimenti `false`.

COLORI E STILI

Trasformazioni

Metodo	*Descrizione*
`scale()`	Applica una scala al disegno corrente.
`rotate()`	Ruota il disegno corrente.
`translate()`	Trasla il disegno corrente.
`transform()`	Sostituisce la matrice di trasformazione corrente del disegno.
`setTransform()`	Reimposta la matrice di trasformazione alla matrice identità.

Immagini

Metodo	*Descrizione*
`drawImage()`	Disegna un'immagine, tela, o un video nel Canvas.

COLORI E STILI

Testi

Proprietà	*Descrizione*
`font`	Imposta o restituisce le proprietà del font.
`textAlign`	Imposta o restituisce l'allineamento del testo.
`textBaseline`	Imposta o restituisce la baseline del testo.

Metodo	*Descrizione*
`fillText()`	Disegna il testo creato nel Canvas.
`strokeText()`	Disegna il testo creato nel Canvas (senza riempimento).
`measureText()`	Restituisce un oggetto che contiene la larghezza del testo specificato.

COLORI E STILI

Manipolazione dei pixel

Proprietà	*Descrizione*
`width`	Restituisce la larghezza di un oggetto `ImageData`.
`Height`	Restituisce l'altezza di un oggetto `ImageData`.
`Data`	Restituisce un oggetto che contiene i dati di un oggetto `ImageData` specificato.

Metodo	*Descrizione*
`createImageData()`	Crea un nuovo oggetto vuoto `ImageData`.
`getImageData()`	Restituisce un oggetto `ImageData` contenente le informazioni dei pixel del rettangolo specificato sul Canvas.
`putImageData()`	Inserisce i pixel di un oggetto `ImageData` nel Canvas.

CANVAS ESEMPIO

- Disegna un rettangolo colorato con ombreggiatura, all'interno dell'elemento <canvas>

```
  <body onload="initCanvas()">
    <canvas id="tela" width="250" height="250">Il browser non supporta l'elemento canvas</canvas>
</body>
<script type="text/javascript ">
    function initCanvas() {
        var canvas = document.getElementById("tela");
        var contesto = canvas.getContext("2d");
        contesto.shadowColor = '#003300';
        contesto.shadowOffsetX = 10;
        contesto.shadowOffsetY = 10;
        contesto.shadowBlur = 20;
        contesto.fillStyle = "rgba(50, 255, 255, 0.2)";
        contesto.rect(10, 10, 170, 150);
        contesto.stroke();
        contesto.fill();
        contesto.fillStyle = "rgba(255, 255, 255, 1)";
        contesto.font = "20px Georgia";
        contesto.fillText("Hello World!", 20, 50);
    }
</script>
```

ALTRE TECNOLOGIE

<canvas>	9.0+	2.0+	3.1+	4.0+	9.0+

VIDEO E AUDIO

- Tra le novità che HTML5 offre rispetto alle versioni precedenti, vi è la possibilità di usufruire dei contenuti multimediali utilizzando due elementi: <video> e <audio>
- L'introduzione della possibilità di riprodurre filmati o tracce audio in un browser senza l'ausilio di plug-in di terze parti, come Flash, rappresenta di per sé una grossa novità.
- Questi due elementi sin dall'inizio sono quelli che hanno destato maggior interesse riguardo a questa nuova tecnologia, non solo nei confronti di piccoli gruppi di programmatori web ma di grandi aziende come Apple, YouTube e Microsoft.

VIDEO E AUDIO

- In precedenza l'inserimento di questo tipo di contenuti all'interno di una pagina HTML richiedeva, il più delle volte, l'utilizzo di applicativi esterni (e non standard) come Adobe Flash
- Inizialmente furono sollevate alcune perplessità riguardanti il video e l'audio di HTML 5 perché inadatti a proteggere adeguatamente i contenuti e ancora impreparati all'integrazione di inserzioni pubblicitarie risolto con le librerie JavaScript
- Tali librerie mirano ad eliminare problemi di compatibilità con vecchi browser che non supportano tali elementi. Sono state create anche altre librerie, come PopCorn.js, con il fine di migliorare l'accessibilità dei contenuti fornendo ulteriori interfacce agli utenti.

ELEMENTO <VIDEO>

- Nella sua forma più semplice il tag video ricorda il tag img: <video src=''video.ogv''>
- Sarebbe fin troppo bello se fosse sufficiente questa dichiarazione per integrare un video in un sito web
- Purtroppo, ancora una volta, i principali browser non supportano le stesse funzionalità
- In questo caso l'argomento riguarda i codec per la riproduzione video, quindi prima della presentazione delle funzionalità dell'elemento, è riportata una tabella con i formati compatibili dei vari browser

ELEMENTO <VIDEO>

- MP4 = documenti MPEG 4 con codec video H264 e codec audio AAC
- WebM = documenti WebM con codec video VP8 e codec audio Vorbis
- Ogg = documenti Ogg con codec video Theora e codec audio Vorbis

Formato					
MP4	9.0+	NO	3.2+	4.0+	No
WebM	No	4.0+	No	6.0+	10.6+
Ogg	No	3.5+	No	4.0+	10.5+

ELEMENTO <VIDEO>

- **Elemento <source>:** Per riprodurre un video in tutti i principali browser bisognerà associare all'elemento più formati dello stesso video. Questo si può fare grazie all'elemento <source> ripetuto più volte all'interno dei tag <video></video>. Gli attributi di <source> sono: src il cui valore corrisponde al percorso associato al video e type che permette di specificare la codifica video.

- **Attributo controls:** Aggiungendo questo attributo booleano vengono visualizzati i classici bottoni per la riproduzione video con il fine di poter tenere sotto controllo l'esecuzione. Google Chrome, Safari e FireFox tramite l'utilizzo di un pulsante permettono di visualizzare il video a schermo intero.

- **Attributo poster:** L'immagine visualizzata prima dell'esecuzione del video corrisponde al primo fotogramma. Se si desidera cambiarla è sufficiente impostare il valore dell'attributo poster all'interno del tag <video> con il percorso dell'immagine.

ELEMENTO <VIDEO>

- **Attributo preload:** Questo attributo può assumere tre valori:
  - **none**: indica allo user-agent di non effettuare nessun scaricamento preventivo del contenuto del file video, il buffer inizierà quindi ad essere riempito alla prima pressione del tasto play.
  - **metadata**: prima dell'esecuzione vengono recuperate solamente le informazioni essenziali come durata, dimensione, ecc...
  - **auto**: impostando questo valore si presuppone che l'utente voglia prendere visione del video, quindi il browser caricherà preventivamente il video.
- **Attributo autoplay** Come si può intuire questo attributo booleano consente l'esecuzione automatica del video.

ELEMENTO <VIDEO>

- **Attributo loop:** Anche questo è un attributo booleano che se abilitato, il video inizia una nuova esecuzione una volta che è giunto al termine.
- **Attributo muted:** Attributo booleano che disabilita preventivamente l'audio, ma permette comunque all'utente di riabilitarlo (a patto che siano presenti i controlli).
- **Attributi width e height:** Permettono di definire la larghezza e l'altezza del video. Se non impostati, saranno stabiliti dal browser in funzione dell'immagine associata all'attributo poster, se presente, oppure in funzione del primo fotogramma del video.
- **Elemento <track>:** Nessuno dei principali browser supporta questo elemento che contenuto all'interno dei tag <video></video> dovrebbe contenere elementi testuali che si possono combinare con il video (ad esempio i sottotitoli). Per il momento si può sostituire questa funzione con la libreria Popcorn.js.

ELEMENTO <VIDEO>

```
<video controls autoplay>
   <source src="http://www.quirksmode.org/html5/videos/big_buck_bunny.mp4" type="video/mp4">
   <source src="http://www.quirksmode.org/html5/videos/big_buck_bunny.ogv" type="video/ogg" >
   <source src="http://www.quirksmode.org/html5/videos/big_buck_bunny.webm" type="video/webm">
</video>
```

ELEMENTO <AUDIO>

- Come per il tag <video> anche il tag <audio>, non esiste un formato comune a tutti i browser supportano un formato comune, <audio src="audio.oga" controls>

Formato					
MP3	9.0+	NO	4.0+	4.0+	NO
Wav	No	4.0+	4.0+	6.0+	10.6+
Ogg	No	3.5+	NO	4.0+	10.5+

ELEMENTO <AUDIO>

- **Elemento <source>**: Anche il tag <audio>, per risolvere le problematiche relative all'ascolto della traccia audio su tutti i browser, può contenere questo elemento permettendo di associare ad una traccia diversi formati.
- Gli attributi dell'elemento audio sono: **src, preload, autoplay, loop e controls** e le loro funzionalità sono analoghe a quelle dell'elemento <video>
- Bisogna notare che in assenza dell'attributo controls i controlli audio non sono visibili di conseguenza tutto l'elemento sarà invisibile.

ELEMENTO <AUDIO>

```
<audio controls autoplay>
   <source src="http://alexandre.alapetite.fr/doc-alex/html5- audio/DTMF1000ms.mp3" type="audio/
mp3">
   <source src="http://alexandre.alapetite.fr/doc-alex/html5- audio/DTMF1000ms.oga" type="audio/ogg"
onerror="alert('Impossibile riprodurre il file audio')" >
</audio>
```

ELEMENTO <AUDIO>

- **JavaScript:** L'elemento <audio> dispone anche di un proprio costruttore JavaScript, ecco la sintassi:
- var audio = new Audio("file_musicale.mp3");
- In questo modo è possibile utilizzare musiche ed effetti audio senza necessariamente creare l'elemento HTML né ricorrere a display:none o similari.

APPLICAZIONI OFFLINE

- HTML5 introduce le API offline che permettono la memorizzazione in locale di documenti per la visualizzazione di pagine web anche in assenza di una connessione internet
- Le informazioni riguardanti i file necessari alla navigazione offline sono definite all'interno di un file dedicato con estensione manifest.
- Questa novità, ad esempio, potrebbe risultare molto utile per la navigazione web tramite dispositivi mobili in aree in cui non c'è una buona copertura od è completamente assente

FILE MANIFEST

- È necessario aggiungere l'attributo manifest all'elemento <html> segnalando così allo user agent l'esistenza del file con le specifiche per una consultazione offline
- Il valore di questo attributo deve corrispondere al percorso del file contenente l'elenco dei documenti per i quali si richiede la memorizzazione
- Il file in questione dovrà essere MIME type text/cache-manifest, quindi si dovrà inoltre impostare il server in modo tale da servire il file con il MIME type corretto

FILE MANIFEST

- Esempio file manifest

```
CACHE MANIFEST
#commento
CACHE:
/pagina.html
/css/cssPagina.css
NETWORK:
/inviaDocumento.aspx
FALLBACK:
news/* /offline.html /contatti.aspx /contatti.html
```

FILE MANIFEST

- **Sezione CACHE:** Se vengono inseriti indirizzi di alcune risorse senza dichiarare una sezione, implicitamente viene applicata la sezione CACHE. Questa sezione rappresenta le risorse che si devono rendere disponibili nel caso in cui non si riesca a stabilire una connessione con il server

- **Sezione NETWORK:** Questa sezione, come CACHE, contiene un'unica risorsa per riga. In questo caso però gli indirizzi indicati si riferiscono a risorse per le quali il reperimento richiede una connessione internet attiva.

- **Sezione FALLBACK:** per ogni riga sono indicate due risorse distinte. La prima indica quella disponibile nel caso di possibile accesso alla rete. La seconda rappresenta invece l'alternativa di cui disporre in caso di offline.

  Nell'esempio la prima riga di questa sezione indica che tutto il contenuto della cartella news, nel caso di mancato accesso alla rete, sarà sostituito con la pagina offline.html.

FILE MANIFEST: COMMENTI

- Il file manifest include anche la possibilità di inserire dei commenti, questo si può fare solamente inserendo all'inizio della riga da commentare il simbolo '#'
- Questa limitazione è imposta per evitare ambiguità che potrebbero sorgere nel caso in cui il commento sia inserito in una riga contenente un indirizzo web

MANIFEST

- Si possono descrivere 4 casi distinti durante la visitazione di un sito web che offre tale servizio.
- **Prima visita:** Se il browser visita una pagina web nella quale sarà rilevato un riferimento a un file manifest per la prima volta, significa che l'utente non conserva una copia in locale di tale file. A questo punto il browser scarica il file, lo interpreta e richiede al server tutte le risorse indicate al suo interno che saranno necessarie in caso di mancato accesso alla rete. Le fasi che compongono il recupero delle informazioni associate a questo caso (prima visita) sono demarcate dai seguenti eventi:
  - **checking**: analisi del file manifest.
  - **downloading**: scaricamento di tutti i file.
  - **progress**: per ogni risorsa da scaricare viene scatenato tale evento.
  - **cached**: operazioni concluse.

MANIFEST

- **Visita successiva (senza modificazione del file manifest):** Il browser rileva che è già stato scaricato un file manifest associato a questo sito web quindi, invece di rieseguire le operazioni corrispondenti alla prima visita, si limita al controllo del file locale e del file server per la rilevazione di eventuali modifiche. In questo caso non sarà individuata alcuna modifica quindi non seguiranno altre fasi. Gli eventi di questo processo sono:
  - **checking**: analisi del file manifest.
  - **noupdate**: evento che comunica l'assenza di differenze tra i due file manifest.

MANIFEST

- **Visita successiva con modifica del file manifest:** Analogamente al caso precedente sarà rilevata la presenza del file manifest. In questo caso però durante l'analisi saranno rilevate modifiche, quindi il file sarà sostituito con quello presente sul server. In seguito saranno scaricate in locale tutte le risorse elencate. Gli eventi di questo scenario sono i seguenti:
  - **checking**: analisi del file manifest.
  - **downloading**: scaricamento dei file causato dalla differenza dei due file.
  - **progress**: evento scatenato per ogni risorsa da scaricare in locale.
  - **updateready**: segnala che tutte le risorse sono state nuovamente scaricate in locale.

MANIFEST

- **Mancanza di risorse definite all'interno del file manifest:** Durante lo scaricamento dei file necessari potrebbe accadere che una o più risorse non siano raggiungibili. In questo caso sarà scatenato un evento con il fine di comunicare un errore durante l'esecuzione. Gli eventi associati a questo caso saranno:
  - **checking**: analisi del file manifest.
  - **downloading**: notifica scaricamento delle risorse.
  - **progress**: evento scatenato per ogni risorsa da scaricare in locale.
  - **error**: evento scatenato a causa dell'impossibilità di scaricare una determinata risorsa in locale.

MANIFEST: ESEMPIO

```
CACHE MANIFEST
# v1 2011-08-14
# This is another comment
index.html
cache.html
style.css
image1.png

# Use from network if available
NETWORK:
network.html

# Fallback content
FALLBACK:
/ fallback.html
```

```
<html manifest="manifest.file">
  ...
</html>
```

MANIFEST

Applicazioni web offline	10.0+	3.5+	4.0+	4.0+	10.6+

WEB STORAGE

- Le WebStorage API permettono il salvataggio locale d'informazioni concernenti la navigazione web
- Nascono per risolvere due principali problematiche relative ai cookie: Lo spazio per la memorizzazione, che passa dai restrittivi 4KB a 5MB, e l'impossibilità di avere due sessioni distinte in un browser relative allo stesso dominio
- Prima del Web Storage di HTML 5 ci furono altri tentativi per la risoluzione di queste due problematiche senza però riscontrare successo perché basate su tecnologie proprietarie (come userData di Microsoft) o perché necessitavano l'installazione di componenti aggiuntivi (flash script).
- Fortunatamente tutti i browser supportano questo nuovo membro della specifica, di conseguenza molto probabilmente, in poco tempo, il Web Storage prenderà il sopravvento sui cookie.

WEB STORAGE: LOCAL

- localStorage: Mantiene il comportamento dei cookie essendo comune a tutte le finestre del browser che condividono lo stesso dominio
- I dati persistono anche dopo la chiusura del browser. Permette ad esempio di aggiungere articoli nel carrello e di poter completare l'acquisto nei giorni successivi
- L'utilizzo ideale quindi riguarda applicazioni in cui un processo può essere interrotto per poi essere ripreso anche a distanza di giorni.
- Entrambi sono inoltre stati studiati per ospitare molti più dati, fino a 5MB, e a differenza dei cookie i dati non sono scambiati sistematicamente tra client e server, generando richieste meno corpose.

WEB STORAGE: LOCAL

```html
<!DOCTYPE html>
<html>
<head>
    <script>
        function clickCounter() {
            if (typeof(Storage) !== "undefined") {
                if (localStorage.clickcount) {
                    localStorage.clickcount = Number(localStorage.clickcount) + 1;
                } else {
                    localStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Contatore: " + localStorage.clickcount;
            } else {
                document.getElementById("result").innerHTML = "Il browser non supporta il web storage...";
            }
        }
    </script>
</head>
<body>
    <p><button onclick="clickCounter()" type="button">Contatore</button></p>
    <div id="result"></div>
    <p>Clicca il pulsante per vedere il contatore aumentare.</p>
    <p>Chiudi il browser e prova a verificare se il contatore riprende a contare dall'ultimo numero.</p>
</body>
</html>
```

WEB STORAGE: SESSION

- I problemi legati ai cookie sono stati risolti con l'introduzione di due nuovi oggetti: sessionStorage
- Consente di avere un meccanismo di persistenza dei dati distinto per ogni sessione di navigazione in ogni finestra, o scheda, del browser eliminando i dati alla chiusura
- Usando sessionStorage ad esempio sarebbe quindi possibile coordinare l'apertura contemporanea di due distinti account GMail sullo stesso browser
- Questo oggetto è da utilizzare nel caso in cui le informazioni da memorizzare sono di carattere temporaneo, quindi per processi di breve durata.

WEB STORAGE: SESSION

```html
<!DOCTYPE html>
<html>
<head>
    <script>
        function clickCounter() {
            if (typeof(Storage) !== "undefined") {
                if (sessionStorage.clickcount) {
                    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
                } else {
                    sessionStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Contatore: " + localStorage.clickcount;
            } else {
                document.getElementById("result").innerHTML = "Il browser non supporta il web storage...";
            }
        }
    </script>
</head>
<body>
    <p><button onclick="clickCounter()" type="button">Contatore</button></p>
    <div id="result"></div>
    <p>Clicca il pulsante per vedere il contatore aumentare.</p>
    <p>Chiudi il browser e prova a verificare se il contatore riprende a contare dall'ultimo numero.</p>
</body>
</html>
```

WEB STORAGE

- Entrambi gli oggetti, sessionStorage e localStorage, implementano l'interfaccia Storage che raggruppa un insieme di metodi comuni ai due oggetti
- La memorizzazione è consentita solo per valori testuali e avviene tramite una coppia chiave valore
- I metodi per accedervi sono: setItem() per valorizzare la chiave e getItem() per ottenerne il valore
- In realtà, in formato stringa, si possono anche memorizzare valori numerici interi o decimali che in seguito, con l'ausilio dei metodi parseInt() o parseFloat(), si possono riottenere in formato numerico.

JSON

- La possibilità di memorizzare dati o comunque di rappresentarli in maniera da poterli trasferire tra programmi anche diversi è fondamentale nella programmazione

- In questo ambito si può far rientrare la serializzazione, cioè il processo di trasformazione di un oggetto in un formato facilmente memorizzabile e/o trasmissibile, ed il relativo processo inverso, cioè la deserializzazione

JSON

- Negli ultimi tempi si sta sempre più affermando la rappresentazione di oggetti in JSON (JavaScript Object Notation)
- Come indica il nome, questo formato nasce dalla notazione letterale degli oggetti di JavaScript, ma è diventato ormai uno standard de facto indipendente dal linguaggio di programmazione e dalla piattaforma.
- Sostanzialmente JSON utilizza un sottoinsieme della notazione letterale degli oggetti di JavaScript per rappresentarli sotto forma di stringa

```
{nome: "Mario", cognome: "Rossi"}
```

- viene rappresentato in JSON con la seguente stringa:

```
'{nome: "Mario", cognome: "Rossi"}'
```

- Da notare che abbiamo utilizzato il singolo apice come delimitatore di stringa per non interferire con i doppi apici delle stringhe interne. Anche un array può essere rappresentato nella notazione JSON

```
[1, "test", true]
```

- viene rappresentato come

```
'[1, "test", true]'
```

JSON

- Naturalmente è possibile rappresentare array di oggetti ed oggetti che prevedono array o altri oggetti come proprietà. La notazione JSON però non prevede la rappresentazione dei metodi di un oggetto.
- Naturalmente l'utilizzo di questo formato per la serializzazione e deserializzazione di oggetti in JavaScript è del tutto immediato, dal momento che in linea di principio sarebbe sufficiente utilizzare l'eval() della stringa per ottenere l'oggetto rappresentato
- In pratica però è opportuno effettuare alcuni controlli per evitare errori e l'introduzione di codice indesiderato.
- A questo scopo, a partire dalla versione 5 di ECMAScript è stato introdotto un parser nativo per la serializzazione e deserializzazione in notazione JSON

JSON

- Il parser è implementato dall'oggetto JSON che prevede due metodi specifici: parse() e stringify().

- Il metodo parse() prende in input una stringa JSON e genera il corrispondente oggetto JavaScript:

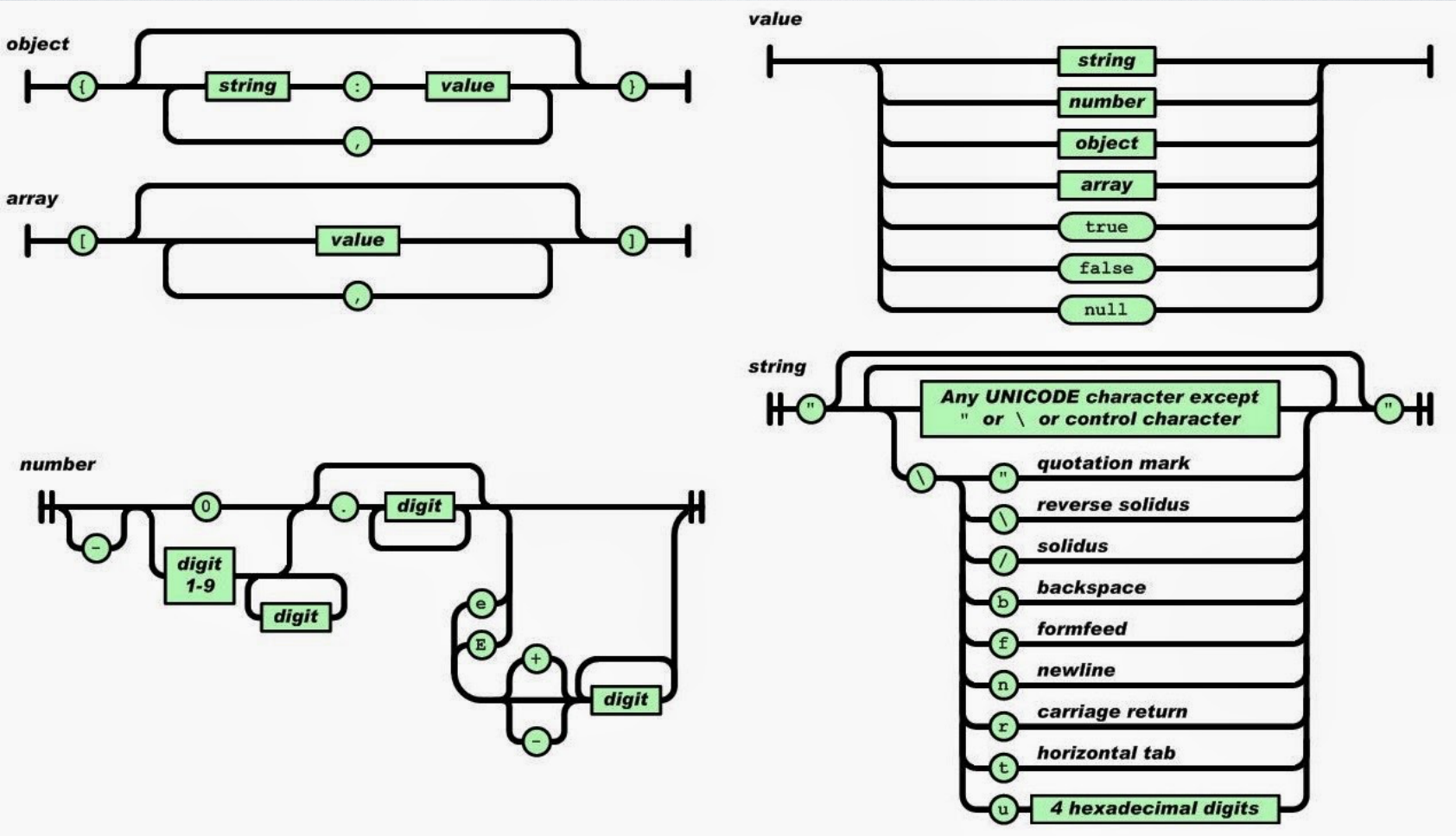
```
var marioRossi = JSON.parse('{nome: "Mario", cognome: "Rossi"}');
```

- Il metodo stringify() genera la rappresentazione JSON dell'oggetto passato come argomento:

```
var jsonMarioRossi = JSON.stringify({nome: "Mario", cognome: "Rossi"});
```

- Se l'oggetto da rappresentare contiene dei metodi, questi vengono semplicemente ignorati dal parser

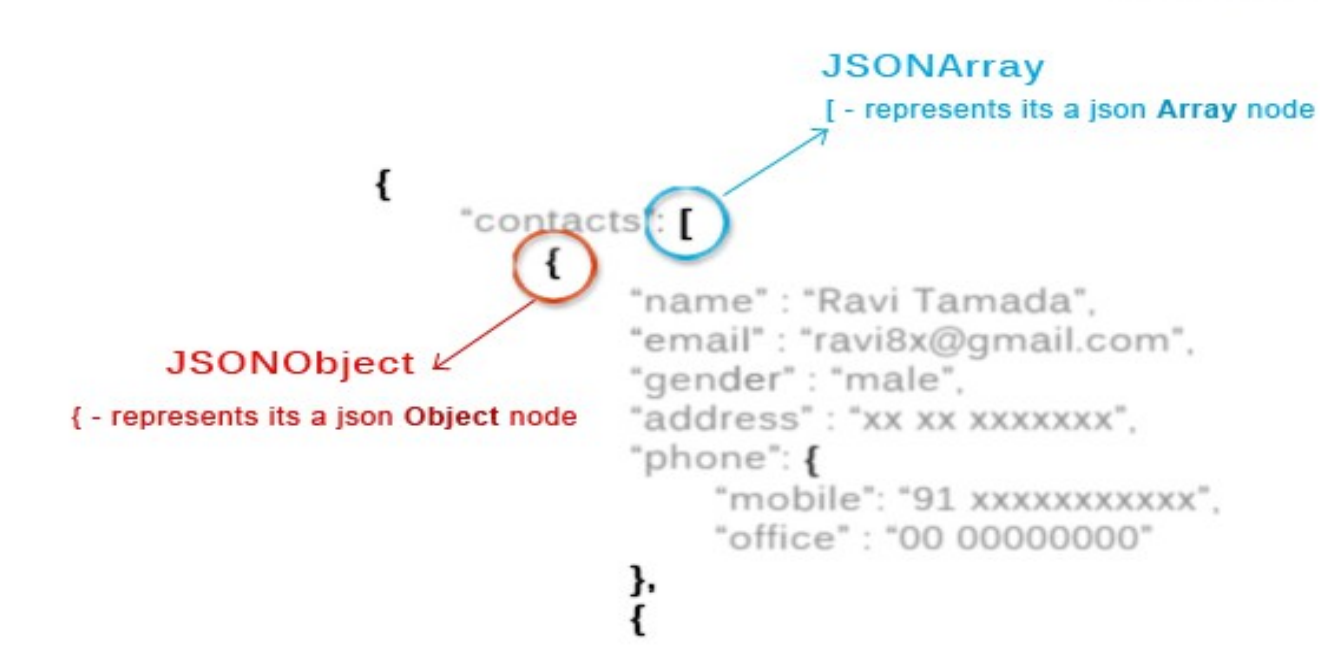
COME È UN JSON



FORMATI JSON

- I tipi di dati supportati da questo formato sono:
- **booleani** (true e false)
- **numeri:** interi, reali, virgola mobile
- **stringhe** racchiuse da doppi apici "
- **array:** sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre [ ]
- **array associativi**: sequenze coppie chiave-valore separate da virgole racchiuse in parentesi graffe

STRUTTURA FORMATO SON



ESEMPIO JSON

- esempio Json di un array di oggetti:

```
[
   {
      "nome":"Roberto",
      "eta":52
   },
   {
      "nome":"Antonio",
      "eta":32
   },
   {
      "nome":"Francesca",
      "eta":40
   }
]
```

WEB STORAGE: JSON

- Nel caso in cui si richieda la memorizzazione di un oggetto, si può utilizzare JSON (JavaScript Object Notation), un semplice formato utilizzato per lo scambio di dati
- La funzione JSON.stringfy() permette di ottenere una rappresentazione stringa di un oggetto
- la funzione inversa è JSON.parse() che permette di riottenere il formato originale

WEB STORAGE: JSON

```
<!DOCTYPE html>
<html>
<body>
<h2>Creare un oggetto da JSON</h2>

<p id="demo"></p>
<script>
var text = '{"employees":[' +
'{"firstName":"Antonio","lastName":"Lezzi" },' +
'{"firstName":"Mario","lastName":"Rossi" },' +
'{"firstName":"Giuseppe","lastName":"Verdi" }]}';

obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

SESSION STORAGE: ESEMPIO

```
<script type="text/javascript">
    var oggetto = {
        "nome": "Antonio",
        "cognome": "Lezzi",
        "email": "email@gmail.com"
    };
    stringaOggetto = JSON.stringify(oggetto);
    sessionStorage.setItem("credenziali", stringaOggetto);
    stringaOggettoRipristinato = sessionStorage.getItem("credenziali");
    var oggettoRipristinato = JSON.parse(stringaOggetto);
    alert(oggettoRipristinato["email"]);
</script>
```

- Nell'esempio si poteva procedere in modo analogo con l'oggetto localStorage.
- Per quanto riguarda la memorizzazione di variabili stringhe i metodi sessionStorage.setItem("chiave","valore") e temp=sessionStorage.getItem("chiave") si possono sostituire rispettivamente con sessionStorage.chiave="valore" e temp=sessionStorage.chiave.

WEB STORAGE: INTERCETTARE MODIFICHE

- Nel caso in cui fosse necessario rilevare eventuali modifiche alle variabili memorizzate, si può intercettare l'evento storage, notificato ogni qualvolta i dati sono modificati
- Questo evento mette a disposizione una serie di proprietà, accessibili solamente in lettura, contenenti alcune informazioni relative alla modifica effettuata:
  - **key**: la chiave dell'oggetto modificato.
  - **oldValue**: il valore associato alla chiave prima della modifica.
  - **newValue**: il valore associato alla chiave dopo la modifica.
  - **url**: l'indirizzo del documento in cui ha avuto luogo la modifica.
  - **storageArea**: riferimento all'oggetto, che rappresenta i dati, nel quale è stata rilevata la modifica.

WEB STORAGE: ESEMPIO EVENTO

```
<script type="text/javascript">
    if (window.addEventListener) {
        window.addEventListener("storage", modificaDati, false);
    } else if (window.attachEvent) { //per Internet Explorer 6,7 e 8
        Window.attachEvent("onstorage", modificadati);
    }

    function modificaDati(event) {
        var dataStorage = event || window.event;
        alert("chiave: " + dataStorage.key + " – valore:" +
dataStorage.newValue);
    }
</script>
```

WEB STORAGE: PRIVACY

- I cookie in passato sono stati visti come una minaccia alla privacy, molti utenti hanno deciso di disabilitarli con la conseguente riduzione dei servizi online.
- Senza cookie ad esempio non si riesce ad utilizzare Gmail, Twitter e molte altre applicazioni web. Il Web Storage amplificando le potenzialità dei cookie rende ancora più sensibile il tema della privacy
- Ad esempio se un ente in grado di distribuire i suoi contenuti su più siti assegnasse un identificativo univoco salvato nell'area di storage locale di un utente, potrebbe essere in grado di tracciare i suoi movimenti nel web
- Così facendo sarebbe più semplice focalizzare i suoi interessi in modo tale da poter predisporre della pubblicità mirata.

WEB STORAGE

WebStorage	8.0+	2.0+	4.0+	4.0+	10.5+

GEOLOCALIZZAZIONE

- Le Geolocation API in realtà non fanno parte della specifica HTML 5, anche se spesso sono associate a quest'ultima (persino articoli di stampa specialistica sbagliano). Gli strumenti di relativi alla geolocalizzazione sono oggetto di standardizzazione da parte del Geolocation Working Group.
- Questa tecnologia è stata sviluppata in concomitanza con HTML 5 ed è anch'essa una componente innovativa della programmazione web
- Molte applicazioni in rete hanno già cominciato a farne uso, soprattutto nel campo mobile.

GEOLOCALIZZAZIONE

- I principali metodi associati alla geoloacalizzazione sono getCurrentPosition e watchPosition, entrambi utili allo stesso scopo: ottenere la posizione corrente
- getCurrentPosition: rileva la posizione un'unica volta
- watchPosition, dopo la prima rilevazione, nel caso in cui la posizione abbia subito una variazione significativa, trasmette una notifica contenente le nuove informazioni. Inoltre watchPosition restituisce un identificativo numerico univoco che servirà a localizzare i successivi spostamenti del dispositivo
- Per terminare la lettura ripetuta delle coordinate è necessario invocare il metodo clearWatch che accetta come argomento tale identificativo. Entrambe le funzioni accettano tre argomenti: successCallback, errorCallback e options.

GEOLOCALIZZAZIONE: SUCCESSCALLBACK

- Funzione da eseguire nel caso in cui il rilevamento della posizione è andato a buon fine. Alla funzione è passato come argomento un riferimento all'oggetto Position. Quest'ultimo ha due proprietà: la data e l'ora del rilevamento della posizione contenute all'interno della proprietà timestamp e l'oggetto Coords che permette l'accesso alle seguenti proprietà:
    - **longitude**: informazioni di longitudine.
    - **latitude**: informazioni di latitudine.
    - **altitude**: informazioni relative all'altitudine. Se il dispositivo su cui è installato il browser non riesce a rilevare l'altitudine, il parametro sarà null.
    - **accuracy**: indica un valore in metri che rappresenta l'approssimazione dei valori di longitudine e latitudine.
    - **altitudeAccuracy**: indica un valore in metri che rappresenta l'approssimazione del valore di altitudine. Se il valore di altitude è null allora lo sarà anche quello di altitudeAccuracy.
    - **heading**: indica la direzione espressa in gradi. Il valore corrisponde a una rotazione oraria rispetto al nord. Se il dispositivo è stazionario, il valore sarà NaN.
    - **speed**: velocità espressa in metri al secondo. Se la velocità è zero heading sarà NaN.

GEOLOCALIZZAZIONE: ERRORCALLBACK

- Parametro facoltativo che rappresenta la funzione da eseguire nel caso in cui il rilevamento della posizione sia fallito. Alla funzione è passato come argomento un riferimento all'oggetto PositionError contenente le due proprietà message e code utili a comprendere la natura dell'erroree può assumere i seguenti valori interi:
  - **PERMISSION_DEINED**, l'utente non ha dato il consenso relativo alla rilevazione della posizione.
  - **POSITION_UNAVAILABLE**, Il tracciamento è fallito a causa di un errore relativo al fornitore dei dati.
  - **TIMEOUT**, il tempo prestabilito entro il quale il tracciamento avrebbe dovuto avere luogo è stato superato.

GEOLOCALIZZAZIONE: OPTIONS

- è un oggetto di tipo PositionOptions le cui tre proprietà enableHighAccuracy, timeout e maximumAge permettono di intervenire nel modo in cui avviene il tracciamento:
  - **enableHighAccuracy**: è una proprietà booleana che se impostata a true viene richiesto di determinare la posizione nel modo più accurato possibile. Le conseguenze però sono un maggior tempo di elaborazione e un aumento del consumo energetico.
  - **timeout**: è la soglia di tempo prefissata entro la quale si desidera una risposta. Se la soglia viene superata l'esecuzione fallisce. Il valore è espresso in millisecondi e nel caso in cui non venga impostata una soglia il valore è pari a zero.
  - **maximumAge**: è un intervallo di tempo espresso in millisecondi che, nel caso di acquisizioni della posizione ripetute, stabilisce quanto tempo deve trascorrere per l'aggiornamento dei valori.

REVERSE GEOCODING

- Con il termine Reverse geocoding s'intende una metodologia che permette, tramite le informazioni di latitudine e longitudine, di ricavare altre informazioni semplici da comprendere come il nome di una via o di una località

- Nel web un modo semplice per farlo potrebbe essere quello di sfruttare le potenzialità di Google Maps. Un esempio si trova alla fine di questa sezione.

GEOLOCALIZZAZIONE: PRIVACY

- Parlando di geolocalizzazione sicuramente il tema della privacy è un argomento da non sottovalutare
- Fortunatamente se un sito web ospita tale tecnologia, l'utilizzo è consentito solo col previo consenso dell'utente.
- Inoltre, dopo aver dato il consenso ad un determinato sito web, il browser deve permettere l'annullamento del consenso.

GEOLOCALIZZAZIONE

```
<!DOCTYPE html>
<html>
<body>
    <button onclick="getLocation()">Localizzami</button>
    <p id="demo"></p>
    <p id="demoUpdate"></p>
    <div id="mapholder"></div>
    <script>
        var x = document.getElementById("demo");
        var xUpdate = document.getElementById("demoUpdate");

        function getLocation() {
            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(showPosition, showError);
                navigator.geolocation.watchPosition(showPositionUpdate);
            } else {
                x.innerHTML = "Geolocation is not supported by this browser.";
            }
        }

        function showPosition(position) {
            x.innerHTML = "Latitude: " + position.coords.latitude +
                "<br>Longitude: " + position.coords.longitude;

            var latlon = position.coords.latitude + "," + position.coords.longitude;

            var img_url = "https://maps.googleapis.com/maps/api/staticmap?center=" +
                latlon + "&zoom=14&size=400x300&sensor=false&key=AIzaSyBu-916DdpKAjTmJNIgngS6HL_kDIKU0aU";
            document.getElementById("mapholder").innerHTML = "<img src='" + img_url + "'>";
        }
```

GEOLOCALIZZAZIONE

```
        function showPositionUpdate(position) {
            xUpdate.innerHTML = "Latitude: " + position.coords.latitude +
                "<br>Longitude: " + position.coords.longitude;
        }

        function showError(error) {
            switch (error.code) {
                case error.PERMISSION_DENIED:
                    x.innerHTML = "User denied the request for Geolocation."
                    break;
                case error.POSITION_UNAVAILABLE:
                    x.innerHTML = "Location information is unavailable."
                    break;
                case error.TIMEOUT:
                    x.innerHTML = "The request to get user location timed out."
                    break;
                case error.UNKNOWN_ERROR:
                    x.innerHTML = "An unknown error occurred."
                    break;
            }
        }
    </script>
</body>
</html>
```

Dott. Antonio Giovanni Lezzi

BACKEND

- Le crescenti capacità di dispositivi e sistemi operativi mobili, insieme alle connessioni in rete sempre più diffuse e veloci hanno offerto nuovi strumenti alla fantasia degli sviluppatori
- In questo contesto le nuove applicazioni tengono ad essere sempre meno "chiuse" nei dispositivi, e maggiormente capaci di interagire con servizi remoti creando reti "social" di utenti.
- Per consentire il funzionamento di tutto ciò, è necessaria una controparte server in grado di rendere disponibili, tramite API, servizi quali autenticazione, storage dei dati, push notification, comunicazione tra utenti e molto altro: i backend

BACKEND

- La creazione di un backend per conto proprio può portare gli sviluppatori ad incontrare delle difficoltà di vario genere
- carattere tecnico: i programmatori potrebbero non avere le competenze e l'esperienza necessarie in ambito server per realizzare un backend di qualità, sicuro ed efficiente
- carattere organizzativo, per esempio nel conciliare tempistiche e risorse umane coinvolte nello sviluppo delle componenti server e client
- carattere infrastrutturale, l'epoca del Cloud, delle risorse e delle funzionalità offerte "as a service", in cui senza avere a disposizione proprie macchine fisiche o virtuali si possono trovare servizi in rete che offrono elaborazione e storage in maniera affidabile, flessibile e con costi contenuti.

DATABASE

- Un database è un contenitore di informazioni, sono software specializzati nella memorizzazione dei dati in maniera persistente
- Consideriamo una libreria, dove ci sono tanti scaffali divisi con qualche particolare logica, per esempio per autore, e dove dentro ad ogni scaffale ci sono le informazioni salvate (nel caso della libreria, sarebbero i libri).
- Chi va in libreria ad affittare o comprare un libro non interessa come questo Database sia organizzato e composto. Vuole semplicemente prendere il libro in questione e andarsene
- Infatti, dentro ad una Libreria c'è sempre un commesso che penserà lui a recuperarlo dato che conosce l'organizzazione della libreria.
- Come esistono librerie che organizzano i libri in maniera differente, esistono anche Database che salvano in memoria i dati in maniera completamente differente

DATABASE

- Esistono due tipologie di database o almeno queste sono quelle più utilizzate:
- **Database relazionali**, organizzano le informazioni in una sorta di tabella divisa per righe e colonne, ogni colonna contiene una particolare informazione
- Per esempio una tabella di un database relazione potrebbe essere Utente. Dentro Utente le colonne dividono le informazioni in età, nome, cognome ecc. Nelle righe vengono messi i valori da memorizzare
- L'accesso a queste informazioni avviene mediante l'utilizzo dei linguaggi SQL e per tanto vengono chiamati database SQL.

DATABASE

- **Database non Relazionali:** Le informazioni vengono memorizzate non più in tabelle ma in strutture di tipo chiave-valore come

- Questa è la tecnologia che sta alla base dei Database Realtime di Firebase. Un esempio, poi lo vedremo nel dettaglio, potrebbe essere quello di un dizionario che organizza le informazioni in questo modo:

```
"utente" = {
    "nome" = "Antonio"
    "cognome" = "Lezzi"
    "età" = "35"
}
```

DATABASE

- la chiave dell'informazione è l'username dell'utente, "utente", associato a questa chiave c'è un dizionario di coppie chiavi valore
- Le chiavi vengono sempre messe a sinistra e i valori a destra. La struttura richiama quella della sintassi JSON
- Questi Database non relazioni, non utilizzano SQL come metodo di lettura ed interazione e, di conseguenza, vengono chiamati Database NoSQL (Not Only SQL dato che alcuni potrebbero anche utilizzare SQL).

FIREBASE

- In casi in cui si richiede un backend per la propria app mobile o web ma si vogliono evitare difficoltà come quelle appena citate, si può fare affidamento ad un servizio online, offerto in cloud e già pronto all'interazione con i nostri software
- Firebase è un servizio online BAAS (Backend As A Service) che permette di salvare e sincronizzare i dati elaborati da applicazioni web e mobile
- È un database NoSQL dalle grandissime risorse, ad alta disponibilità ed integrabile in tempi rapidissimi in altri progetti software, semplicemente sottoscrivendo un account al servizio
- www.firebase.com

FIREBASE

- Nel mese di ottobre 2014 James Tamplin, CEO e cofondatore, annunciava che Firebase entrava a fare parte dei servizi Google, dichiarava nel post che in soli tre anni la sua "pazza idea che avrebbe potuto funzionare" si era trasformata in una realtà su cui 110.000 sviluppatori basavano il proprio lavoro

- Il successo di Firebase è legato alle sue caratteristiche:

  - **capacità di sincronizzazione dei dati oltre che di storage**: Firebase è in grado di aggiornare i dati istantaneamente, sia se integrato in app web che mobile. Non appena l'app recupera la connettività sincronizza i dati mostrati con le ultime modifiche apportate

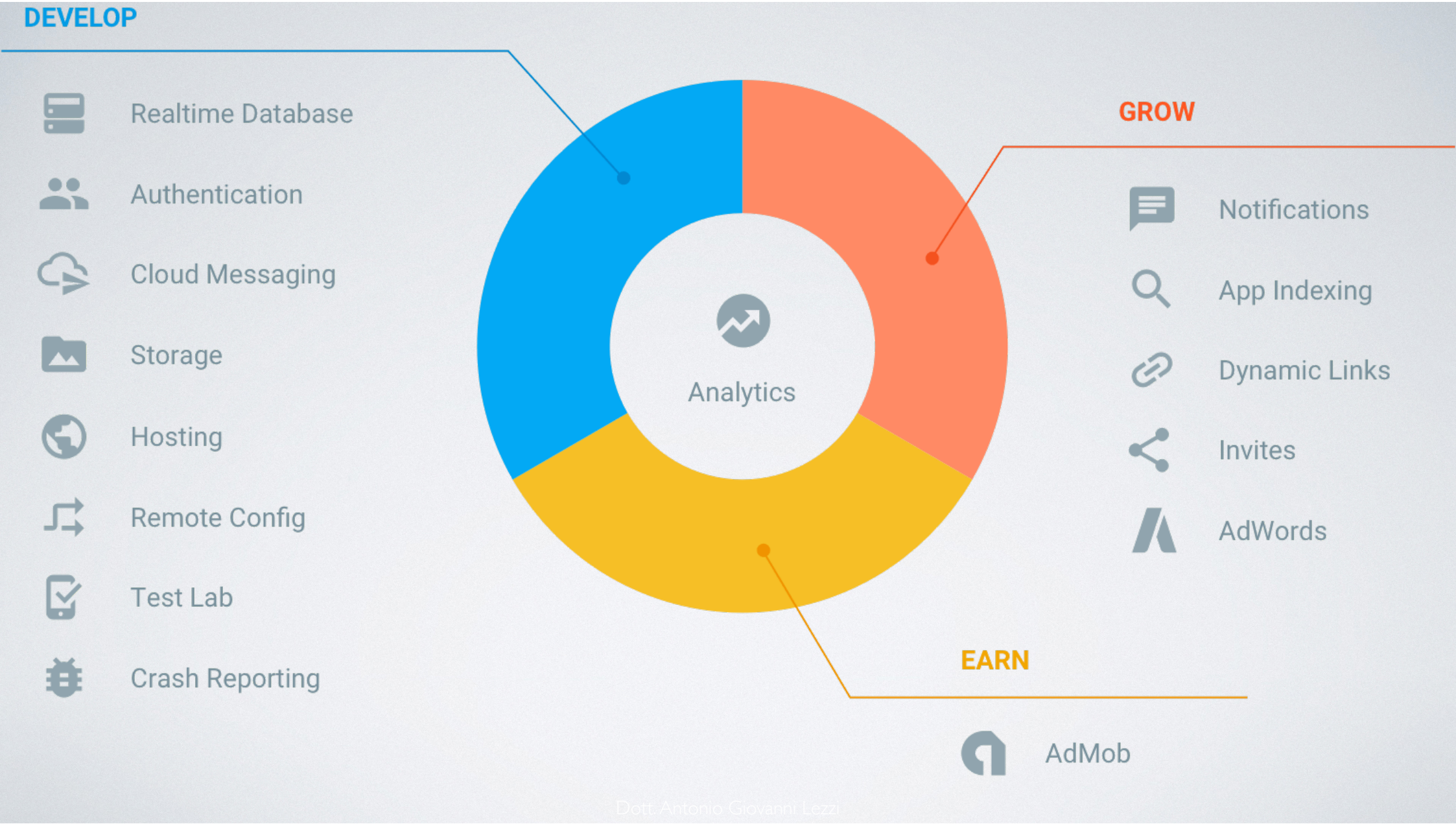
  - **disponibilità di librerie client per integrare Firebase in ogni app**. Android, Javascript e framework con esso realizzati (Node.js, Angular.js, Ember.js, Backbone.js e molti altri), Java e sistemi Apple

  - **Sicurezza**: i dati immagazzinati in Firebase sono replicati e sottoposti a backup continuamente. La comunicazione con i client avviene sempre in modalità crittografata tramite SSL con certificati a 2048-bit;

  - **costi differenziati in base all'uso e alle capacità richiesti**. Si parte dal "Hacker Plan" ed esistono altri piani per la produzione (Candle, Bonfire, Blaze, Inferno Plan) con prezzi mensili variabili da 49 $ fino a 1.499 $ e caratterizzati da capacità di memorizzazione crescenti

FIREBASE

- La caratteristica più di spicco di Firebase è la sua tecnologia Real Time, per Real Time intendono che i dati che scriverai nel DB saranno immediatamente disponibili a tutti gli utenti connessi a quello spazio
- Le caratteristiche principali per poi cominciare a vedere come poter utilizzare il database Realtime di Firebase con il linguaggio Swift:
  - **Realtime Database**: I dati saranno accessibili ovunque ed in maniera istantanea. Es. scrivo un messaggio, lo salvo su Firebase e quel messaggio viene subito letto da un'altra applicazione.
  - **Autenticazione**: Puoi creare un tuo sistema di autenticazione (per esempio con email e password) oppure puoi utilizzare quelli dei Social in maniera semplificata.
  - **Storage**: Puoi archiviare su Firebase qualsiasi tipologia di dati. Dalle immagini, ai PDF ai normali tipi di dato come le stringhe e i numeri.
  - **AdMob**: Con Firebase accedi anche al framework AdMob che ti permette di monetizzare la tua applicazione tramite l'advertising (i banner).
  - **Analytics**: Ti mette a disposizione la potenza di Google Analytics per le statistiche della tua applicazione.
  - **Altro**


DEVELOP
Realtime Database
Authentication
Cloud Messaging
Storage
Hosting
Remote Config
Test Lab
Crash Reporting
Analytics
GROW
Notifications
App Indexing
Dynamic Links
Invites
AdWords
EARN
AdMob
Dott. Antonio Giovanni Lezzi

FIREBASE

- Per creare l'account Hacker Plan è sufficiente sottoscrivere un account Google sul sito di Firebase https://www.firebase.com/signup/
- Consente di avere a disposizione un massimo di 50 connessioni, 5 GB di trasferimento dati e 100 MB di storage
- L'uso di questo piano gratuito è esclusivamente didattico e di sperimentazione ma utilissimo per imparare ad integrare Firebase nei propri progetti

FIREBASE

- All'interno, se non hai mai utilizzato Firebase, troverai un enorme bottone con scritto Create New Project
- Successivamente ti chiederà di assegnare un nome al progetto e lo stato della tua residenza o società
- Il nome può essere indipendente dal nome che poi darai effettivamente alla tua applicazione (consiglio di mettere nomi simili in modo da ricordarti che quel progetto Firebase è associato a quella determinata App dato che potrai avere N progetti su Firebase)

Firebase

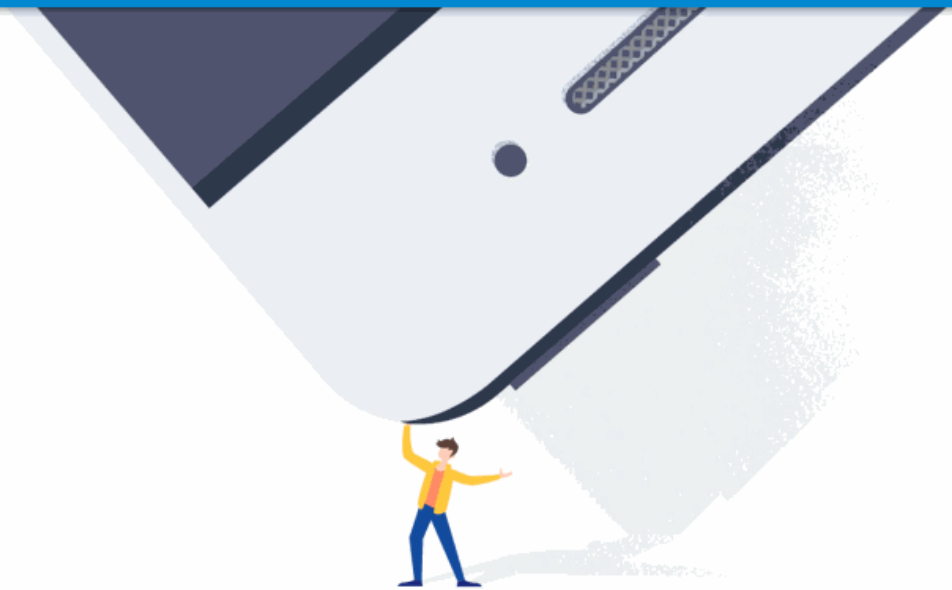
Docs

Welcome to Firebase

Tools from Google for developing great apps, engaging with your users, and earning more through mobile ads. Learn more

CREATE NEW PROJECT

or import a Google project



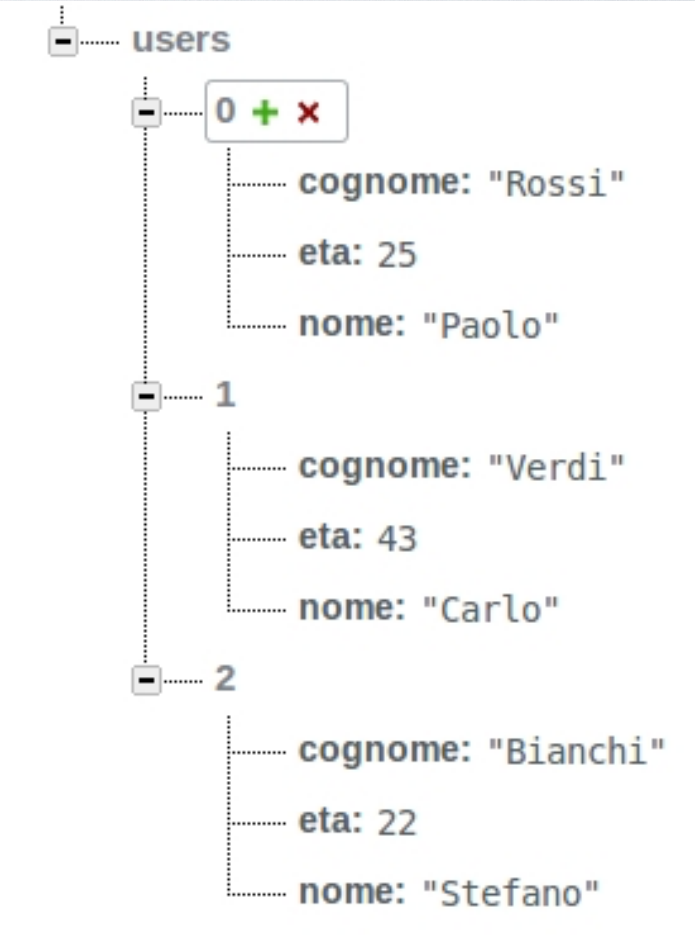
Projects using Firebase

FIREBASE: SETUP

- All'account creato sarà associato un indirizzo, nel formato <nomefornito>.firebaseio.com, indirizzo che verrà usato nello script in JS, ma mascherato dall'invocazione di firebase.database().ref()
- Tale riferimento potrà essere utilizzato all'interno del codice alla base di dati remoti e tramite metodi child(string)
- Non vi sono tabelle né record, nessuna traccia dell'approccio relazionale. Tutti i dati inseriti in Firebase vanno a costituire un albero JSON
- I dati al suo interno potranno essere strutturati secondo questo formato: si potranno inserire mappe, liste, stringhe, tipi numerici e booleani

FIREBASE: SETUP

- Per prendere confidenza con la struttura dati di Firebase, conviene usare il pannello di controllo, per il momento senza scrivere codice
- La figura seguente mostra informazioni che definiscono tre utenti, ognuno dei quali caratterizzato da nome, cognome ed età.



FIREBASE

- I dati aggiunti saranno usati come chiavi e potranno essere utilizzati per recuperare i valori eventualmente ad essi associati.
- È interessante notare che ogni nodo ha un suo indirizzo, che si renderà utile nella programmazione delle nostre app per accedere ai dati
- Ad esempio, con riferimento alla precedente figura, all'URL https://user-firebase-1000.firebaseio.com/users/1/cognome corrisponderà il valore Verdi. Osservando l'indirizzo da vicino si vede che users è la radice della struttura dati, il valore 1 indica la posizione del secondo nodo della lista di utenti (l'indice ha base 0), e cognome sarà la chiave con cui accedere al valore rintracciato.
- Una volta preso contatto con la struttura dati di Firebase si potrà iniziare a programmare
- Qualunque tecnologia si decida di utilizzare, sarà comunque necessario effettuare il link alla libreria corrispondente nel progetto.

FIREBASE: STRUTTURA DATI

- Ipotizziamo che un utente della tua applicazione voglia salvare delle note personali associate solamente al suo account
- La tentazione più forte, in questi casi, è quella di innestare all'interno di un dato utente un nuovo sotto nodo chiamato note. Cosa che crea non pochi problemi.
- Il problema principale di una struttura del genere è che, quando accedi ad un nodo, per esempio "utenti", si scarica tutta la struttura dei dati a lui associata. Quindi accedendo a "utenti" scaricherai sia l'utente in questione e tutti i suoi sotto nodi che tutti gli altri utenti con i loro rispettivi sotto nodi.
- Firebase Database Realtime permette di innestare i dati fino a 32 livelli di profondità, loro stessi consigliano nella guida ufficiale, è preferibile utilizzare una struttura dei dati Flat
- Si potrebbe creare un sotto nodo del nodo principale chiamato note dove suddividi le note degli utenti per username:
- Tra le tantissime proprietà che derivano dall'utilizzare questa struttura c'è indubbiamente sia quella della facilità di lettura (intesa anche come velocità dell'elaborazione) che quella di possedere una struttura di facile manutenzione e upgrade.

FIREBASE: STRUTTURA DATI

- All'interno del Database Realtime puoi salvare diversi tipi di dato tra cui:
  - Stringhe
  - Numeri (Int, Double, Float ecc)
  - Array
  - Dizionari
- Gli array ed i dizionari devono contenere elementi che hanno valori String e Numeri. Puoi creare anche array che contengono dizionari e viceversa.

FIREBASE IN JS

- In JavaScript si potrà importare la libreria tramite il tag HTML <script> fornito dalla creazione del DB:

```
<script src="https://www.gstatic.com/firebasejs/3.9.0/firebase.js"></script>
<script>
    // Initialize Firebase
    var config = {
        apiKey: "AIzaSyAR8-qS6dAu559AvboEwn0ApAz3uInAt5c",
        authDomain: "resplendent-inferno-6984.firebaseapp.com",
        databaseURL: "https://resplendent-inferno-6984.firebaseio.com",
        projectId: "resplendent-inferno-6984",
        storageBucket: "resplendent-inferno-6984.appspot.com",
        messagingSenderId: "4749346084"
    };
    firebase.initializeApp(config);

    var todoList = firebase.database().ref();
</script>
```

FIREBASE: INSERIMENTO DATI

- L'inserimento di nuovi dati può essere fatto usando un oggetto Firebase per collegarsi alla struttura dati remota, aggiungendo dei nodi figli con relativo valore

```
var fb = firebase.database().ref().child(key);
fb.set({ name: "valore" });
fb.push({ name: "valore" });
```

FIREBASE: LETTURA DATI

- Dopo aver inserito dati all'interno del nostro account Firebase, avremo bisogno di leggerli, ogni singolo dato ha un indirizzo che lo rende accessibile in una modalità stile REST.

- Quando si vorrà leggere un dato depositato nel proprio account si dovrà sempre predisporre un riferimento all'indirizzo da leggere e successivamente attivare un evento di callback ad esso collegato

- In JavaScript, il seguente codice recupera il valore 43 collegato al nodo https://user-firebase-1000.firebaseio.com/users/1/eta:

```
// creazione del riferimento al nodo Firebase
//var ref = firebase.database().ref().child('users/1/eta');
var ref = firebase.database().ref('users/1/eta');

// collegamento della funzione callback all'evento value
ref.on("value", function(snapshot) {
  console.log("Valore recuperato: " + snapshot.val());
}, function (errorObject) {
  console.log("Errore: " + errorObject.code);
});
```

- Al momento dell'effettivo recupero del valore verrà invocata la funzione di callback, che stamperà un messaggio nella console.

FIREBASE: LETTURA DATI

- Usando la terminologia Firebase, il valore recuperato prende il nome di DataSnapshot e rappresenta l'immagine del contenuto di un nodo in un dato istante
- La stringa passata come primo argomento al metodo on indica il tipo di evento di cui siamo in attesa
- L'evento value, usato nel caso precedente, serve a leggere uno snapshot dei dati.
- Altri eventi, alternativi a value, offrono contemporaneamente la lettura di uno snapshot e la sincronizzazione automatica a fronte di modifiche intervenute sui dati remoti

FIREBASE: EVENTI

- **child_added**: utilizzato per le liste di oggetti, viene invocato per ognuno degli elementi che le compongono e per ogni nuovo elemento che vi viene aggiunto;
- **child_changed**: viene attivato quando il nodo puntato è modificato. Nel metodo di callback sarà disponibile il nuovo dato aggiunto in remoto;
- **child_removed**: invocato alla rimozione del nodo.
- Tipicamente i tre eventi vengono configurati insieme in uno script, per poter fornire alla pagina un flusso continuo di aggiornamento dei dati ottenuti.
- I callback che abbiamo assegnato con il metodo on possono essere rimossi con il metodo off.

FIREBASE: EVENTI

- Assumendo che ref sia il riferimento ad un nodo Firebase, il seguente codice cancella tutti i metodi di callback assegnati all'evento value del riferimento ref:

```
ref.off("value")
```

- Altrimenti, in maniera più mirata, si può specificare solo il callback da eliminare su quel particolare evento:

```
ref.off("value", callbackFun)
```

- Infine, ref.off() rimuove tutti i callback di qualunque tipo.

- È possibile anche fare in modo che un callback venga cancellato subito dopo avere ottenuto lo snapshot dei dati. In questo caso si userà il metodo once:

```
ref.once("value", function(snapshot) {
     //operazioni da eseguire
});
```

FIREBASE

- Il recupero dei dati può essere arricchito con funzionalità "da query", principalmente, ordinamento e limitazione del numero di risultati.
- I metodi che svolgono questi compiti sono:
- **ordinamento**: orderByChild(), orderByKey() e orderByPriority(). Il primo ordinerà i nodi in base ad una child key, ossia una chiave contenuta in essi; il secondo ordinerà i dati in base alle chiavi; l'ultimo li ordinerà in base alla priorità del nodo;
- **selezione del numero**: limitToFirst(), limitToLast(), startAt(), endAt() e equalTo(). Lo scopo di questi metodi essenzialmente è prelevare una porzione dei dati ottenuti. Ad esempio, limitToFirst e limitToLast li preleveranno rispettivamente all'inizio e alla fine della lista, mentre startAt ed endAt permetteranno di ottenere una porzione centrale.

FIREBASE

- Ad esempio, si potrebbe eseguire una query per recuperare i due oggetti con il prezzo minore:

```
ref.orderByChild("price").limitToFirst(2).on("child_added", function(snapshot)
{
    // operazioni da svolgere
});
```

- I dati ottenuti apparterranno alla classe DataSnapshot.

- I metodi di callback verranno collegati mediante listener. L'evento value verrà trattato usando il metodo addValueEventListener della classe Firebase. Il parametro che riceverà in input sarà di tipo ValueEventListener ed al suo interno effettueremo l'override dei metodi onDataChange ed onCancelled.

- Analogamente, si potrà definire un ChildEventListener che con i suoi metodi onChildAdded, onChildChanged e onChildRemoved risponderà ai corrispondenti eventi.

FIREBASE SNIPPET

```
//creazione di un utente
var email = "myemail@email.com";
var password = "mypassword";
firebase.auth().createUserWithEmailAndPassword(email, password).catch(function(error) {
   console.log(error.code);
   console.log(error.message);
});

//login di un utente
var email = "myemail@email.com";
var password = "mypassword";
firebase.auth().signInWithEmailAndPassword(email, password).catch(function(error) {
   console.log(error.code);
   console.log(error.message);
});

//logout
firebase.auth().signOut().then(function() {
   console.log("Logged out!")
}, function(error) {
   console.log(error.code);
   console.log(error.message);
});
```

FIREBASE ESEMPIO

```
<!DOCTYPE html>
<html>

<head>
    <title>Elenco</title>
    <script src="https://www.gstatic.com/firebasejs/3.9.0/firebase.js"></script>
    <script>
        // Initialize Firebase
        var config = {
            apiKey: "AIzeSyAT8-qS7dAu555AvboEwn1TpAz3uInAt4c",
            authDomain: "resplendent-inferno-6981.firebaseapp.com",
            databaseURL: "https://resplendent-inferno-6981.firebaseio.com",
            projectId: "resplendent-inferno-6981",
            storageBucket: "resplendent-inferno-6981.appspot.com",
            messagingSenderId: "474934608216"
        };
        firebase.initializeApp(config);

        var todoList = firebase.database().ref();
```

FIREBASE ESEMPIO

```
function saveToList(event) {
    if (event.which == 13 || event.keyCode == 13) { // appena si preme invio
        var todoName = document.getElementById('todoName').value.trim();
        if (todoName.length > 0) {
            saveToFB(todoName);
        }
        document.getElementById('todoName').value = '';
        return false;
    }
};

function saveToFB(todoName) {
    todoList.push({
        name: todoName
    });
};

function refreshUI(list) {
    var lis = '';
    for (var i = 0; i < list.length; i++) {
        lis += '<li data-key="' + list[i].key + '">' + list[i].name + ' [' + genLinks(list[i].key, list[i].name) + ']</
li>';
    };
    document.getElementById('todoList').innerHTML = lis;
};
```

FIREBASE ESEMPIO

```
function genLinks(key, mvName) {
    var links = '';
    links += '<a href="javascript:edit(\'' + key + '\',\'' + mvName + '\')">Edit</a> | ';
    links += '<a href="javascript:del(\'' + key + '\',\'' + mvName + '\')">Delete</a>';
    return links;
};

function edit(key, mvName) {
    var todoName = prompt("Aggiorna le cose da fare", mvName);
    if (todoName && todoName.length > 0) {
        // si costruisce endpoint a Firebase alla collezione dei todo
        var updatetodoRef = buildEndPoint(key);
        updatetodoRef.update({
            name: todoName
        });
    }
}

function del(key, mvName) {
    var response = confirm("Sicuro di voler rimuovere \"" + mvName + "\" dalla lista?");
    if (response == true) {
        // si costruisce endpoint a Firebase alla collezione dei todo
        var deletetodoRef = buildEndPoint(key);
        deletetodoRef.remove();
    }
}
```

FIREBASE ESEMPIO

```
        function buildEndPoint(key) {
            return firebase.database().ref().child(key);
        }
        // viene invocato appena il caricamento iniziale è stato effettuato o quando i dati sono cambiati
        todoList.on("value", function(snapshot) {
            var data = snapshot.val();
            var list = [];
            for (var key in data) {
                if (data.hasOwnProperty(key)) {
                    name = data[key].name ? data[key].name : '';
                    if (name.trim().length > 0) {
                        list.push({
                            name: name,
                            key: key
                        })
                    }
                }
            }

            refreshUI(list);
        });
    </script>
</head>
```

FIREBASE ESEMPIO

```
<body>
    <h1>Elenco</h1>
    <hr />
    <input type="text" id="todoName" placeholder="Inserisci
qui le info" onkeypress="return saveToList(event)" />
    <h4>Cose da fare</h4>
    <ol id="todoList">
        Caricamento dati...
    </ol>
</body>
</html>
```

FIREBASE STORAGE

```
<img id="imageFirebase"><br>
<progress value="0" max="100" id="uploader"></progress>
<input type="file" value="upload" id="fileButton"><br>
```

FIREBASE FILE DOWNLOAD

```
var file = “preview.jpg"; //nome preso da DB
var imgDownloadRef = firebase.storage().ref("/images/" +
file);

imgDownloadRef.getDownloadURL().then(function(url) {
    document.querySelector('img').src = url;
}).catch(function(error) {
    console.error(error);
});
```

FIREBASE FILE UPLOAD

```
var uploader = document.getElementById('uploader');
var fileButton = document.getElementById('fileButton');

fileButton.addEventListener('change', function(e) {
    var file = e.target.files[0];

    var storageRef = firebase.storage().ref("/images/" + file.name);

    var task = storageRef.put(file);

    task.on('state_changed',
        function progress(snapshot) {
            var percentage = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
            uploader.value = percentage;
            if (percentage == 100) {
                alert("file uploaded Successfully");
            }
        },
        function error(err) {   },
        function complete() {   }
    );
});
```

CONCLUSIONI

- Esiste una documentazione molto semplice, con concetti espressi chiaramente ed esempi molto utili
- Oltre a quanto visto si potranno realizzare query più complesse con filtraggio dei valori ed iterazioni
- Va affrontato in dettaglio l'argomento della sicurezza delle informazioni, a partire dalla validazione dei dati e autenticazione degli accessi.
- Sono tante le funzionalità che Firebase offre e sicuramente aumenteranno ancora quando la neonata alleanza con Google inizierà a dare frutti.

Dott. Antonio Giovanni Lezzi

BOOTSTRAP

- Il framework più utilizzato, il più copiato (20.000 fork su github), nato da Twitter ed oggi è indipendente
- Bootstrap può considerarsi il principale framework per lo sviluppo di interfacce Web e lo standard de facto come punto di partenza in ambiti come la creazione di template HTML preconfezionati e temi per i principali CMS, soprattutto in un'ottica responsiva.

BOOTSTRAP

- È un aggregato di diversi progetti:
- Grid System: 960 Grid System è uno sforzo per snellire lo sviluppo del flusso di lavoro su web, fornendo le dimensioni comunemente utilizzate, sulla base di una larghezza di 960 pixel. Ci sono due varianti: 12 e 16 colonne, che possono essere utilizzate
- Less: E' una tecnica di refactoring css, sostituita al foglio di stile permette di assegnare costanti su proprietà che all'interno di un normale css si ripeterebbero all'infinito
- Bootstrap Javascript: contiene la logica e i componenti grafici che permettono la realizzazione di un lavoro perfetto e adattabile a seconda delle nostre esigenze.

BOOTSTRAP: PERCHÉ?

- Velocità di sviluppo: Bootstrap porta già a metà dell'opera e risparmia la parte di sviluppo più noiosa
- Layout responsive: Attribuendo una semplice classe potrete gestire il lato responsive del vostro layout rendendolo automaticamente compatibile per mobile e tablet.
- Componenti integrati: I componenti javascript più utilizzati sono inclusi all'interno del framework (carousel slider, modali, popup, e quant'altro), questo riduce a zero la possibilità di incorrere in conflitti fra librerie aumentando l'affidabilità del progetto web.

BOOTSTRAP: COME?

- Iniziare con bootstrap è semplice e ci sono 2 soluzioni:
  - Scarica il pacchetto formato zip dal sito ufficiale;
  - Personalizza il tuo bootstrap, scegliendo e/o scartando le componenti e personalizzando il css attraverso un widget: http://twitter.github.com/bootstrap/download.html

BOOTSTRAP: COME?

- contiene in tre cartelle IMG, JS e CSS tutto l'occorrente per partire ed è possibile partire scrivendo il seguente file:

```
<!DOCTYPE html>
<html>
    <head>
    <title>Bootstrap Template</title>
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
</head>
<body>
    <h1>Hello, world!</h1>
    <script src="js/bootstrap.min.js"></script>
</body>
</html>
```

BOOTSTRAP: COME?

- Vediamo rapidamente quali sono i principali componenti che possiamo usare.
- Grid System
- CSS Base
- Componenti e Javascript

MEDIA QUERIES

- Nei file css le regole attribuite ai media queries, cioè @media si identifica il supporto (schermo, stampa, etc.) cui applicare un particolare foglio di stile
- È un attributo opzionale

Valore	**Descrizione**
all	il CSS si applica a tutti i dispositivi di visualizzazione
screen	schermo di computer
print	pagina stampata
projection	presentazioni e proiezioni
speech	dispositivi a sintesi vocale
braille	supporti basati sull'uso del braille
embossed	stampanti braille
handheld	dispositivi mobili con schermo piccolo e in genere dotati di browser con limitate capacità grafiche
tty	dispositivi a carattere fisso come i terminali
tv	visualizzazione su schermi televisivi

MEDIA QUERIES

- **Width**: designa la larghezza dell'area di visualizzazione del documento. Su un normale browser web essa è rappresentata dal viewport e non ha nulla a che vedere con le dimensioni dello schermo del dispositivo.
- **Height**: designa l'altezza dell'area di visualizzazione del documento. Come nel caso di width può essere definita con i prefissi min- (min-height) e max- (max-height).
- **Device-width**: si descrive la larghezza dell'intera area di rendering di un dispositivo. Su un computer o su un dispositivo mobile si intende la larghezza dell'intero schermo del dispositivo, non semplicemente quindi dell'area di visualizzazione del documento!
- Accetta i prefissi min- (min-device-width) e max- (max-device-width).
- **Device-height**: descrive l'altezza dell'intera area di rendering di un dispositivo. Accetta i prefissi min- e max-.

MEDIA QUERIES

- **Orientation**: indica se il dispositivo di output è in modalità landscape (la larghezza è maggiore dell'altezza) oppure portrait (l'altezza è maggiore della larghezza). I valori che può assumere sono appunto landscape o portrait.
- **Aspect-ratio**: Definisce il rapporto tra la larghezza (width) e l'altezza (height) dell'area di visualizzazione di un documento. Può assumere i prefissi min- e max-. I valori si esprimono attraverso due numeri interi separati dal simbolo /: 3/4, 16/9, etc.
- **Device-spect-ratio**:Descrive il rapporto tra la larghezza (device-width) e l'altezza (device-height) dell'area di rendering di un dispositivo. Può assumere i prefissi min- e max-.
- **Resolution**: Descrive la risoluzione (ovvero la densità di pixel) del dispositivo di output. I valori della risoluzione possono essere espressi in dpi (punti per pollice) oppure in dpcm (punti per centimetro). Può assumere i prefissi min- e max-

MEDIA QUERIES

```
@media screen and (min-width: 480px) {
    #leftsidebar {width: 200px; float: left;}
    #main {margin-left:216px;}
}
```

BOOTSTRAP: MOBILE FIRST

- Se in passato le funzionalità responsive erano opzionali, ora si è abbracciato l'approccio mobile first
- Una pagina basata su Bootstrap è di default adattabile ad una gamma ampia di device, dagli smartphone a crescere fino ai desktop ad altissima risoluzione.
- Il componente che ha subito le modifiche più radicali è la griglia, infatti in Bootstrap 3.0 il layout è di default fluido, con larghezze espresse in percentuale, e pertanto maggiormente adattabile a differenti risoluzioni di schermo
- Una serie di classi speciali abbinate a specifici breakpoint e di funzionalità avanzate rendono possibile la realizzazione di layout complessi e robusti in ogni scenario dettato dal dispositivo in uso

BOOTSTRAP: MOBILE FIRST

- Per assicurare un rendering appropriato e lo zoom al tocco, aggiungi il meta tag viewport nell'intestazione, <head>.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

- Si può disabilitare la funzionalità di zoom sui dispositivi Mobile aggiungendo user-scalable=no al meta tag viewport e gli utenti saranno soltanto capaci di scorrere su e giù la pagina

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
```

LESS

- Less è un pre-processore CSS, significa che estende il linguaggio CSS, aggiungendo funzionalità che permette di usare variabili, funzioni e molte altre tecniche che consentono di rendere CSS più sostenibile, creare dei temi e estendibile.
- Less ha a disposizione molti strumenti di terze parti che consentono di compilare i file e di osservare le modifiche
- Il luogo più veloce per primi esperimenti con Less è http://less2css.org
- NB: per usarlo si deve installare NodeJS o si deve integrare una libreria less.js che rallenta il render della pagina nel Browser

LESS

```
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  -webkit-box-shadow: @style @c;
  box-shadow:         @style @c;
}
.box-shadow(@style, @alpha: 50%) when
(isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}
.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%) }
}
```

```
.box {
  color: #fe33ac;
  border-color: #fdcdea;
}
.box div {
  -webkit-box-shadow: 0 0 5px rgba(0, 0, 0,
0.3);
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}
```

GRID SYSTEM

- Consente di creare dei design proporzionati, efficaci e armoniosi, sono i cosiddetti Grid Systems
- In parole povere, si suddivide la pagina in un certo numero di colonne immaginarie di dimensione fissa e separate da un margine prestabilito, in questo modo quando dobbiamo decidere la dimensione di un certo elemento, possiamo ragionare non più in termini assoluti (pixel, punti o altro) ma in termini appunto di "colonne"
- Questo ci consente di avere un naturale allineamento delle diverse sezioni e di divedere lo spazio senza dover calcolare ogni volta le dimensioni da assegnare
- Bootstrap utilizza un Grid System a 12 colonne da 60 px ognuna e distanziate di 20 px l'una, inoltre ha anche delle classi per gestire i Layout Fluidi, ovvero con dimensioni in percentuale anziché fisse.

GRID SYSTEM

- I sistemi a Griglia (grid systems) sono usati per creare layout di pagina attraverso righe e colonne che contengono i tuoi contenuti
  - Le righe devono essere piazzate all'interno del .container (fixed-width) o del .container-fluid (full-width) per un allineamento e un padding corretto.
  - Usa le righe per creare un gruppo orizzontale di colonne.
  - I contenuti dovrebbero essere inseriti all'interno delle colonne, e solo le colonne possono essere figli immediati delle righe nel codice html.
  - Sono disponibili classi predefinite come .row e .col-xs-4 per la creazione rapida dei layout a griglia
  - Le Colonne creano degli spazi attraverso padding. Questo padding si compensa nelle righe per la prima e l'ultima colonna con un margine negativo sulle .rows.
  - Le colonne della griglia si creano specificando un numero delle 12 colonne disponibili che si vogliono occupare. Per esempio, tre colonne uguali useranno tre .col-xs-4

GRID SYSTEM

- Per creare i breakpoint nel grid system (sistema a griglia) usiamo queste media queries nei file Less.

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */

/* Small devices (tablets, 768px and up) */
@media (min-width: @screen-sm-min) { ... }

/* Medium devices (desktops, 992px and up) */
@media (min-width: @screen-md-min) { ... }

/* Large devices (large desktops, 1200px and up) */
@media (min-width: @screen-lg-min) { ... }
```

- Occasionalmente allarghiamo queste media queries includendo un `max-width` per limitare il CSS a un insieme ristretto di device.

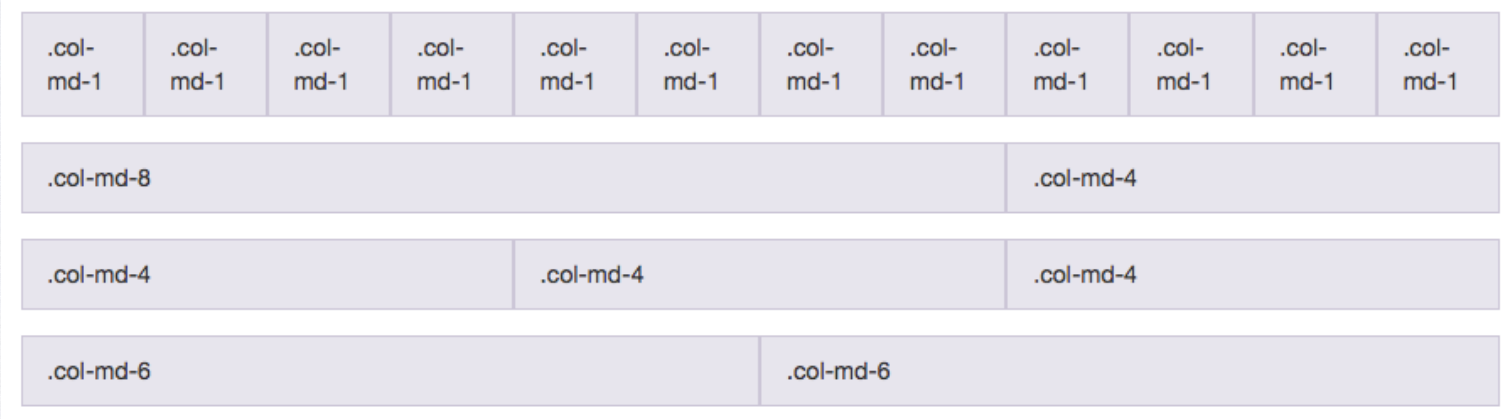
```
@media (max-width: @screen-xs-max) { ... }
@media (min-width: @screen-sm-min) and (max-width: @screen-sm-max) { ... }
@media (min-width: @screen-md-min) and (max-width: @screen-md-max) { ... }
@media (min-width: @screen-lg-min) { ... }
```

GRID SYSTEM

	Device extra piccoli Smartphone (<768px)	Piccoli device Tablet (≥768px)	Device Medi PC (≥992px)	Grandi device PC (≥1200px)
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Class prefix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
# of columns	12			
Column width	Auto	60px	78px	95px
Gutter width	30px (15px on each side of a column)			
Nestable	Yes			
Offsets	Yes			
Column ordering	Yes			

GRID SYSTEM

```
<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```



CONTENITORE FLUIDO

- Trasforma ogni layout (a griglia) fixed-width in un layout full-width cambiando il .container più esterno in un .container-fluid.

```
<div class="container-fluid">
  <div class="row">
    ...
  </div>
</div>
```

GRID SYSTEM

- Non vuoi che le tue colonne si mettano in pila nei device più piccoli? Per le colonne si usano le classi della griglia extra small e medium device .col-xs-* .col-md-*

```
<!-- Sugli smartphone impila le colonne creando un unico full-width e gli altri half-width -->
<div class="row">
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Le colonne hanno una larghezza del 50% su smartphone e sobbalzano fino ad una larghezza di 33.3% su
desktop -->
<div class="row">
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Le colonne hanno una larghezza sempre del 50%, sia su mobile che su desktop -->
<div class="row">
  <div class="col-xs-6">.col-xs-6</div>
  <div class="col-xs-6">.col-xs-6</div>
</div>
```

CSS BASE

- Bootstrap definisce un gran numero di classi CSS, per dare un aspetto gradevole a molti elementi usati molto spesso se non sempre
- Ad esempio, aggiungendo semplicemente class="btn" ad un' anchor link o ad un input button, possiamo renderlo molto più piacevole dal punto di vista grafico
- Anche per le tabelle, i form, i titoli, le immagini vale la stessa cosa
- Il modo migliore per capire quello che si può ottenere e dare un'occhiata agli esempi presenti sul sito ufficiale
- Non è da sottovalutare inoltre, che usando bootstrap saremo abbastanza sicuri che non ci saranno problemi di compatibilità multi browser, perché il framework è abbondantemente testato e ottimizzato.

CSS BASE

```
<p class="text-left">Left aligned text.</p>
<p class="text-center">Center aligned text.</p>
<p class="text-right">Right aligned text.</p>
<p class="text-justify">Justified text.</p>
```

COMPONENTI E JAVASCRIPT

- Oltre che dal punto di vista grafico, Bootstrap ci aiuta molto anche nell'implementazione di controllo ed elementi dinamici delle pagine
- Ad esempio, di menu dropdown, interfacce a tab, tooltip, alert, menu ad accordion, slider, banner di navigazione e molto altro
- Spesso non è necessario neanche scrivere una sola riga di codice Javascript per ottenere i risultati desiderati, infatti basta usare i data-attributes, lo script di base di bootstrap interpreta e gestisce senza nessun intervento da parte nostra
- Ad esempio se vogliamo creare un tooltip su un link, basterà aggiungere gli attributi rel="tooltip" title="first tooltip" nel TAG a.
- Tra le altre cose, possiamo anche gestire in maniera semplice dei design Responsive, ovvero adattabili dinamicamente a diverse risoluzioni e dispositivi (mobile, tablet, desktop ecc…).

BOOTSTRAP: STRUTTURA

- Usare un framework come Bootstrap comporta essenzialmente la definizione di specifici componenti nel codice HTML attraverso la corretta strutturazione del markup e l'associazione agli elementi HTML di classi e stili CSS
- È opportuno crearsi una sorta di scheletro, una struttura HTML di partenza per tutte la pagine del sito
- Potrà chiaramente variare in base alle richieste di questo o quel progetto, ma una base minima e condivisa si può trovare.

BOOTSTRAP: STRUTTURA

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bootstrap Starter Template</title>
  <!-- Fogli di stile -->
  <link href="css/bootstrap.css" rel="stylesheet" media="screen">
  <link href="css/stili-custom.css" rel="stylesheet" media="screen">
  <!-- Modernizr -->
  <script src="js/modernizr.custom.js"></script>
 </head>
 <body>
  <h1>Hello, world!</h1>
 <!-- jQuery e plugin JavaScript  -->
 <script src="http://code.jquery.com/jquery.js"></script>
 <script src="js/bootstrap.min.js"></script>
 </body>
</html>
```

LA FORMA PERFETTA

- Il DOCTYPE deve essere quello HTML5
- Nella sezione <head>, l'istruzione <meta http-equiv="X-UA-Compatible" content="IE=edge"> forza il browser di Microsoft a usare la modalità di rendering più avanzata, dal momento che il framework non supporta la modalità di compatibilità.
- L'uso del meta tag viewport è fondamentale. Lavoriamo con un framework responsivo per default.
- Passando ai fogli di stile, è opportuno sin dall'inizio collegare, oltre al CSS principale di Bootstrap, un CSS che ospiterà gli stili aggiuntivi, oltre che le regole con cui andremo a sovrascrivere e/o modificare gli stili predefiniti del framework.
- Al posto dell'HTML5 Shiv per Internet Explorer suggerito dagli sviluppatori è stato messo Modernizr, è un eccellente ausilio in svariati contesti.
- L'inserimento di jQuery e il collegamento al file con i plugin Javascript.

BOOTSTRAP: ESEMPI

EXAMPLE

Home Profile Messages

```
<ul class="nav nav-tabs">
  <li class="active">&lt;a href="#"&gt;Home&lt;/a&gt;</li>
  <li><a href="#">Profile</a></li>
  <li><a href="#">Messages</a></li>
</ul>
```

EXAMPLE

140x140 140x140 140x140

```
<img src="..." alt="..." class="img-rounded">
<img src="..." alt="..." class="img-circle">
<img src="..." alt="..." class="img-thumbnail">
```

EXAMPLE

Default Primary Success Info Warning Danger Link

```
<!-- Standard button -->
<button type="button" class="btn btn-default">Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of buttons -->
<button type="button" class="btn btn-primary">Primary</button>

<!-- Indicates a successful or positive action -->
<button type="button" class="btn btn-success">Success</button>

<!-- Contextual button for informational alert messages -->
<button type="button" class="btn btn-info">Info</button>

<!-- Indicates caution should be taken with this action -->
<button type="button" class="btn btn-warning">Warning</button>

<!-- Indicates a dangerous or potentially negative action -->
<button type="button" class="btn btn-danger">Danger</button>

<!-- Deemphasize a button by making it look like a link while maintaining button behavior -->
<button type="button" class="btn btn-link">Link</button>
```

RIFERIMENTI DI ESEMPI

- https://hackerstribe.com/guide/IT-bootstrap-3.1.1/css/
- https://hackerstribe.com/guide/IT-bootstrap-3.1.1/components/
- https://hackerstribe.com/guide/IT-bootstrap-3.1.1/javascript/

Dott. Antonio Giovanni Lezzi

INTRODUZIONE

- JavaScript è uno dei linguaggi di programmazione più usati al mondo
- L'enorme diffusione è dovuta al fatto che numerose librerie sono nate allo scopo di semplificare la programmazione sul browser
- La nascita di framework lato server e nel mondo mobile che lo supportano come linguaggio principale hanno aumentato la sua popolarità
- La diffusione di JavaScript non è necessariamente indice della sua effettiva conoscenza (chi copia codice adattandolo per necessità, esempio i grafici)

INTRODUZIONE

- JavaScript è un linguaggio di scripting lato client utilizzato per rendere dinamico il codice HTML.
  - Il documento HTML è generato staticamente.
  - Il codice JavaScript è immerso nel documento HTML ma viene eseguito dinamicamente solo al momento della richiesta del web client (browser).
- Principali usi:
  – posizionamento dinamico degli oggetti; – validazione campi dei moduli;
  – effetti grafici;
  –...

INTRODUZIONE

- JavaScript è un linguaggio di programmazione di tipo script:
  - Non è compilato
  - E' ad alto livello
- La sua caratteristica principale è di essere nato espressamente per il WWW (è stato introdotto da Netscape nel 1995) e di essere supportato dalla maggior parte dei browser in uso
- Non può essere adoperato per costruire programmi complessi o con particolari requisiti di prestazioni, tuttavia può essere impiegato per implementare velocemente piccole procedure e controlli.
- Il suo campo d'impiego naturale è la gestione sul web client (browser) di alcuni elementi delle applicazioni per WWW

STORIA

- JavaScript è stato ideato nel 1995 da Netscape e rilasciato con la versione 2.0 del suo browser, Netscape Navigator, dapprima con il nome LiveScript, successivamente JavaScript, creando confusione con Java
- JavaScript aggiunse alle pagine HTML la possibilità di essere modificate in modo dinamico, in base all'interazione dell'utente con il browser e senza coinvolgere il server
- Questa caratteristica è stata sottolineata durante gli anni '90 con il nome che veniva dato all'accoppiata HTML-JavaScript: Dynamic HTML (DHTML).
- Nel 1997, sulla base del linguaggio di Netscape, nacque lo standard ECMA-262 definito dall'organismo di standardizzazione industriale ECMA International che rappresentava le specifiche del linguaggio ECMAScript (punto di riferimento anche per altri linguaggi come ActionScript, WMLScript e QtScript)

STORIA

- L'esigenza di rendere il Web sempre più interattivo e dinamico favorì la nascita di tecnologie concorrenti: Flash, ActiveX e gli Applet Java
- Queste tecnologie fornivano funzionalità di maggior impatto rispetto a JavaScript, ma richiedevano specifici runtime o giravano solo su uno specifico browser.

STORIA

- La concorrenza è durata diversi anni infatti Flash, sul fronte dell'interazione utente e dei formati per l'advertising, sembrava essere destinato ad un lento declino.
- Flash, consente di creare semplicemente contenuti e interfacce, ma anche grazie all'assenza di una implementazione univoca degli standard HTML e JavaScript e alla Guerra dei Browser tra Microsoft e Netscape terminata nel 2000 con gli standard W3C
- Grazie ad Ajax, la possibilità cioè di comunicare in maniera asincrona con il server tramite script, a riportare JavaScript sulla cresta dell'onda facendo nascere il cosiddetto Web 2.0
- Con HTML5 ha amplificato le possibilità applicative del linguaggio, anche al di fuori del semplice Web browser. JavaScript può essere utilizzato anche lato server, in applicazioni desktop e mobile

COSA SERVE

- Per creare una pagina HTML non occorrono particolari strumenti chissà quanto sofisticati. Basta un normale editor di testo, va benissimo il Notepad di Windows o il Vi o Xedit di Unix/Linux.
- Naturalmente esistono decine di editor specializzati e molti di essi sono anche WYSIWYG (What You See Is What You Get), cioè permettono di creare la pagina disponendo tutti gli elementi proprio come essi appariranno una volta avviato il browser
- Browser. Naturalmente occorre un programma che sia in grado di leggere il file HTML e di visualizzarlo sullo schermo come una pagina; questo per poter vedere il lavoro che abbiamo fatto
- Anche di browser ne esistono decine; i più conosciuti sono Internet Explorer/Edge e FireFox, Chrome
- Questi browser non sono uguali; sono stati messi sul mercato in tempi diversi e i progettisti hanno avuto indicazioni diverse su come implementare alcune funzionalità. Può capitare che una funzionalità sia implementata su un browser ma non sull'altro oppure che per eseguire la stessa funzione sui due browser occorrano due codici distinti; qualora dovesse capitare

PAGINE WEB

- Con HTML quindi indichiamo, attraverso i tag, quali elementi dovranno apparire su uno schermo e come essi debbano essere disposti.
- Una pagina HTML è rappresentata da un file di testo, ovvero un file che possiamo modificare con programmi come notepad e in genere hanno un nome che finisce con l'estensione .html.
- Ciò significa che HTML non possiede i costrutti propri della programmazione, come i meccanismi "condizionali", che consentono di reagire in modo diverso a seconda del verificarsi di una condizione ("in questa situazione fai questo, in quest'altra fai quest'altro"), o i costrutti iterativi ("ripeti questa azione, finché non succede questo").
- Nota: Si parla in questo caso di linguaggio dichiarativo, che serve appunto ad indicare cosa deve apparire sullo schermo (testi, immagini, suoni, etc.), dove e in che sequenza. Nel caso di linguaggi in cui specifichiamo algoritmi precisi con "strutture di controllo", come C, C++, Java, o anche PHP e JavaScript, parliamo di "linguaggi imperativi".

PAGINE WEB

- Per lavorare con HTML è utile conoscere tutto il cosiddetto "stack", ovvero tutte le tecnologie necessarie a realizzare un sito o una applicazione.
- In passato si utilizzavano alcuni tag HTML per definire font (tipi di carattere), i colori o le dimensioni degli oggetti sullo schermo. Oggi il quadro è definitivamente cambiato e molte di quelle funzionalità sono deprecate favorendo una divisione dei compiti più chiara tra diversi strumenti:
- **HTML:** serve a definire quali sono gli elementi in gioco, stabilire collegamenti (link) tra le pagine e l'importanza (non la forma o il colore) che hanno i testi, creare form per gli utenti, fissare titoli, caricare immagini, video, etc.
- **CSS:** "fogli di stile". Si tratta di una serie di regole che permettono di definire l'aspetto (lo stile) che devono assumere gli elementi sulla pagina. Dimensioni, colori, animazioni, ogni caratteristica visuale può essere manipolata.
- **JavaScript:** è un linguaggio di programmazione che consente di manipolare veramente qualunque cosa nella pagina HTML: lo stile, i contenuti della pagina, ma soprattutto l'interazione con l'utente. Ci permette di creare la logica dell'interfaccia utente (o anche di una applicazione) e di sfruttare le API messe a disposizione dal browser: dalla gestione del mouse al touch, dalla manipolazione delle immagini, alle richieste di dati dinamiche (in modalità Ajax) alla gestione di dati in locale

STRUMENTI DI SVILUPPO

- Gli strumenti principali che servono a realizzare pagine HTML sono editor di testo, browser, software per manipolare foto o immagini vettoriali, tool per la progettazione e la messa online del sito o il deploy dell'app.

- Un buon ambiente di sviluppo può fare la differenza sia quando iniziamo a sviluppare in HTML, sia quando abbiamo a che fare con la manutenzione di lavori già fatti

- Per ogni lavoro avere gli attrezzi giusti è fondamentale

EDITOR DI TESTO

- La scelta dell'editor è piuttosto importante, è il principale compagno di strada in questo percorso e tipicamente diventa un'estensione naturale del pensiero
- Ce ne sarebbero numerosi altri da citare, oltre a suggerire alcuni nomi vediamo quali sono le caratteristiche e i criteri che ce li hanno fatti scegliere

SYNTAX HIGHLIGHTING

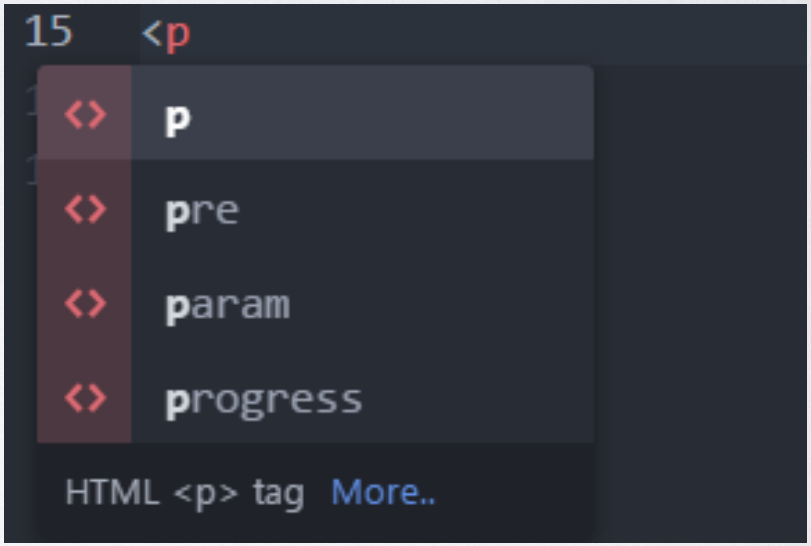
- Serve a evidenziare la sintassi del linguaggio e rendere più leggibile il codice sorgente
- Ogni tag, attributo, numero o parola chiave assume uno specifico colore e diventa subito riconoscibile
- Grazie a questa caratteristica emergono più facilmente eventuali errori di battitura e diventa anche più chiara la struttura della pagina

```html
<!doctype html>
<html lang="it">
<head><title>Ciao Mondo!</title></head>
<body>
  <h1>Ciao Mondo!</h1>
  <p>Questa è la nostra prima pagina HTML!</p>
```

AUTOCOMPLETAMENTO

- Mentre scriviamo il codice l'editor propone di completare automaticamente la parola o il comando che stiamo digitando. Ci sono diversi livelli di autocompletamento anche in base ai linguaggi.

- In HTML ad esempio può risultare comoda la chiusura automatica dei tag: quando finiamo di scrivere viene subito scritto automaticamente lasciandoci la comodità di scrivere dentro altro contenuto.



COMMUNITY E PLUGIN

- Anche se gli editor offrono poche funzioni di base, le esigenze possono portarci a voler aggiungere alcune caratteristiche e diventa importante valutare quanti e quali plugin sono disponibili, quanto è numerosa la base di utenti e quanto è attiva la comunità degli sviluppatori. Un plugin a volte può essere determinante

- Tra i plugin più utili ci sono ad esempio quelli che permettono di memorizzare e replicare degli snippet di codice (o il testo lorem ipsum ad esempio), quelli che permettono il confronto tra versioni diverse dei file (diff).

MACRO E SHORTCUT

- Le scorciatoie da tastiera sono estremamente importanti quando si sviluppa
- Tutti gli editor mettono a disposizione le combinazioni di tasti per le operazioni più comuni: copia, incolla, taglia, undo etc.
- Possiamo aver bisogno però di ulteriori automatismi, come richiamare uno snippet di codice, selezionare direttamente un tag etc
- Per ottenere questo può rivelarsi determinante un editor che consenta di registrare sequenze di azioni (macro) da poter associare a combinazioni di tasti (shortcut).

BROWSER

- **Il programma che interpreta il markup delle nostre pagine HTML e mostra a video tutto ciò che indichiamo si chiama Browser**
- La parola tradotta letteralmente sta per "**colui che dà un'occhiata in giro**" ed è stata utilizzata per definire il tipo di programmi che usiamo per navigare sul Web.
- Per capire meglio come lavorare in HTML è importante soffermarci su alcuni degli aspetti più importanti che riguardano proprio questi strumenti, diventati nel tempo molto di più che semplici software per la "navigazione di Internet"
- Tra i compiti svolti dal browser, sono due quelli che ci interessano di più: il caricamento e la visualizzazione (rendering) della pagina HTML

BROWSER: CARICARE HTML

- Caricare la pagina significa acquisirne il contenuto, più tecnicamente il browser richiede che venga effettuato un trasferimento di file
- Perché sia possibile effettuare le richieste, la pagina HTML, come pure tutti i file ad essa correlati, è raggiungibile facendo riferimento ad un certo indirizzo, come questi che seguono:
- http://www.sito.it/guida/introduzione.html
- file:///C:/Sito/GUIDA/introduzione.html
- Semplificando possiamo dire che grazie agli indirizzi possiamo effettuare richieste:
  - **in "locale"**, quando i file si trovano sul device o sul computer su cui gira il browser
  - **in "remoto"**, quando i file sono su un server da contattare tramite internet (o altra rete).

BROWSER: CARICAMENTO

- La fase di caricamento della pagina è il primo punto cruciale quando si parla di perfomance
- un sito o una applicazione risultino gradevoli da utilizzare bisognerà fare in modo di minimizzare quanto più possibile i tempi di caricamento
- Alcuni di questi accorgimenti come la cache sono inseriti anche nei browser. Inoltre tutti i browser inoltre forniscono dei tool per gli sviluppatori che consentono, tra le altre cose, di monitorare i tempi di caricamento dei file

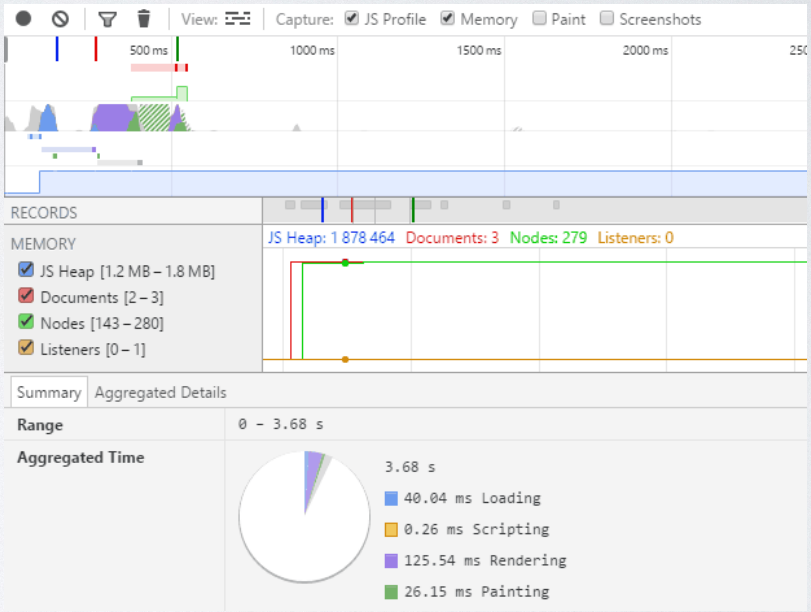
Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
02.%20II%...	Finished	document	Other	0 B	8 ms	
css?family...	200	stylesheet	02. Il Brows...	2.3 KB	157 ms	
page-loadi...	Finished	png	02. Il Brows...	0 B	3 ms	
data:image...	200	svg+xml	02. Il Brows...	(from cache)	0 ms	

BROWSER: RENDERING

- La seconda fase cruciale è quella della visualizzazione ovvero del cosiddetto rendering della pagina, la fase in cui il browser interpreta i documenti HTML e dispone sullo schermo gli elementi (testi, immagini, filmati) a seconda delle indicazioni ivi contenute.

- Il componente del browser che svolge questo compito è detto "Rendering Engine" (motore di rendering), che in sostanza si comporta come un sistema operativo in miniatura e che sfrutta direttamente le caratteristiche hardware del device su cui visualizziamo le nostre pagine.

- In questa fase quindi, specie nei progetti più complessi o quando le pagine sono destinate a dispositivi lenti o con poca memoria, bisogna fare attenzione a non sovraccaricare di lavoro l'engine del browser

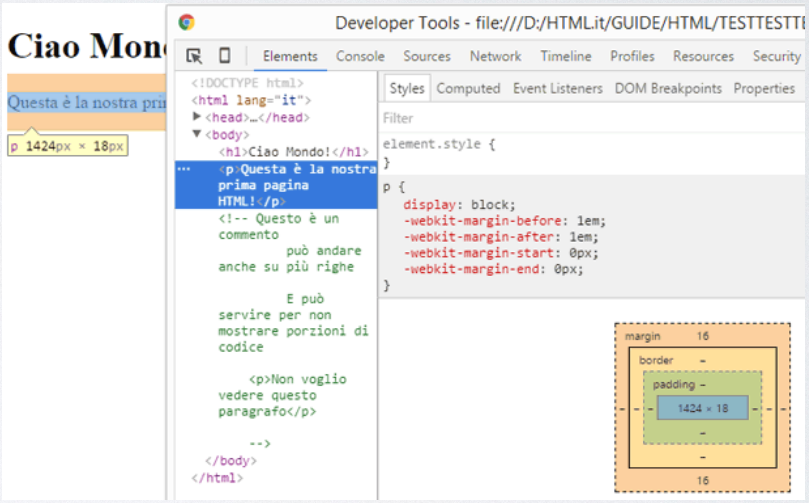


BROWSER PER SVILUPPATORI

- Il browser è fondamentale per testare il lavoro sotto il profilo visuale e non solo. I browser moderni come Google Chrome, Mozilla Firefox, Microsoft Edge e Apple Safari, forniscono gli "strumenti di sviluppo" grazie ai quali possiamo tenere sotto controllo diversi aspetti delle pagine HTML.

- Tra questi strumenti troviamo moltissime cose, dall'analisi del caricamento della pagina all'emulazione dei sensori dei device, dal debug step-by-step del codice JavaScript agli editor di SVG.

- L'esplorazione del codice e modifica dei CSS: Possiamo sezionare la pagina e vedere cosa c'è dietro a ogni elemento. Inoltre possiamo esaminare e modificare il CSS per effettuare delle modifiche e verificare "al volo" gli effetti.



BROWSER: DIFFERENZE

- Anche se praticamente tutti i browser moderni garantiscono una pressoché totale aderenza agli standard del W3C, rimangono ancora alcune piccole differenze nell'implementazione delle direttive, quindi nel modo in cui i singoli browser mostrano i contenuti.
- In sostanza quando lavoriamo al nostro HTML, possiamo trovarci a fare i conti con:
  - la compatibilità dei browser (specie quelli più datati possono dare problemi);
  - le capacità dei dispositivi (in termini di performance e banda ad esempio);
  - le tipologie di schermo cui sono destinate le pagine HTML.
- Se in passato l'idea dominante era quella di tentare di ottenere la stessa visualizzazione su tutti i browser, oggi è fondamentale che i contenuti siano proposti in modo adeguato alla singola situazione (es. mobile, large screen, desktop, vecchi browser, etc.).

STRUMENTI DI SVILUPPO

- Gli ambienti di sviluppo che prevedono un engine hanno in genere un debugger integrato che consente di analizzare il codice durante l'esecuzione
- In assenza di un ambiente di sviluppo integrato, possiamo comunque ricorrere a un comune browser, dal momento che i più diffusi prevedono ormai un ambiente di debugging

CONSOLE

- Tutti i browser prevedono un ambiente per l'esecuzione diretta di codice JavaScript, la cosiddetta console.
- In questa finestra speciale possiamo scrivere codice JavaScript per vederne l'immediata esecuzione
- La stessa console può essere utilizzata come ambiente per la visualizzazione dell'output di uno script, grazie all'omonimo oggetto console

```
var a = 'pippo';
console.log('a = '+a); // stamperà in console: "a = pippo"
```

SINTASSI

- Un programma JavaScript è composto da una sequenza di istruzioni terminate dal punto e virgola (;)
- Un insieme di istruzioni delimitate da parentesi graffe “{“ e “}” costituiscono un blocco
- Il linguaggio è case-sensitive, ovvero distingue le lettere maiuscole dalle minuscole

SINTASSI

- Un identificatore è una sequenza di simboli che identifica un elemento all'interno del programma e può essere composto da lettere e numeri, ma:
  - non deve contenere elementi di punteggiatura o spazi
  - deve cominciare con una lettera.
- <u>Identificatori validi</u>
  nome
  Codice_Fiscale
  a0
- <u>Identificatori non validi</u>
  0a
  x.y
  Partita IVA

COMMENTI

- JavaScript prevede delle sequenze di caratteri per inserire commenti nel codice. Tutto ciò che viene marcato come commento non viene preso in considerazione dall'interprete JavaScript

- Possiamo inserire due tipi di commenti nel codice:

- **commento per singola riga:** Il commento a singola riga (inline) inizia con i caratteri // (doppio slash)
  // Questo è un commento
  var x = 5;
  var x = 5; // Questo è un commento

- **commento multiriga:** prevede la sequenza iniziale /* (slash e asterisco) e si conclude con */ (asterisco e slash). Tutto ciò che viene inserito all'interno di questa coppia di terminazioni sarà considerato un commento indipendentemente dal numero di righe utilizzate

ASSEGNAZIONE

- L'assegnazione è l'operazione fondamentale nei linguaggi di programmazione imperativa
- id = expr
- Dapprima viene "valutata" l'espressione expr, quindi il risultato viene "assegnato" alla variabile id
- Si consideri il seguente frammento di codice

```
var x = 10;
var y = 7;
var z = 3*(x-y);
```

- Al termine dell'esecuzione la variabile z assume il valore 9

ASSEGNAZIONE CONDIZIONALE, OPERATORE TERNARIO

- L'unico operatore ternario previsto da JavaScript è l'operatore condizionale. Esso restituisce un valore in base ad una espressione boleana. La sua sintassi è:
- condizione ? valore1 : valore 2
- Se condizione è vera viene restituito valore1, altrimenti valore2.
- Ad esempio, la seguente espressione ha valore "pari" o "dispari" in base alla divisibilità per due della variabile x:
- x%2 == 0 ? "pari" : "dispari"

ESPRESSIONI

- Un'espressione è composta da
- – identificatori di variabili
  Esempio: x, nome, eta, importo, ...
- – costanti
  Esempio:10, 3.14, "<HTML>", ...
- – operatori
  Esempio: +, –, *, /, &, |, ! e le parentesi (e) per alterare le regole di precedenza tra operatori

OPERATORI ARITMETICI

- Operano tra valori (costanti o variabili) numerici

  – Somma (1+1)
  – Sottrazione (3-5)
  – Moltiplicazione (3*4)
  – Divisione (6/4)
  – Modulo (6%5)
  – Cambiamento di segno (-3)

VARIABILI

- Una variabile è un'area di memoria contenente informazioni che possono "variare" nel corso dell'esecuzione del programma.
- Una variabile è caratterizzata da un nome identificativo e dal tipo dell'informazione contenuta
- Una variabile è identificata da un nome che deve rispettare alcune regole:
  - non deve coincidere con una delle parole chiave del linguaggio;
  - non può iniziare con un numero;
  - non può contenere caratteri speciali

DEFINIZIONE DI VARIABILI

- JavaScript è un linguaggio "debolmente tipizzato" e prima di essere adoperata una variabile deve essere definita
  var eta;

- La definizione stabilisce il nome della variabile, mentre il tipo dipende dall'assegnazione
  eta = 35; //intero
  nome = "Mario Rossi"; //stringa

- Il tipo di una variabile dipende dall'ultima assegnazione, quindi una variabile può cambiare tipo nel corso del suo ciclo di vita
  x = 10; //intero
  x = "a"; //stringa

- JavaScript non prevede la dichiarazione obbligatoria delle variabili, cioè un'operazione esplicita di creazione. Il semplice utilizzo di un identificatore indica all'engine di creare implicitamente la variabile, se non esiste già. Tuttavia questa pratica può favorire errori ed ambiguità.

OPERATORI PRE/POST INCREMENTO DECREMENTO

- Derivano dal linguaggio C (e sono presenti in C++ e Java)
- Alterano e restituiscono il valore di una variabile
- Consentono di scrivere codice compatto ed efficiente

```
var x = 10;
var y = x++; //y=10 e x=11
var z = ++x; //z=12 e x=12
var w = x--; //w=12 e x=11
var v = --x; //v=10 e x=10
```

COSTANTI

- Fino alla versione 5 di ECMAScript non è presente il concetto di costante, cioè di un valore non modificabile. È necessario ricorrere alle variabili usando una notazione convenzionale per indicare che si tratta di valori da non modificare, ad esempio l'uso delle maiuscole:
  var PIGRECO = 3.14; //JavaScript non garantisce la sua immutabilità.

- A partire dalla versione 6 dello standard viene introdotta la possibilità di dichiarare costanti tramite la parola chiave const:
  const PIGRECO = 3.14;

- Questo garantisce l'utilizzo in sola lettura del valore, evitando rischi di modifica durante l'esecuzione.

TIPI DI VARIABILI

- Number
- Boolean
- Null
- String
- Date
- Array

TIPO NUMBER

- Una variabile di tipo Number assume valori numerici interi o decimali

  var x = 10; //valore intero

  var y = -5.3; //valore decimale

- Sono definite le operazioni aritmetiche fondamentali ed una serie di funzioni matematiche di base

TIPO BOOLEAN

- Una variabile di tipo Boolean assume i soli valori della logica booleana vero e falso

  var pagato = true; //valore logico vero

  var consegnato = false; //valore logico falso

- Sono definite le operazioni logiche fondamentali (AND, OR e NOT)

TIPO BOOLEAN

X	Y	AND
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

X	Y	OR
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

X	NOT
TRUE	FALSE
FALSE	TRUE

TIPO NULL

- Si tratta di un tipo che può assumere un unico valore
  var a = null;

- Serve ad indicare che il contenuto della variabile è non significativo
  var sesso = “f”;
  var militesente = null;

TIPO STRING

- Una variabile di tipo String contiene una sequenza arbitraria di caratteri ed è delimitato da apici (') o doppi-apici ("")

  var nome = "Mario Rossi";

  var empty = ""; //Stringa vuota

  var empty2 = new String(); //Stringa vuota

  var str = 'Anche questa è una stringa';

  var str2 = new String("Un'altra stringa");

TIPO DATE

- Una variabile di tipo Date rappresenta un istante temporale (data e ora)

  var adesso = new Date();

  var natale2012 = new Date(2017,11,25);

  var capodanno2013 = new Date("Gen 1 2013");

- È definito l'operatore di sottrazione (-) tra due date che restituisce la differenza con segno espressa in millisecondi

TIPO ARRAY

- Gli array consentono di associare più valori ad un unico nome di variabile (o identificatore). In genere i valori contenuti in un array hanno una qualche affinità (ad esempio l'elenco dei giorni della settimana)
- var giorniSettimana = ["lunedì", "martedì", "mercoledì", "giovedì", "venerdì", "sabato", "domenica"];
- I valori dell'array sono delimitati da parentesi quadre e separati da virgole
- È un modo di definire un array basato sulla rappresentazione letterale, più avanti vedremo la rappresentazione come oggetto
- Una volta definito un array possiamo accedere ai singoli elementi facendo riferimento al nome della variabile e all'indice corrispondente all'elemento, ricordandoci che la numerazione degli indici parte da zero

TIPO ARRAY

- Un array può essere vuoto, cioè senza elementi, ed è rappresentato dalle sole parentesi quadre [ ].
- Se definiamo un array nel seguente modo:
  var myArray = [,''elemento"]; //array con due elementi di cui il primo è undefined
- Se invece l'elemento finale non è indicato, come nel seguente esempio:
  var myArray = ["elemento", "altroelemento",];
  JavaScript ignorerà l'ultima virgola generando un array di due soli elementi.

TIPO ARRAY

- Gli elementi di un array possono essere di qualsiasi tipo di dato previsto da JavaScript, quindi possiamo avere array con elementi di diverso tipo:
  var myArray = [123, "stringa", true, null];

- Un elemento di un array può anche essere un altro array:
  var myArray = [123, "stringa", ["a", "b", 99]];

- In questo caso, per accedere ad esempio all'elemento 99 dovremo accedere prima al terzo elemento dell'array principale e poi al terzo elemento dell'array contenuto in esso:
  var novantanove = myArray[2][2];

TIPO ARRAY

- A partire dalla versione 6 di ECMAScript abbiamo la possibilità di assegnare contemporaneamente a più variabili i valori contenuti in un array
- Sfruttando questo meccanismo, detto destrutturazione dell'assegnamento, possiamo riscrivere gli assegnamenti precedenti con la seguente istruzione:
- var [primoGiorno,,terzoGiorno,,,,settimoGiorno] = giorniSettimana;

TIPO ARRAY

- Un array è un vettore monodimensionale di elementi di tipo arbitrario

  var v = new Array(); //Vettore vuoto

  var w = new Array("Qui","Quo","Qua");

  var u = new Array("Lun","Mar","Mer","Gio","Ven","Sab","Dom");

- Non è necessario specificare la dimensione come lo è in alcuni altri linguaggi di programmazione

OPERATORI SU STRINGHE

- L'unica operazione possibile in un'espressione è la concatenazione di stringhe

  nomeCompleto = titolo + " " + nome + " " + cognome
  indirizzo = recapito + ", " + numCivico + " " + CAP + " - " + citta + " (" + prov + ")"

- Esiste una forma più compatta per esprimere l'accodamento di
  str = str + newStr;
  è equivalente
  str += newStr;

OPERATORI SU VETTORI

- L'operatore definito sui vettori è l'accesso ad un determinato elemento dato l'indice
  v[3] = 10;
  a[i] = b[i] * c[i];
  p = v[1];

- Da ricordarsi che il primo elemento di un vettore ha sempre l'indice 0 (come in C/C++ e Java)

- Se si assegna un valore ad un elemento non presente questo viene creato
  var v = new Array();
  v[0] = 1;

- Se si accede al valore di un elemento non presente si ottiene un valore indefinito
  var v = new Array();
  var a = v[0];

OPERATORI RELAZIONI

- Confrontano due valori e restituiscono l'esito come valore booleano
  - Uguaglianza (==)
  - Disuguaglianza (!=)
  - Minore (<)
  - Maggiore (>)
  - Minore o uguale (<=)
  - Maggiore o uguale (>=)

OPERATORI LOGICI

- Operano tra valori (costanti o variabili) booleani
- AND (&&)
- OR (||)
- NOT (!)
- Solitamente gli operandi sono l'esito di un confronto
- L'utilizzo di operatori relazionali e logici consente di formulare delle condizioni che possono essere utilizzate per controllare l'esecuzione del programma

  (metodoPagamento=="contrassegno")&& (!residenteInItalia)

BLOCCHI DI CODICE

- Più istruzioni possono essere raggruppate in blocchi che fungono come una singola istruzione
- I blocchi di codice sono importanti quando si vuole evidenziare che un gruppo di istruzioni deve essere eseguito interamente, ad esempio in seguito alla valutazione di una condizione.
- Un blocco di codice è racchiuso tra parentesi graffe come nel seguente esempio:

```
{
    x = x + 3;
    y = x * 2;
}
```

- Al termine di un blocco di codice non va inserito il punto e virgola. Questa è l'unica eccezione all'uso del punto e virgola come delimitatore di istruzioni.

CONTROLLO DELL'ESECUZIONE

- L'esecuzione di un programma è generalmente sequenziale ma in determinate "condizioni" può essere necessario eseguire solo alcune istruzioni, ma non altre, oppure ripetere più volte un'operazione
- L'istruzione instr viene eseguita solo se la condizione cond risulta vera
- if(cond)
  instr
- L'istruzione instr può essere sostituita da un gruppo di istruzioni tra parentesi graffe { e }

CONTROLLO DELL'ESECUZIONE

- Una costruzione alternativa prevede la presenza di una seconda istruzione da eseguire nel caso la condizione risulti falsa

```
if(cond)
    instr_then
else
    instr_else
```

- Condizioni in cascata:

```
if (condizione1) {
    istruzioni1
} else if (condizione2) {
    istruzioni2
} else {
    istruzioni3
}
```

CONTROLLO DELL'ESECUZIONE

```
if (x >= 37) {
    messaggio = "Hai la febbre.\n";
    messaggio = messaggio + "Temperatura: " + x + " gradi."
}
else {
    messaggio = “Puoi andare al lavoro.\n”;
}
```

SWITCH CASE

- Quando siamo di fronte a diverse alternative, anche un if a cascata può risultare difficile da leggere. In questi casi possiamo ricorrere all'istruzione switch, il cui schema sintattico è:

```
switch (espressione) {
    case espressione1:
        istruzioni1;
    break;
    case espressione2:
    case espressione3:
        istruzioni2;
    break;
    // ...
    default:
        istruzioni4;
    break;
}
```

- L'espressione di riferimento dello switch viene confrontata in sequenza con le espressioni dei vari case. Non appena viene individuata un'espressione corrispondente si esegue il blocco di istruzioni associato. Se non si trova nessuna corrispondenza verrà eseguito il blocco associato alla parola chiave default. La presenza di un caso di default è comunque opzionale.

- È importante però inserire l'istruzione break al termine del blocco di istruzioni per evitare la prosecuzione con il caso successivo.

CICLO ITERATIVO WHILE

- L'istruzione viene eseguita finché la condizione condizione risulta essere verificata

```
while (condizione) {
  // istruzioni
}
```

- Finché la condizione sarà vera verranno eseguite le istruzioni contenute nel blocco di codice

- Requisito fondamentale nell'uso del while è che le istruzioni contenute nel blocco di codice modifichino la condizione, altrimenti si rischia di incorrere in un ciclo infinito

CICLO ITERATIVO DO - WHILE

- Una variante del while è il do…while:

```
do {
    // istruzioni
} while (condizione);
```

- La differenza sostanziale rispetto al while classico consiste nel fatto che la condizione viene valutata dopo aver eseguito le istruzioni.

- Garantisce che il blocco di codice verrà eseguito almeno una volta.

CICLO ITERATIVO FOR

- Viene dapprima eseguita l'istruzione inizializzazione, quindi l'istruzione e viene ripetuta finché la condizione risulta vera, dopo ogni ripetizione viene eseguita l'istruzione di modifica

```
for (inizializzazione; condizione; modifica) {
    // istruzioni
}
```

- Inizializzare a 0 gli n elementi del vettore a

```
for(var i=0; i<n; i++)
  a[i]=0;
```

- Copiare gli n elementi del vettore a nel vettore b

```
for(var i=0; i<n; i++)
  b[i]=a[i];
```

FOR IN E FOR OF

- Sfruttando questa variante del for non abbiamo bisogno di specificare la lunghezza dell'array né l'istruzione di modifica della condizione. JavaScript rileva che la variabile quantità è un array ed assegna ad ogni iterazione alla variabile indice il valore dell'indice corrente.

```
var quantita = [12, 34, 45, 7, 19];
var totale = 0, indice;
for (indice in quantita) {
    totale = totale +  quantita[indice];
}
```

- Utilizzando il for..of invece possiamo scrivere il ciclo in questo modo:

```
var quantita = [12, 34, 45, 7, 19];
var totale = 0, valore;
for (valore of quantita) {
    totale = totale +  valore;
}
```

- Ad ogni iterazione JavaScript assegna alla variabile valore il contenuto di ciascun elemento dell'array.

- C'è da tenere presente, però, che il for...of fa parte delle specifiche di ECMAScript 6 e potrebbe non essere disponibile in engine JavaScript meno recenti

CONSIDERAZIONE SUI CICLI

- **Cicli infiniti:** Sono rari i casi in cui si richiede l'uso di un loop infinito e naturalmente bisogna fare molta attenzione ad utilizzarlo perché può causare blocco dell'istanza in esecuzione

```
while(true) {
    // istruzioni
}
```

- Per questo bisogna sempre considerare l'ipotesi di avere una condizione di uscita, che causi l'interruzione del loop.

- **Interrompere un ciclo, break:** Il comando break interrompe un blocco di istruzioni e forza l'engine ad eseguire la prima istruzione che segue il blocco

```
var x=0;
while (true) {
    // condizione di uscita
    if (x > 100) break;
    x++;
}
```

- **Continue:** Il comando continue, serve per interrompere l'esecuzione della iterazione, saltando le istruzioni che seguono e riprendere l'iterazione successiva

```
var x = 0;
while (x < 10) {
    x++;
    if (x > 3) continue;
    // se x è maggiore di 3, questa istruzione non viene più eseguita
    prova = "ciao";
}
```

FUNZIONI

- una funzione è un insieme di istruzioni racchiuse in un blocco di codice, che può essere contraddistinto da un nome, può accettare argomenti o parametri di ingresso e restituire valori

- L'utilizzo delle funzioni nella programmazione strutturata aumenta la modularità e favorisce il riutilizzo

```
function nome(arg0, arg1, . . . , argn) {
    // istruzioni
}
```

- La funzione definita è identificata da name e dipende dagli argomenti arg0, arg1, ..., argn. L'esecuzione vera e propria avviene con l'invocazione o chiamata, il cui schema sintattico è: **nome(valori);**

- Gli eventuali valori inseriti nella chiamata di una funzione vengono passati, cioè assegnati, ai corrispondenti argomenti della definizione della funzione.

- **Return**: Nel corpo della funzione può essere presente l'istruzione return che consente di terminare e restituire un valore al codice che l'ha chiamata

INVOCAZIONE DI FUNZIONI

- Quando una funzione viene invocata gli argomenti sono inizializzati con i valori specificati, si procede con l'esecuzione delle istruzioni costituenti la funzione e l'istruzione return restituisce il valore calcolato al chiamante
- La chiamata x = somma(1, 2)
- inizializza gli argomenti a e b rispettivamente ai valori 1 e 2
- L'istruzione return a+b;
- valuta l'espressione e restituisce il risultato (3) che viene assegnato alla variabile x

FUNZIONI

```
function somma() {
    var z = 11 + 5;
    return z;
}

function somma(x, y) {
    var z = x + y;
    return z;
}

var risultato = 5 + somma() * 2;
var risultato = somma(11, 5);
```

PROCEDURE

- Una funzione che non restituisce valori viene detta procedura (in analogia con il Pascal)
- Una procedura agisce in genere su variabili globali (side-effect)

PARAMETRI OPZIONALI

- Gli argomenti di una funzione, non è obbligatorio passarli tutti in una chiamata di funzione, possiamo invocare la funzione nei seguenti modi:

```
var risultato1 = somma(11);
var risultato2 = somma();
var risultato3 = somma(11, 5, 7, 9);
```

- Non generano nessun errore in JavaScript, assegna automaticamente al secondo argomento mancante il valore undefined restituendo come somma il valore NaN

- Se passiamo più argomenti di quelli previsti nella definizione della funzione, JavaScript ignorerà gli argomenti in più restituendo lo stesso risultato di somma(11, 5)

PARAMETRI OPZIONALI

- Una analoga flessibilità l'abbiamo anche nella dichiarazione degli argomenti quando definiamo una funzione, non specifichiamo gli argomenti a fianco al nome della funzione, ma accediamo ad essi sfruttando questo array predefinito arguments e consente di creare funzioni con un numero di parametri non definito

```
function somma() {
    var z = 0, i;
    for (i in arguments)
        z = z + arguments[i];
    return z;
}
```

- In questo modo potremo chiamare la funzione per sommare un qualsiasi numero di valori:

```
somma(2, 78);
somma(17, 32, 4, 19, 52);
```

PARAMETRI DEFAULT E REST

- **Argomenti con valori di default:** Con l'avvento di ECMAScript 6 è stata introdotta la possibilità di specificare dei valori di default:

```
function somma(x = 0, y = 0) {
    var z = x + y;
    return z;
}
```

- **Rest parameter:** Si tratta di una notazione speciale per indicare un elenco indefinito di argomenti aggiuntivi

- Supponiamo di definire una funzione che implementa diverse operazioni aritmetiche e prende come primo argomento il nome dell'operazione da eseguire ed a seguire un numero variabile di valori

```
function eseguiOperazione(x, ...y) {
    var z = 0;
    switch (x) {
        case "somma":
            for (i in y) z = z + y[i];
            break;
        case "moltiplica":
            for (i in y) z = z * y[i];
            break;
        default:
            z = NaN; break;
    }
    return z;
}
```

PARAMETRI REST

- La notazione dell'argomento preceduto dai puntini cattura l'elenco degli argomenti successivi al primo e lo rende disponibile all'interno della funzione sotto forma di array.

- L'approccio è simile all'array predefinito arguments, ma mentre questo cattura tutti gli argomenti della funzione, il rest parameter cattura soltanto gli argomenti in più rispetto a quelli specificati singolarmente

```
eseguiOperazione("somma", 12, 54, 2, 7, 12);
eseguiOperazione("moltiplica", 4, 11, 32);
eseguiOperazione("dividi", 45, 9, 6, 17);
```

- La prima e la seconda chiamata otterranno rispettivamente la somma ed il prodotto dei valori passati dal secondo argomento in poi, mentre la terza chiamata otterrà la divisione del secondo e terzo argomento, ignorando gli altri

PARAMETRI REST

- La stessa notazione del rest parameter può essere utilizzata nelle chiamate a funzioni che prevedono diversi argomenti. In questo caso si parla di spread operator, cioè di un operatore che sparge i valori contenuti in un array sugli argomenti di una funzione, come nel seguente esempio:

```
var addendi = [8, 23, 19, 72, 3, 39];
somma(...addendi);
```

- La chiamata con lo spread operator è equivalente alla seguente chiamata:

```
somma(8, 23, 19, 72, 3, 39);
```

- Ricordiamo che rest parameter e spread operator fanno parte delle specifiche di ECMAScript 6 e che quindi potrebbero non essere ancora supportati da tutti i browser

SCOPE DELLE VARIABILI

- All'interno di una funzione è possibile definire delle variabili "confinate" all'interno della funzione stessa
- Tali variabili, dette locali, sono create all'atto dell'invocazione della funzione e sono distrutte al termine dell'esecuzione
- Il loro valore non è accessibile dall'esterno della funzione
- Ogni argomento di una funzione è una variabile locale definita implicitamente
- Un'altra osservazione, legata in qualche modo a questa appena accennata, è che a differenza di quanto accade in altri linguaggi come ad esempio C e C#:
- **dichiarare una variabile all'interno di un blocco di codice non crea un nuovo scope per la variabile!**

SCOPE DELLE VARIABILI

- Le variabili definite all'esterno di una funzione sono denominate, invece, globali
- A differenza delle variabili locali, le variabili globali sono accessibili da qualsiasi punto del programma, anche dall'interno di una funzione, sempre che in quest'ultima non sia stata definita una variabile locale con lo stesso nome
- Si consideri il seguente frammento di codice

```
function f(...) {
    ...
    var x = 1;...
}
var x = -1;
...
f(...);
```

- La variabile globale x continua a valere -1

LET

- Se invece abbiamo bisogno di creare uno scope specifico per una o più variabili possiamo ricorrere all'istruzione let

- Let consente di dichiarare una o più variabili in modo analogo a var, ma a differenza di quest'ultima limita lo scope della variabile al blocco di codice, all'istruzione o all'espressione in cui viene utilizzata. Quindi, ad esempio, nel seguente codice:

```
var x = 10;
var y;
let (x= 20) {
    y = x + 1;
}
y = x + y;
```

- avremo uno scope a livello di blocco di codice in cui la variabile x dichiarata tramite let nasconde quella esterna dichiarata con var. Il risultato finale combina i valori delle due variabili omonime.

LET

- Un ambito in cui è senz'altro utile utilizzare l'istruzione let è nelle iterazioni, come ad esempio nel for:

```
var x = 0;
for (let i = 0; i < 10; i++) {
    x = x + 1;
}
```

- Grazie a let dichiariamo ed utilizziamo la variabile i soltanto all'interno del for, evitando eventuali collisioni con altre variabili omonime definite in scope più esterni.

- Come possiamo intuire, la distinzione tra scope globale e locale delle variabili è in realtà troppo semplicistica

CONVERSIONE DEI DATI

- Le variabili JavaScript possono contenere valori di qualsiasi tipo e cambiare senza problemi il tipo di dato contenuto e lascia il compito all'engine di prendere delle decisioni in determinate situazioni ed effettuare delle conversioni implicite.

```
var x = "12";
var y = "13";
var z = x * y;
```

- Il valore che ci aspettiamo nella variabile z è il risultato della moltiplicazione dei valori numerici risultanti dalla conversione delle stringhe contenute in x e y. Ed in effetti questo è il comportamento di JavaScript

```
var x = "12";
var y = x + 13;
```

- La variabile y conterrà la stringa "1213"

CONVERSIONE DEI DATI

- Le conversioni implicite di JavaScript sono spesso fonte di bug
- Il modo migliore per non incappare in questi potenziali bug è evitare le conversioni implicite
- È preferibile confrontare espressioni e valori dello stesso tipo ed utilizzare sempre gli operatori di uguaglianza e disuguaglianza stretta invece di quelli ordinari, in modo da essere sicuri di quello che si sta confrontando.

UGUAGLIANZA STRETTA

- Gli operatori di uguaglianza e disuguaglianza stretta (=== e !==).
- Confrontano gli operandi senza effettuare alcuna conversione
- Quindi due espressioni vengono considerate uguali soltanto se sono dello stesso tipo ed rappresentano effettivamente lo stesso valore

TYPEOF

- In qualsiasi momento si può verificare il tipo della variabile tramite l'operatore typeof
- I valori che restituisce possono essere: "string", "boolean", "number", "function", "object", "undefined", ''xml"

```
var prova = new Function();
var numero = 1;
var carattere = "Salve";

console.log(typeof prova);     // ritorna "function"
console.log(typeof numero);    // ritorna "number"
console.log(typeof carattere); // ritorna "string"
```

FUNZIONI PREDEFINITE

- In JavaScript sono presenti alcune funzioni predefinite
  - isNaN(v) verifica se v non è un numero
  - isFinite(v) verifica se v è finito
  - parseFloat(str) converte str in un numero decimale
  - parseInt(str) converte str in un numero intero

ENCODEURI E DECODEURI

- Quando una stringa rappresenta un URI, la funzione escape() è deprecata inoltre verrebbero codificati alcuni caratteri speciali, come ad esempio ? e #, che hanno un significato speciale

- In presenza di un URI utilizziamo la funzione encodeURI() che esclude dalla codifica i caratteri , /?:@&=+$#. Nel seguente esempio:

```
var escapedURI = escape("http://www.html.it/a b.html");
var encodedURI = encodeURI("http://www.html.it/a b.html");
```

- la variabile escapedURI avrà il valore http%3A//www.html.it/a%20b.html mentre encodedURI conterrà il valore http://www.html.it/a%20b.html.

- La controparte di encodeURI() è la funzione decodeURI() che, come è intuibile, restituisce la stringa decodificata

ENCODEURI E DECODEURI

- La funzione encodeURIComponent() codifica anche i caratteri speciali esclusi dalla encodeURI().
- Essa è pensata per codificare i valori di eventuali parametri passati in un URI, come mostrato nel seguente esempio:

```
var param = encodeURIComponent("Che cos'è?");
var encodedURI = encodeURI("http://www.html.it/a b.php?x=") + param;
```

- Il valore finale della variabile encodedURI sarà:
- http://www.html.it/a%20b.php?x=Che%20cos'%C3%A8%3F
- Da notare come il valore originario codificato da encodeURIComponent() conteneva il simbolo del punto interrogativo (?) che se non codificato avrebbe generato un URI malformato.

EVAL

- Tra le funzioni predefinite, eval() ha un ruolo controverso, prende come argomento una stringa e la valuta o la esegue come se fosse codice JavaScript. La stringa può rappresentare un'espressione o una serie di istruzioni, come mostrato nel seguente esempio:

```
var x = eval("5 + 1");
var z = eval("x + 1");
eval("var a = 4; a = a *3;");
a = a + 1;
```

- La prima istruzione valuta un'espressione numerica e la assegna alla variabile x, che assume quindi il valore di 6. La seconda istruzione valuta una stringa che contiene la variabile x incrementandone il valore di uno. La variabile z avrà il valore di 7. Questo ci fa capire che lo scope di riferimento di eval() è lo scope corrente.

- Infine l'ultima invocazione di eval() esegue due istruzioni JavaScript creando nello scope corrente la variabile a il cui valore finale sarà 13

EVAL

- Le potenzialità di eval() sono enormi, dal momento che mette a disposizione del programmatore un vero e proprio interprete JavaScript
- Insieme alle potenzialità anche il pericolo di avere risultati inattesi e non facilmente controllabili è alto, tanto che è stato detto che eval is evil.
- Diciamo che in linea di massima è opportuno evitare di utilizzare questa funzione perché può creare più problemi di quelli che vorrebbe risolvere

OGGETTI

- Un oggetto è un elemento caratterizzato da uno stato rappresentato mediante proprietà e da un insieme di azioni (o metodi) che può eseguire
- Oggetti caratterizzati dagli stessi metodi e dalle stesse proprietà, ma non dallo stesso stato, sono detti della stessa classe
- JavaScript è un linguaggio orientato agli oggetti
- Molti tipi di dato fondamentali sono, in effetti, degli oggetti (String, Date, Array,...)

PROPRIETÀ E METODI

- Una proprietà di un oggetto è assimilabile ad una variabile che ne definisce lo stato interno di un oggetto
  cliente.nome = "Carlo Bianchi";
  x = ordine.aliquotaIVA;

- Un metodo, invece, è simile ad una funzione
  tot = ordine.calcolaTotale();

- Esistono due sintassi alternative per accedere alle proprietà degli oggetti
  oggetto.proprieta
  oggetto["proprieta"]

- La seconda è utile quando il nome della proprietà viene determinato durante l'esecuzione del programma

OGGETTI MATH

- Proprietà
  - E costante di Eulero
  - PI pi greco
- Metodi
  - abs(val) valore assoluto
  - ceil(val)/floor(val) troncamento
  - exp(val) esponenziale
  - log(val) logaritmo
  - pow(base, exp) elevamento a potenza
  - sqrt(val) radice quadrata

OGGETTI DI TIPO STRING

- Proprietà
  - length lunghezza della stringa
- Metodi
  - charAt(pos) carattere alla posizione pos
  - subString(start, end) sottostringa dalla posizione start alla posizione end
  - toUpperCase()/toLowerCase() converte la stringa in maiuscolo/minuscolo
  - indexOf(str, pos) posizione della prima occorrenza della string str cercata a partire dalla posizione pos

OGGETTI DI TIPO ARRAY

- Proprietà
  - length lunghezza del vettore
- Metodi
  - sort() ordina gli elementi del vettore
  - reverse() inverte l'ordine degli elementi del vettore

OGGETTI DI TIPO DATE

- Metodi
  - getXXX() restituisce il valore della caratteristica XXX della data (es. getFullYear()).
  - setXXX(val) imposta il valore della caratteristica XXX della data (es. setFullYear(2013,1,1));
  - toString() restituisce la data come stringa formattata

nome caratteristica	significato
Date	Giorno del mese
Day	Giorno della settimana
FullYear	Anno
Minutes	Minuti
Hours	Ore
Month	Mese
Seconds	Secondi
Time	Tempo (hh:mm:ss)

ELEMENTI IN HTML

- In una pagina HTML tutti gli elementi sono connotati da tag (letteralmente "etichette")
- Si tratta di marcatori che evidenziano non tanto l'aspetto, quanto il senso, il ruolo, o l'organizzazione che vogliamo assegnare ai contenuti. Ad esempio se indichiamo un titolo con <h1> per noi le parole in quel titolo rappresentano il tema principale della pagina, al di là del modo in cui saranno visualizzate.
- **Questa visione "semantica" degli elementi è fondamentale**, ci aiuta a non perdere di vista il fatto che una pagina deve essere un luogo ordinato di informazioni, soprattutto sul Web. Se pensiamo ad esempio al classico contesto delle ricerche online, risulta naturale comprendere che tanto più una pagina sarà associabile a una keyword o a un tema, tanto più facile sarà trovarla. Per questo sarà importante curare cose come l'organizzazione del testo, la definizione dei titoli, dei link dei grassetti.
- Tuttavia in HTML possiamo anche definire lo scheletro dell'interfaccia utente di una app, e in questo frangente i tag diventano utili come supporto all'organizzazione del layout o alla definizione di aree specifiche per l'esperienza utente.
- In tutti e due i casi continuiamo a non parlare di "grafica", ma di struttura.

```
<HTML>

 <HEAD>
  <TITLE>Title Text</TITLE>
 </HEAD>

 <BODY>
  <H1>H1 Heading</H1>
  <P>Paragraph 1</P>
  <P>Paragraph 2</P>
 </BODY>

</HTML>
```

ESEMPIO

- Le pagine HTML sono quindi formate per composizione di contenitori (annidati) l'uno dentro l'altro

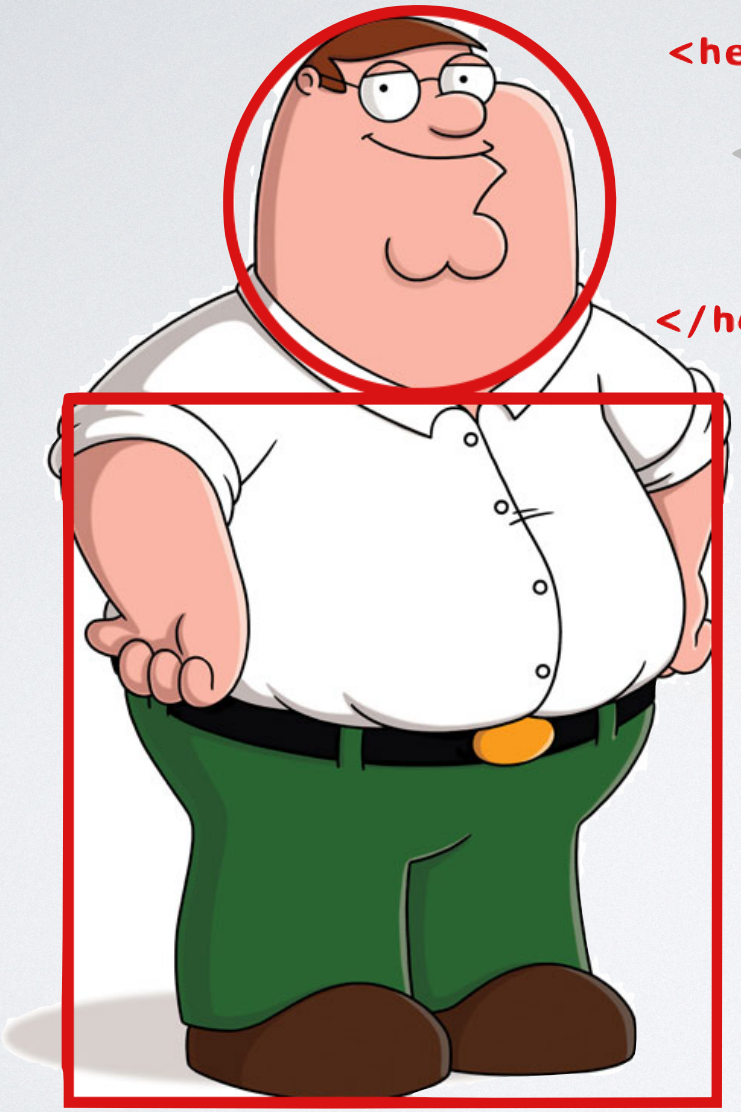
```html
<!doctype html>
<html lang="it">
    <head>
        <title>Ciao Mondo!</title>
    </head>
    <body>
        <h1>Ciao Mondo!</h1>
        <p>Questa è la nostra prima pagina HTML!</p>
        <img src="immagine.jpg">
    </body>
</html>
```

- La struttura che emerge è quella di un albero, in cui i rami sono tutti tag contenitori e le estremità sono composte da testi, immagini o altri elementi come le caselle di input. Tutti questi elementi finali non sono contenitori e non necessitano di un tag di chiusura

ESEMPIO

```
html
|
+---head
|   |
|   +---title
|       |
|       +---"Ciao Mondo!"
|
+---body
    |
    +---h1
        |
        +---"Ciao Mondo!"
        |
        p
        |
        +---"Questa è la nostra prima pagina HTML!"
        |
        img
```

- L'albero che abbiamo tracciato ricorda il modo in cui viene rappresentato il documento nella memoria del browser, ovvero il DOM (Document Object Model), il modello del documento HTML di cui l'elemento <html> è il nodo radice



```
<html>
    <head>
        <title>
            Peter Griffin's HTML page
        </title>
    </head>
    <body>
        Page content goes here..
    </body>
</html>
```

INTEGRAZIONE CON IL BROWSER

- La caratteristica principale di JavaScript è di essere "integrabile" all'interno delle pagine web
- In particolare consente di aggiungere una logica procedurale alle pagine rendendole "dinamiche"
- A differenza di altre tecnologie, JavaScript funziona completamente sul client

INTEGRAZIONE CON IL BROWSER

- I campi di impiego tradizionali sono
  - Validazione dell'input dell'utente e controllo dell'interazione
  - Effetti visivi di presentazione
- Con l'avvento di HTML5 i potenziali usi di JavaScript sono notevolmente aumentati
- JavaScript è supportato da tutti i browser più diffusi

DYNAMIC HTML - DHTML

- L'integrazione degli script all'interno di una pagina web avviene in due modi
  - Associando funzioni JavaScript agli eventi che si intende gestire
  - Accedendo dalle funzioni JavaScript alle proprietà degli oggetti che costituiscono la pagina

DYNAMIC HTML - DHTML

- Una pagina "dinamica" (DHTML) è una pagina HTML contenente codice JavaScript associato a determinati eventi che implementa specifiche funzionalità
- Non bisogna confondere una pagina DHTML con una pagina generata dinamicamente dal server

MODELLO AD AGGETTI

- Un browser web esporta verso JavaScript un modello ad oggetti della pagina e dell'"ambiente" in cui la pagina è visualizzata
  - Una funzione JavaScript adopera tali oggetti invocando i metodi e accedendo alle proprietà
  - Il modello ad oggetti, a differenza del linguaggio, non è standard

MODELLO NAVIGATOR

- L'oggetto navigator rappresenta l'istanza del browser in cui lo script è in esecuzione ed ha le seguenti proprietà:

  - **appCodeName** codice identificativo del browser

  - **appName** nome del browser

  - **appVersion** numero di versione

MODELLO WINDOW

- Questo oggetto rappresenta la finestra in cui il documento corrente viene visualizzato
- Una funzione può accedere alle proprietà della finestra corrente, ma può creare e manipolare nuove finestre (pop-up)
- Proprietà
  - **title** titolo della finestra
  - **statusbar** testo mostrato sulla barra di stato
  - **location** URL del documento visualizzato
  - **outerHeight, outerWidth** dimensioni esterne
  - **innerHeight, innerWidth** dimensioni interne

MODELLO WINDOW

- Modificare il titolo della finestra corrente
  window.title = “Questo è il nuovo titolo”;

- Accedere ad un nuovo documento
  window.location = “http://www.yahoo.com/”;

- Calcolare l’area in pixel della finestra
  var area = window.innerWidth * window.innerHeight;

WINDOW: METODI

- open(location, title) apre una nuova finestra
- alert(message) visualizza il messaggio in una finestra di dialogo (utile per il debug)
- confirm(message) visualizza il messaggio e richiede una conferma all'utente
- moveTo(x, y) sposta la finestra alle coordinate indicate
- resizeTo(width, height) dimensiona la finestra

MODELLO WINDOW

- Creazione e posizionamento di una nuova finestra
  var w = window.open("http://www. google.com/", "Google");
  w.moveTo(0, 0);

- Visualizzazione di un messaggio
  window.alert("Attenzione si è verificato un errore");

- Visualizzazione del browser in uso
  window.alert("Sei connesso con " + navigator.appName + " versione " + navigator.version);

- Richiesta conferma all'utente

```
if(confirm("Vuoi proseguire con l'operazione?")) {
    //L'utente ha risposto Sì
    ...
} else {
    //L'utente ha risposto NO
... }
```

OGGETTO HISTORY

- Rappresenta la sequenza di pagine visitate dall'utente ed è rappresentata mediante un vettore
- Metodi
  - **back()** torna alla pagina precedente
  - **forward()** passa alla pagina successiva

OGGETTO DOCUMENT

- Rappresenta il documento HTML che costituisce la pagina visualizzata
- Non è possibile accedere a tutti gli elementi del documento
- Tuttavia è possibile accedere agli elementi dei moduli (form) ed alle proprietà di visualizzazione
- Inoltre, è possibile costruire on-the-fly il documento prima che questo sia stato completamente caricato e visualizzato

OGGETTO DOCUMENT

- Proprietà
  - **bgColor** consente di impostare il colore dello sfondo
  - **fgColor** colore del testo da visualizzare nel testo
  - **forms** vettore dei moduli presenti nella pagina
  - **title** titolo del documento
  - **URL** indirizzo del documento

- Metodo
  - **write(string)** accoda string al documento, serve per la costruzione on-the-fly

OGGETTO DOCUMENT

- Supponendo che nel documento HTML sia definito un modulo di nome modulo
  <FORM NAME="modulo"...>
  ...
  </FORM>

- Si può accedere a tale oggetto in due diversi modi

  document.forms["modulo"];
  document.modulo;

- Ciò è possibile, in generale, per tutti gli elementi del documento con un attributo NAME

- Dal momento che la proprietà forms è di tipo Array è possibile accedervi anche tramite l'indice numerico dell'elemento

```
for(var i=0; i<document.forms.length; i++) {
   //Accedi a document.forms[i]
   ... = document.forms[i];
   ...
}
```

OGGETTO FORM

- Un oggetto Form corrisponde ad un modulo all'interno di una pagina HTML e tramite le proprietà di questo oggetto è possibile accedere ai diversi elementi (o controlli) del modulo (inputbox, listbox, checkbox, ecc.)

- Proprietà
  - **action** valore dell'attributo ACTION
  - **elements** vettore contenente gli elementi del modulo
  - **length** numero di elementi del modulo
  - **method** valore dell'attributo METHOD
  - **target** valore dell'attributo TARGET

- Metodi
  - **reset()** azzera il modulo reimpostando i valori di default per i vari elementi
  - **submit()** invia il modulo

OGGETTO FORM

- Supponendo che l'i-esimo elemento di un modulo mod sia denominato nome_i è possibile farvi riferimento in 3 modi diversi

```
document.mod.elements[i-1];
document.mod.elements[“nome_i”];
document.mod.name_i;
```

- Attenzione l'indice del primo elemento di un vettore è sempre 0 (quindi l'i-esimo elemento ha indice i-1)
- All'interno di un modulo possono comparire diversi tipi di elementi, corrispondenti ai vari costrutti HTML
- Ogni tipo ha proprietà e metodi specifici, per una trattazione approfondita si rimanda alla guida di riferimento del modello ad oggetti implementato dal browser

ELEMENTI DEI MODULI

HTML	JavaScript
<INPUT TYPE="text">	Text
<TEXTAREA></TEXTAREA>	Textarea
<SELECT></SELECT>	Select
<INPUT TYPE="checkbox">	Checkbox
<INPUT TYPE="radio">	Radio
<INPUT TYPE="button">	Button

ELEMENTI DEI MODULI

- Tutti i tipi di elementi possiedono le seguenti proprietà
  - **name** nome dell'elemento
  - **value** valore corrente dell'elemento

- Gli elementi di tipo Input possiedono la proprietà defaultValue che contiene il valore predefinito del campo (attributo VALUE del tag HTML)

ELEMENTI DEI MODULI

- Gli elementi di tipo Radio e Checkbox possiedono la proprietà checked che indica se l'elemento è stato selezionato
- Gli elementi di tipo Select possiedono la proprietà selectedIndex, che contiene l'indice dell'elemento selezionato nella lista
- la proprietà options, che contiene il vettore delle scelte dell'elenco
- E' possibile modificare i valori contenuti negli elementi dei moduli pertanto è possibile utilizzare questi elementi anche per fornire risultati all'utente
- Se un elemento ha scopi esclusivamente di rappresentazione può essere marcato come READONLY

EVENTI

- Ogni oggetto di un documento HTML "genera" degli eventi in risposta alle azioni dell'utente
- Ad esempio, l'evento click corrisponde al click del puntatore sull'oggetto
- Per gestire l'interazione con l'utente si associano funzioni JavaScript a particolari eventi e gli eventi generati da un oggetto dipendono dal tipo di quest'ultimo
- Oggetti **Form**
  – onSubmit invio del modulo
  – onReset azzeramento del modulo
- Oggetti **Button**
  – onClick click del puntatore
- Oggetti **Select, Text e Textarea**
  – onChange modifica del contenuto
  – onFocus selezione dell'elemento
- Oggetti **Radio e Checkbox**
  – onClick click del puntatore
  – onFocus selezione dell'elemento

JS IN HTML

- Il codice JavaScript viene inserito all'interno di una pagina HTML delimitato dal tag <SCRIPT>...</SCRIPT>
- Per motivi di compatibilità con i vecchi browser il codice è incluso in un commento HTML <!--...-->
- Generalmente il codice è incluso all'interno dell'intestazione (<HEAD>) del documento ed il codice viene eseguito prima della visualizzazione del documento

JS IN HTML

```
<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript"
  LANGUAGE="javascript">
<!--
function f(...);
var v=...;
...
//-->
</SCRIPT>
...
</HEAD>
...
</HTML>
```

JS IN HTML

- Eventualmente il codice JavaScript può risiedere in un documento esterno
  `<SCRIPT TYPE="text/javascript" LANGUAGE="javascript" SRC="URL.js"> </SCRIPT>`
- Ciò è utile per aumentare la modularità e "nascondere" all'utente il codice
- Adoperando il metodo write()dell'oggetto document è possibile costruire in fase di caricamento parte del corpo del documento HTML (<BODY>…</BODY>). Tuttavia questo sistema è poco pratico e se ne sconsiglia l'impiego

INTERCETTARE EVENTI

- Per intercettare l'evento E di un tag T ed associare l'esecuzione di una funzione f() `<T onE="return f();">`
- Se il risultato della valutazione della funzione è false viene interrotta l'esecuzione del commando corrente, ad esempio l'invio di un modulo
- Ad esempio, la funzione valida() verifica se i dati immessi nel modulo modulo sono corretti ed eventualmente procede con l'invio di quest'ultimo
  `<FORM NAME="modulo" onSubmit="return valida();" ...> ... </FORM>`

VALIDAZIONE DEI DATI

- Nel modulo di sottoscrizione del servizio è necessario indicare il nominativo del sottoscrittore
- Quindi il valore del campo corrispondente non deve essere una stringa vuota
- Il modulo viene ridefinito come
  `<FORM NAME="iscrizione" ACTION="" METHOD="POST" onSubmit="return validaIscrizione();">`
- Nell'intestazione del documento viene definita una funzione validaIscrizione()

VALIDAZIONE DEI DATI

```
function validaIscrizione() {
    if(document.iscrizione.nominativo.value==“”) {
       alert(“Campo obbligatorio, impossibile procedere.”);
       return false;
    }
    ...
    //Altri controlli
    return true;
}
```

VALIDAZIONE DEI DATI

- Tra le proprietà degli elementi del modello ad oggetti del documento sono presenti alcune specifiche della modalità di presentazione all'utente degli elementi medesimi
- La possibilità di poter accedere e manipolare tali caratteristiche permette di utilizzare JavaScript per ottenere sofisticati effetti di presentazione visiva ma purtroppo, queste proprietà sono specifiche dei singoli browser
- Le proprietà di visualizzazione sono connesse con l'uso dei CSS, infatti, alcune proprietà degli stili possono essere modificate dinamicamente da funzioni JavaScript
- Tra le proprietà più utili per la creazione di effetti visivi abbiamo la visualizzazione ed il posizionamento degli elementi

ESEMPIO MODULO ISCRIZIONE

- Il modulo deve raccogliere i dati su un utente che vuole sottoscrive un certo servizio e per ogni utente deve richiedere:
  – nominativo, età, sesso
  – se desidera ricevere informazioni commerciali
  – il servizio cui desidera iscriversi
- Il modulo deve suggerire un'età di 18 anni ed il consenso all'invio di informazioni commerciali
- I servizi disponibili sono denominati "Servizio 1", "Servizio 2" e "Servizio 3"
- Il modulo deve suggerire la sottoscrizione al primo servizio
- Per ogni dato si determina il tipo di elemento da inserire nel modulo
  – nominativo ed età con elementi di tipo Text
  – sesso con un elementi di tipo Radio
  – infoComm con un elemento di tipo Checkbox
  – servizio con un elemento di tipo Select
- Si deve, inoltre, inserire il pulsante di invio del modulo
- Per gestire la presentazione si adopera una tabella HTML che presenta sulla prima colonna il nome del campo e nella seconda gli elementi corrispondenti

ESEMPIO MODULO ISCRIZIONE

```
<HTML>
<BODY>
    <FORM NAME="iscrizione" ACTION="" METHOD="POST">
        <TABLE>
            <!-- Nominativo -->
            <TR>
                <TD>Nominativo</TD>
                <TD>
                    <INPUT NAME="nominativo" TYPE="text" SIZE="40">
                </TD>
            </TR>
            <!-- Eta' -->
            <TR>
                <TD>Et&agrave;</TD>
                <TD>
                    <INPUT NAME="eta" TYPE="text" SIZE="3" VALUE="18">
                </TD>
            </TR>
            <!-- Sesso -->
            <TR>
                <TD>Sesso</TD>
                <TD>
                    <INPUT NAME="sesso" TYPE="radio" VALUE="M">M
                    <INPUT NAME="sesso" TYPE="radio" VALUE="F">F
                </TD>
            </TR>
```

ESEMPIO MODULO ISCRIZIONE

```
            <!-- Inform. commerciali -->
            <TR>
                <TD>Inform. commerciali</TD>
                <TD>
                    <INPUT NAME="infoComm" TYPE="checkbox" CHECKED>
                </TD>
            </TR>
            <!-- Servizio -->
            <TR>
                <TD>Servizio</TD>
                <TD>
                    <SELECT NAME="servizio">
<OPTION VALUE="s1" SELECTED>Servizio 1</OPTION>
<OPTION VALUE="s2">Servizio 2</OPTION>
<OPTION VALUE="s3">Servizio 3</OPTION>
</SELECT>
                </TD>
            </TR>
            <!-- Invio del modulo -->
            <TR>
                <TD>
                    <INPUT TYPE="submit" VALUE="Invia">
                </TD>
            </TR>
        </TABLE>
    </FORM>
</BODY>
</HTML>
```

ESEMPIO MODULO ISCRIZIONE

- • Per accedere al nominativo immesso `document.iscrizione.nominativo.value`
- Per accedere all'età `document.iscrizione.eta.value`
- Per accedere al valore numerico dell'età `parseInt(document.iscrizione.eta.value)`
- Per visualizzare un messaggio relativo alla scelta di riceve informazioni commerciali:

```
if(document.iscrizione.infoComm.checked)
    alert("Vuoi ricevere informazioni commerciali");
else
    alert("Non vuoi ricevere informazioni commerciali");
```

ESERCIZI

- Implementare una funzione in JS che verifichi se una parola inserita è palindroma
- Implementare una funzione in JS che verifichi se un anno è bisestile
- Creare una pagina html che chieda all'utente di selezionare un mese da una combobox e ne visualizzi il numero dei giorni del mese
- Scrivere una pagina web che chieda all'utente di inserire un numero in un campo di testo e stampi il numero in cifre romane
- Creare una calcolatrice in una pagina web
- Implementare il gioco del Tris

ESERCIZI

- Esercizio 1
  Scrivere un documento HTML contenente una form contenente i seguenti campi:
  cognome (casella di testo editabile lunga 40 caratteri)
  nome (casella di testo editabile lunga 30 caratteri)
  matricola (casella di testo editabile lunga 12 caratteri)
  regione di residenza (da scegliere da un menu che riporta le 20 regioni italiane) • email (casella di testo editabile lunga 30 caratteri)
  telefono (casella di testo editabile lunga 15 caratteri)
  richieste particolari (area di testo editabile di 12 righe per 60 colonne)
  bottone di invio e bottone di reset

- Aggiungere al documento HTML una funzione JavaScript che esegue i seguenti controlli:
  - verifica che il cognome non sia vuoto;
  - verifica che il nome non sia vuoto;
  - verifica che la matricola sia un numero;
  - verifica che sia stata selezionata una regione;
  - verifica che o l'email o il telefono siano non vuoti.

- Inoltre, fare in modo che, nel documento HTML, tale funzione JavaScript venga eseguita quando l'utente invia la form (senza far uso di HTML5)

Dott. Antonio Giovanni Lezzi

OGGETTI DI TIPO STRING

- Anche le stringhe hanno un corrispondente oggetto predefinito che prevede metodi per la loro manipolazione

- Tramite l'operatore new possiamo creare istanze di oggetti String come mostrato di seguito:

```
var x = new String("Una nuova stringa”);
```

- **replace():** sostituisce all'interno dell'oggetto la stringa passata come primo argomento con quella passata come secondo; come secondo argomento può essere passata anche una espressione regolare

```
var miaStringa = "Visita Google"
// effettuo una sostituzione semplice
alert(miaStringa.replace('Google','Yahoo'));
```

OGGETTI DI TIPO STRING

- **search()**: effettua una ricerca della stringa passata in argomento all'interno dell'oggetto e ne restituisce la posizione (restituisce -1 se non trovata); come argomento può essere passata anche una espressione regolare

```
var miaStringa = "Evviva JavaScript"
alert(miaStringa.search('JavaScript')); // restuirà: 7
```

- **slice()**: consente di tagliare la stringa specificando il carattere di partenza e quello finale (che sono compresi)

```
var miaStringa = "JavaScript";
alert(miaStringa.slice(3,5)); // Verrà visualizzato "vaS"
```

- **split()**: divide una stringa sulla base di un separatore e restituisce un array

```
var miaStringa = "divido la stringa in base agli spazi vuoti!";
alert(miaStringa.split(" "));
```

OGGETTI DI TIPO STRING

- **substr()**: consente di estrarre dall'oggetto una sottostringa specificando il carattere di partenza (che verrà incluso) ed il numero di caratteri da estrarre

```
var miaStringa = "JavaScript";
alert(miaStringa.substr(3,5)); // Verrà visualizzato "vaScr"
```

- **substring()**: consente di estrarre dall'oggetto una sottostringa specificando il carattere di partenza (che verrà incluso) ed il carattere finale (che non verrà incluso nella sottostringa)

```
var miaStringa = "JavaScript";
alert(miaStringa.substr(3,5)); // Verrà visualizzato "va"
```

TEMPLATE STRING

- Le template string permettono di risolvere i problemi di composizione di una stringa complessa con semplicità
- È una sequenza di caratteri delimitata da backtick (`) al posto di singoli o doppi apici. Ecco un esempio:
- var somma = `La somma di ${a} e ${b} è ${a+b}`;
- Come possiamo vedere, la definizione della stringa risultante risulta meno macchinosa e meno soggetta a errori rispetto alla concatenazione di stringhe
- Nota: Il layout delle tastiere italiane non prevede il backtick, per ottenerlo dovremo utilizzare la combinazione ALT + 96 e al suo interno usiamo l'interpolazione, cioè all'inserimento di espressioni definite tramite la sintassi: ${espressione} per effettuare la concatenazione di stringhe
- I ritorni a capo inseriti all'interno di una template string vengono mantenuti consentendo di creare stringhe multilinea senza ricorrere al carattere \n.

TAGGED TEMPLATE STRING

- Sintatticamente esse si presentano come una normale template string preceduta da un prefisso (tag) scelto dallo sviluppatore

```
var somma = maiuscolo`La somma di ${a} e ${b} è ${a+b}`;
```

- Il tag indicato sul template string corrisponde a una funzione, cui viene passato come primo parametro un array contenente le sottostringhe costanti e come parametri successivi le espressioni in essa contenute

```
function maiuscolo(strings, a, b, c) {
    return (`${strings[0]}${a}${strings[1]}${b}${strings[2]}${c}`).toUpperCase();
}
```

- Come possiamo notare, il primo parametro della funzione rappresenta un array di stringhe, ciascuna rappresentante la sottostringa della template string che separa due espressionistrings:Array[4]

```
0: "La somma di "
1: " e "
2: " è "
3: ""
```

ENCODING URL

- La possibilità di modificare l'output di una template string tramite un tag si presta a diverse applicazioni

```
function safeUrl(strings, protocol, params) {
    return `${protocol}${strings[1]}${encodeURIComponent(params)}`;
}

var protocol = "http";
var params = "a:b";

console.log(safeUrl`${protocol}://www.site.it/page?x=${params}`);
```

REGEXP

- JavaScript ha un supporto nativo per le espressioni regolari basato sull'oggetto RegExp
- Sono previsti due approcci per la creazione di un'espressione regolare: facendo riferimento all'oggetto RegExp o utilizzando una notazione letterale:

```
var x = new RegExp("abc");
var y = /abc/;
```

- Una espressione regolare è uno schema di stringa (pattern) composto da una sequenza di caratteri alfanumerici e di eventuali caratteri speciali e indica direttamente la stringa da ricercare all'interno di un'altra stringa
- **i:** esegue una ricerca ignorando la distinzione tra maiuscole e minuscole
- **g:** esegue una ricerca globale, cioè individua tutte le occorrenza di un pattern
- **m:** esegue una ricerca su stringhe multilinea

REGEXP

- Nel caso di utilizzo di un letterale i modificatori vengono specificati subito dopo il letterale stesso

- In caso di utilizzo dell'oggetto RegExp i modificatori vengono indicati come parametro aggiuntivo

```
var x = new RegExp("abc", “i");
var y = /abc/i;

var x = new RegExp("abc", “ig”);
var y = /abc/ig;
```

REGEXP: METACARATTERI

- I caratteri speciali consentono di creare pattern che individuano un insiemi di stringhe nelle stringhe da cercare, sono:
- parentesi quadre che permettono di specificare un insieme di caratteri alfanumerici, possiamo specificare un intervallo di caratteri indicano l'elemento iniziale e quello finale, come ad esempio [a-z] o [0-9].

```
var y = /[aeiou]bc/i; //indica l'insieme delle stringhe che iniziano per vocale e finiscono con bc
```

- I **metacaratteri** sono dei caratteri che ne indicano altri, per evitare che siano interpretati come i caratteri alfabetici si fa ricorso al carattere di escaping quindi \w e \d
- **.** : Il punto indica un qualsiasi carattere
- **\w** : indica un carattere alfanumerico
- **\d** : indica una cifra numerica

```
var y = /\d\dzz.\w/i; //stringhe con due cifre, zz,  un carattere qualsiasi e terminato con carattere
alfanumerici
```

REGEXP: QUANTIFICATORI

- I **quantificatori** sono caratteri che indicano quante volte un carattere può comparire in una stringa
- **+**: messo dopo un carattere o un metacarattere, indica che ci si aspetta l'esistenza di una o più occorrenze del carattere/metacarattere
- *****: indica l'esistenza di zero o più occorrenze
- **?**: indica l'esistenza di zero o una occorrenza
- **{n}**: indica l'esistenza di esattamente n occorrenze

```
var y = /[a-z]+\w*/i; //carattere alfabetico seguito da valori alfanumerici di lunghezza variabile
var codiceFiscale = /[a-z]{6}\d{2}[abcdehlmprst]\d{2}[a-z]\d{3}[a-z]/i;
```

REGEXP: TEST E EXEC

- Il metodo test() consente di verificare se una stringa individuata dall'espressione regolare è contenuta nella stringa passata come argomento e restituisce un valore boolenano

```
var y = /\d/;
y.test("str1nga");  //true
y.test("stringa");  //false
```

- Il metodo exec() restituisce un array con la sottostringa individuata o il valore null in caso di esito negativo

```
var y = /\d/;
y.exec("str1nga");  //["1"]
y.exec("stringa");  //null
```

REGEXP: SEARCH, REPLACE, …

- Il metodo search() dell'oggetto restituisce l'indice della prima occorrenza di una stringa individuata tramite l'espressione regolare passata come argomento:

```
var x = “str1nga".search(/\d/); //Il risultato sarà 3
```

- Nel caso in cui non venga trovata una corrispondenza il metodo search() restituisce il valore -1.

- Anche il metodo split(), che genera un array a partire da una stringa, usando espressioni regolari come input:

```
var x = "str1nga".split(/\d/);  //["str", "nga"]
```

- Il metodo replace() consente di sostituire una o più occorrenze di una stringa individuata da un'espressione regolare con un'altra stringa

```
var x = "str1ng4".replace(/\d/g, "numero"); //"strnumerongnumero"
```

- Infine il metodo match() consente di ottenere un array con le sottostringhe individuate da un'espressione regolare:

```
var x = "str1ng4".match(/\d/g); //["1", "4"]
```

REGEXP: ESEMPI

- indirizzo di posta elettronica: **/\w+@\w+\.\w{2,4}/i**
- indirizzo IP: **/^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/**
- elemento HTML : **/<div\b[^>]*>(.*?)<\/div>/i**
- valori esadecimali: **/[a-f0-9]+/i**
- URL: **/(ftp|http|https):\/\/(\w+:{0,1}\w*@)?(\S+)(:[0-9]+)?(\/|\/([\w#!:.?+=&%@!\-\/]))?/g**

ARRAY: OPERAZIONI

- Splice() viene usato per l'aggiungere o rimuovere elementi in qualsiasi posizione all'interno di un array, con i seguenti argomenti
- Il primo argomento indica la posizione dell'array da cui eliminare o aggiungere elementi; Obbligatorio
- il secondo argomento indica il numero di elementi da eliminare; Obbligatorio
- dal terzo argomento in poi è possibile specificare gli elementi da aggiungere. Lista di parametri opzionali
- In pratica splice() individua una posizione nell'array a partire dalla quale elimina il numero di elementi specificati come secondo argomento ed aggiunge gli eventuali elementi passati dal terzo argomento in poi.

```
var x = new Array("uno", "due", "tre");
var y = x.splice(1, 1, "quattro");
// x = ["uno", "quattro", "tre"]
// y = ["due"]
var z = x.splice(2, 0, "cinque", "sei", "sette");
// x = ["uno", "quattro", "cinque", "sei", "sette", "tre"]
// z = []
```

ARRAY: OPERAZIONI

- Tramite il metodo slice() abbiamo la possibilità di estrarre una porzione di array fornendo l'indice iniziale e quello finale:

```
var x = new Array("uno", "due", "tre", "quattro");
var y = x.slice(1, 3); // y = ["due", "tre"]
```

- Come possiamo notare, l'elemento corrispondente all'indice finale (3 nel nostro esempio) non viene incluso nell'array estratto.

ARRAY: OPERAZIONI

- Si possono concatenare due array usando concat

```
var x = new Array("uno", "due", "tre");
var y = new Array("quattro", "cinque");
var z = x.concat(y);    //["uno", "due", "tre", "quattro", "cinque"]
```

- L'ordinamento si effettua con metodo sort() e l'inversione dell'ordine degli elementi tramite reverse():

```
var x = new Array("uno", "due", "tre");
x.sort();        //["due", "tre", "uno"]
x.reverse();     //["uno", "tre", "due"]
```

- Per individuare l'indice di un elemento all'interno di un array tramite il metodo indexOf() che restituisce l'indice della prima occorrenza trovata. La scansione dell'array effettuata da indexOf() alla avviene partendo dal primo elemento, se l'elemento non viene trovato nell'array indexOf() restituisce -1.

```
var x = new Array("uno", "due", "tre");
var y = x.indexOf("due");                // 1
```

ARRAY: OPERAZIONI

- Il metodo join() che consente di ottenere una stringa a partire dagli elementi di un array, La stringa ottenuta non è altro che la concatenazione degli elementi dell'array separati dalla virgola

```
var x = new Array("uno", "due", "tre");
var y = x.join();        //"uno,due,tre"
```

- Possiamo indicare quale carattere o quale stringa utilizzare come separatore specificando un argomento, come mostrato di seguito:

```
var x = new Array("uno", "due", "tre");
var y = x.join(" e poi ");       //"uno e poi due e poi tre"
```

SET

- Un Set può contenere dati di qualsiasi tipo ma senza duplicati, le operazioni più comuni che vengono fatte sugli insiemi sono l'aggiunta o la rimozione di elementi e la verifica dell'esistenza di un elemento. Analizziamo il seguente esempio:

```
var mySet = new Set();
mySet.add(1);
mySet.add(2);
mySet.add("tre");
```

- la proprietà size, che indica il numero di elemento contenuti nella struttura dati

- il metodo has(), che indica se l'elemento passato come argomento è contenuto nell'insieme.

```
console.log(mySet.size);     //3
console.log(mySet.has(2));   //true
```

- Per eliminare un elemento da un Set utilizziamo il metodo delete() passandogli l'elemento da eliminare:

```
mySet.delete(1);
```

- Il metodo clear() elimina tutti gli elementi contenuti nella struttura dati.

MAP

- Le mappe associative consentono di abbinare un valore ad una chiave, gli oggetti JavaScript non sono altro che coppie di chiavi e valori che ne rappresentano le proprietà

- La mappa associativa originaria prevedeva che la chiave fosse una stringa, ora consente di creare associazioni tra chiavi e valori di qualsiasi tipo

```
var myMap = new Map();
myMap.set("nome", "Mario");
myMap.set(3.14, "Pi greco");

var myObj = {id: 123, data: "test"};
myMap.set(myObj, "Oggetto");
```

- Analogamente agli insiemi, le mappe prevedono il metodo has() per verificare se una chiave esiste e delete() per eliminare un'associazione

- La proprietà size consente di ottenere il numero degli elementi della mappa mentre get() consente di accedere al valore associato ad una chiave:

```
myMap.delete(3.14);

console.log(myMap.has(3.14));   //false
console.log(myMap.size);        //2
console.log(myMap.get("nome")); //'Mario'
console.log(myMap.get(myObj));  //'Oggetto'
```

WEAKMAP

- Si tratta di una mappa simile alla Map i cui elementi sono gestiti tramite un riferimento debole il quale richiede al garbage collector di eliminarli dalla memoria se non presenta più alcun riferimento
- Può essere utilizzata mettendo in corrispondenza elementi del DOM, cioè della rappresentazione ad oggetti della struttura di una pagina HTML, con dei valori. In seguito a manipolazioni del DOM un elemento viene eliminato, la corrispondente associazione nella WeakMap scompare di conseguenza.
- È da sottolineare che le WeakMap non ammettono chiavi di tipo semplice, come stringhe o numeri, ma soltanto oggetti.

OGGETTI

- Un oggetto è un contenitore di valori eterogenei, messi insieme a formare una struttura dati unica e tale da avere generalmente una particolare identità
- Un oggetto è utilizzato per rappresentare un'entità specifica come ad esempio una persona, un ordine, una fattura, una prenotazione, etc. tramite un'aggregazione di dati e di funzionalità.
- Un oggetto tipicamente possiede:
  - Dati, detti proprietà e rappresentati da coppie di nome-valore;
  - Funzionalità, sono dette metodi e rappresentate da funzioni.

OGGETTI

- Un oggetto è in fondo una lista di coppie di valori e come ogni lista può essere vuota
- Un modo per creare un oggetto è la rappresentazione letterale, rappresentiamo un oggetto racchiudendo tra parentesi graffe le sue proprietà e i suoi metodi.

```
var oggettoVuoto = {};
var persona = { "nome": "Mario", "cognome": "Rossi"};
```

- Nella prima dichiarazione è stato creato un oggetto vuoto
- Nella seconda dichiarazione, per definire l'oggetto persona abbiamo elencato due coppie di stringhe separate da virgole. Ciascuna coppia è formata da due stringhe separate da due punti. La prima stringa rappresenta il nome della proprietà mentre la seconda indica il suo valore

OGGETTI: PROPRIETÀ

- I nomi delle proprietà di un oggetto non hanno le restrizioni dei nomi delle variabili

- I doppi apici intorno al nome della proprietà sono opzionali, ma diventano obbligatori quando il nome non segue le regole per nomi delle variabili

```
var persona = { nome: "Mario", cognome: "Rossi"};
```

- per una definizione come quella che segue siamo obbligati ad usare i doppi apici:

```
var persona = { "primo-nome": "Mario", "secondo.nome": "Rossi"};
```

- Una proprietà può assumere qualsiasi valore derivante da un'espressione JavaScript, compreso un altro oggetto, quindi possiamo creare oggetti annidati come mostrato di seguito:

```
var persona = {
    nome: "Mario",
    cognome: "Rossi",
    indirizzo: {
        via: "Via Garibaldi",
        numero: 15,
        CAP: "00100",
        citta: "Roma"
    }
};
```

OGGETTI: PROPRIETÀ

- Per accedere alle proprietà di un oggetto possiamo usare la dot-notation in base alla quale indichiamo un oggetto e la proprietà a cui siamo interessati separandoli con un punto

```
var nome = persona.nome;
```

- L'approccio alternativo è quello in cui specifichiamo la proprietà di un oggetto indicando il suo nome sotto forma di stringa tra parentesi quadre:

```
var nome = persona["nome"];
```

- Questo approccio è obbligatorio quando il nome della proprietà non segue le regole per i nomi delle variabili.

- Il tentativo di accedere ad una proprietà non definita in un oggetto non genera un errore, ma restituisce il valore undefined

```
var eta = persona.eta;
```

OGGETTI: PROPRIETÀ

- Gli oggetti in JavaScript siano entità dinamiche, nel senso che la loro struttura è molto flessibile e può essere modificata dinamicamente anche durante l'esecuzione di uno script, consente la creazione di un oggetto basato su una sorta di definizione incrementale.
- Se proviamo ad assegnare un valore ad una proprietà non definita creiamo di fatto questa proprietà inizializzandola con il valore assegnato
- In pratica, invece di definire con un unica rappresentazione letterale il nostro oggetto, possiamo partire da una rappresentazione di base ed arricchirla via via con le proprietà che vogliamo

```
var persona = {};

persona.nome = "Mario";
persona.cognome = "Rossi";
persona.indirizzo = {
    via: "Via Garibaldi",
    numero: 15,
    CAP: "00100",
    citta: "Roma"
};

persona.eta = 32;
```

OGGETTI: PROPRIETÀ

- È prevista anche la possibilità di eliminare una proprietà tramite l'istruzione delete:

```
delete persona.cognome;
```

- Dopo l'eliminazione della proprietà cognome, ogni tentativo di accesso ad essa restituirà il valore undefined.

- Possiamo anche assegnare ad un oggetto una proprietà con un nome che non segue le regole sintattiche per variabili e identificatori, come nel seguente esempio:

```
persona["data di nascita"] = new Date(1981, 11, 19);
```

OGGETTI: METODI

- I metodi rappresentano attività che un oggetto può compiere

```
function visualizzaNomeCognome() { return "Mario Rossi"; }
persona.nomeCognome = visualizzaNomeCognome;
```

- Abbiamo definito una funzione che si limita a restituire una stringa ed abbiamo assegnato il nome della funzione ad una nuova proprietà dell'oggetto persona. Da notare che non assegniamo alla proprietà il risultato della chiamata alla funzione, ma la funzione stessa tramite il suo nome.

- La proprietà nomeCognome, dal momento che contiene una funzione, è di fatto un metodo. Per eseguirlo dobbiamo far riferimento ad esso specificando le parentesi tonde come in una chiamata di funzione:

```
var nomeCognome = persona.nomeCognome();
```

OGGETTI: METODI

- Naturalmente un metodo può prevedere degli argomenti come avviene per una normale funzione.
- Un modo alternativo per definire un metodo è mostrato di seguito:

```
persona.nomeCognome = function () { return "Mario Rossi"; }
```

- In questo caso definiamo la funzione direttamente nell'assegnamento alla proprietà. Da notare come in questo caso non abbiamo specificato un nome per la funzione
- Indipendentemente da come è stato definito il metodo, la chiamata avviene nello stesso modo.

OGGETTI: THIS

- La parola chiave this rappresenta l'oggetto a cui appartiene il metodo invocato. Quindi possiamo riscrivere il nostro metodo nel seguente modo:

```
persona.nomeCognome = function () {
    return this.nome + " " + this.cognome;
}
```

- Questo ci garantisce la visualizzazione corretta dei dati correnti al momento della chiamata.

OGGETTI: ALIAS

- Gli oggetti sono strutture complesse e l'assegnamento di un oggetto ad una variabile non fa in modo che la variabile contenga effettivamente l'oggetto. Essa contiene in realtà un riferimento all'oggetto.

```
var persona = {nome: "Mario", cognome: "Rossi"};
var altraPersona = persona;

altraPersona.nome = "Giuseppe";
```

- In questo caso se proviamo ad accedere alla proprietà *nome* dell'oggetto *persona* scopriremo che anch'essa ha ora il valore Giuseppe

- In realtà entrambe le variabili *persona* e *altraPersona* contengono un riferimento allo stesso oggetto e pertanto ogni modifica effettuata alla variabile persona si riflette anche sulla variabile altraPersona

OGGETTI: PARAMETRI

- Questo è un aspetto molto importante da tenere presente quando si lavora con gli oggetti, in particolar modo nel passaggio di valori agli argomenti delle funzioni.
- Il meccanismo del passaggio di valori alle funzioni adotta lo stesso approccio dell'assegnamento alle variabili
- Il passaggio di valori relativi a tipi di di tipo numero avviene sempre per valore mentre il passaggio di oggetti avviene sempre per riferimento.

OGGETTI

- Immaginiamo però di aver bisogno di più oggetti dello stesso tipo, ad esempio di più oggetti persona, che condividono la stessa struttura:

```
var persona = {
    nome: "Mario",
    cognome: "Rossi",
    indirizzo: "Via Garibaldi, 50 - Roma",
    email: “mario.rossi@mail.it",
    mostraNomeCompleto: function() { ... },
    calcolaCodiceFiscale: function() { ... }
}
```

- Utilizzando la notazione letterale saremmo costretti a ripetere la definizione per ciascun oggetto che vogliamo creare, cosa indispensabile per gli elementi identificativi della persona ma inutilmente ripetitiva per le parti costanti come ad esempio i metodi. In altre parole, ricorrendo alla notazione letterale nella definizione degli oggetti otteniamo un risultato non riutilizzabile.

COSTRUTTORE

- Per evitare quindi di dover ridefinire da zero oggetti che hanno la stessa struttura possiamo ricorrere ad un costruttore

- Un costruttore non è altro che una normale funzione JavaScript invocata mediante l'operatore new

```
function persona() {
    this.nome = "";
    this.cognome = "";
    this.indirizzo = "";
    this.email = "";
    this.mostraNomeCompleto = function() {...};
    this.calcolaCodiceFiscale = function() {...};
}
```

COSTRUTTORE

- Questa funzione definisce le proprietà del nostro oggetto assegnandole a se stessa (this) in qualità di oggetto ed impostando i valori predefiniti. Per creare un oggetto di tipo persona dovremo a questo punto invocare la funzione premettendo l'operatore new:

```
var marioRossi = new persona();
marioRossi.nome = "Mario";
marioRossi.cognome = "Rossi";
var giuseppeVerdi = new persona();
giuseppeVerdi.nome = "Giuseppe";
giuseppeVerdi.cognome = "Verdi";
```

- In questo modo nella creazione di più oggetti con la stessa struttura ci limiteremo ad impostare i soli valori specifici che differenziano un oggetto dall'altro.

COSTRUTTORE

- Nella definizione di un costruttore possiamo prevedere la presenza di parametri che possiamo utilizzare nell'inizializzazione del nostro oggetto. Consideriamo ad esempio la seguente definizione del costruttore dell'oggetto persona:

```
function persona(nome, cognome) {
    this.nome = nome;
    this.cognome = cognome;
    this.indirizzo = "";
    this.email = "";
    this.mostraNomeCompleto = function() {...};
    this.calcolaCodiceFiscale = function() {...};
}
```

COSTRUTTORE

- Esso ci consente di creare ed inizializzare un oggetto specificando i valori nella chiamata al costruttore:

```
var marioRossi = new persona("Mario", "Rossi");
var giuseppeVerdi = new persona("Giuseppe", "Verdi");
```

- È fondamentale utilizzare l'operatore new nella creazione di un oggetto tramite costruttore. Infatti se lo omettiamo, magari per dimenticanza, quello che otterremo non sarà la creazione di un oggetto ma l'esecuzione della funzione, con risultati imprevedibili. Ad esempio, se tentiamo di creare un oggetto persona omettendo l'operatore new:

```
var marioRossi = persona();
```

- il valore della variabile marioRossi sarà undefined, dal momento che la funzione persona() non restituisce alcun valore. Inoltre, tutte le proprietà e i metodi definiti all'interno del corpo della funzione saranno assegnati all'oggetto this valido nel contesto di esecuzione della funzione, come può essere ad esempio l'oggetto globale. Questa assegnazione potrebbe ridefinire il valore di variabili omonime con effetti collaterali difficilmente prevedibili.

COSTRUTTORE

- Per cercare di ovviare a dimenticanze accidentali dell'operatore new si può ricorrere ad una convenzione abbastanza diffusa che prevede l'uso del maiuscolo per la prima lettera del nome di un costruttore. Nel nostro caso, ad esempio, dovremmo definire il costruttore nel seguente modo:

```
function Persona() {
  ...
}
```

- Naturalmente si tratta di una semplice convenzione per aiutarci a distinguere un costruttore da una normale funzione, senza alcuna garanzia di successo.

STRICT MODE

- Possiamo affiancare a questa convenzione l'uso dello strict mode, come mostrato di seguito:

```
function Persona() {
'use strict'
...
}
```

- In questa modalità, l'oggetto this risulta undefined durante l'esecuzione del codice in seguito ad una chiamata alla funzione. Questo genera un errore a runtime per il tentativo di accedere alla proprietà di un oggetto non esistente, evitando quindi invocazioni non volute del costruttore

- Purtroppo questo approccio non è sufficiente nel caso in cui il costruttore rappresenti il metodo di un oggetto:

```
var gente = {
    ...
    persona: function() {
        'use strict'
        ...
    }
};

var marioRossi = gente.persona();
```

- In questo caso this viene associato all'oggetto gente e quindi non avremmo un errore che ci avvisi dell'errato utilizzo del costruttore.

OGGETTI

- L'oggetto Object è anche in grado di generare istanze di oggetti a partire da una qualunque espressione

```
var numero      = new Object(12);
var altroNumero = new Object(3*2);
var stringa     = new Object("test");
var persona     = new Object({nome: "Mario", cognome: "Rossi"});
```

- A parte l'ultima istruzione che è equivalente alla creazione di un oggetto tramite la sua rappresentazione letterale

- Per ogni tipo di dato primitivo, tranne null e undefined, esiste un corrispondente oggetto specializzato che mette a disposizione del programmatore specifiche proprietà e metodi.

OGGETTI: METODI CONDIVISI

- Dal momento che tutti gli oggetti JavaScript sono basati su Object, essi condividono alcuni metodi: toString() e valueOf().

- Il primo metodo restituisce una versione in stringa dell'oggetto, come mostrato nel seguente esempio:

```
var x = new Object(32);
x.toString();            //restituisce "32"
```

- Nel caso di oggetto non riconducibile ad un tipo di dato primitivo sarà restituita la stringa [object Object]:

```
var persona = new Object({nome: "Mario", cognome: "Rossi"});
persona.toString();     //restituisce "[object Object]"
```

- Il metodo valueOf() restituisce il corrispondente valore del tipo di dato primitivo associato all'oggetto:

```
var x = new Object(32);
x.valueOf();             //restituisce 32
```

- Nel caso di oggetto non associato ad un tipo di dato primitivo viene restituito l'oggetto stesso.

- È da sottolineare come JavaScript chiami implicitamente questi metodi quando è necessario effettuare delle conversioni o quando in un'espressione è richiesto il valore primitivo dell'oggetto.

FUNZIONI ANONIME

- Una funzione nella maggior parte dei linguaggi di programmazione è intesa come un meccanismo per identificare un gruppo di istruzioni da eseguire su richiesta, eventualmente con un passaggio di parametri

- Anche se a prima vista le funzioni JavaScript sembrano del tutto analoghe alle funzioni supportate da altri linguaggi di programmazione come C, Java, C#, in realtà in JavaScript è qualcosa in più: un oggetto, oggetto di prima classe (first class object), cioè un'entità che può essere dinamicamente creata, distrutta, passata ad una funzione, assegnata ad una variabile, restituita come valore.

- A differenza degli altri oggetti JavaScript basati, come abbiamo visto, su Object, una funzione ha come oggetto base Function. In quanto oggetto, pertanto, una funzione ha proprietà e metodi e può essere utilizzata come un qualsiasi altro oggetto.

- Ad esempio, possiamo assegnare una funzione ad un nome come una variabile, come mostra il seguente esempio:

```
var somma = function(x, y) {
    return x + y;
};
```

- Come possiamo vedere dall'esempio, abbiamo utilizzato la parola chiave function per creare la funzione da assegnare alla variabile somma, proprio come una normale definizione di funzione. A differenza della definizione classica, però, in questo caso non abbiamo specificato un nome per la funzione.

FUNZIONI ANONIME

- Normalmente, nella definizione classica di una funzione specifichiamo un nome per poter accedere ad essa in un'altra parte di uno script.
- Una definizione di funzione che non specifica il nome è detta in genere funzione anonima o funzione letterale o funzione lambda
- Nel caso in cui una funzione viene assegnata ad una variabile, invece, l'utilità del nome viene meno, dal momento che possiamo accedere ad essa tramite la variabile.
- Tuttavia, nessuno ci vieta di assegnare un nome ad una funzione anche quando questa viene assegnata ad una variabile.

```
var z = somma(4, 8);
console.log(z); // 12
```

- Possiamo quindi vedere le parentesi tonde come una sorta di operatore per l'invocazione ed il passaggio dei parametri applicabile a variabili che fanno riferimento ad una funzione.

```
(function(x, y) {
    return x + y;

}(4,8));
```

FUNZIONI ANONIME

- Da notare che l'intera espressione è racchiusa tra parentesi tonde, questo è necessario perché il parser JavaScript, trovando la parola chiave function, tenterebbe di interpretare l'espressione come una definizione di funzione nel contesto globale.
- Questo tipo di espressione è in genere chiamata **immediately-invoked function expression (IIFE)** oppure **self-executing anonymous function** ed offre una significativa capacità espressiva quando è utilizzata insieme ad altre caratteristiche delle funzioni JavaScript
- Una funzione, essendo un oggetto, prevede proprietà e metodi, tra le proprietà di una funzione abbiamo ad esempio:
    - name, che indica il nome assegnatole;
    - length, che indica il numero di argomenti previsti.

```
var somma = function Sum(x, y) {
    return x + y;
};

console.log(somma.name);    // stampa "Sum"
console.log(somma.length);  // stampa 2
```

- Da notare che nel caso di funzione anonima la proprietà name ha come valore una stringa vuota

CALLBACK

- Poiché le funzioni JavaScript sono oggetti di prima classe, possono essere passate come parametri di un'altra funzione, la funzione passata come parametro è detta generalmente funzione di callback

```
function calcola(func, arg1, arg2) {
    return func(arg1, arg2);
}
```

- La funzione calcola() prevede tre argomenti. Il primo argomento è una funzione di callback che viene invocata sul secondo e terzo argomento

- In questo caso il valore restituito dalla funzione è il valore restituito dall'applicazione della funzione di callback sugli altri due argomenti.

- Possiamo sfruttare la funzione calcola() per eseguire ad esempio la funzione somma() definita negli esempi precedenti:

```
console.log(calcola(somma, 13, 25));    // 38
```

CALLBACK

- Un aspetto a cui prestare attenzione quando scriviamo funzioni che accettano callback è quello di accertarsi che venga passata effettivamente una funzione prima di invocarla. Il seguente esempio mostra come effettuare tale verifica:

```
function calcola(func, arg1, arg2) {
    if (func && typeof func === "function") {
        return func(arg1, arg2);
    }
}
```

- La condizione dell'if verifica che l'argomento func sia stato effettivamente passato, cioè che non sia null, e che il suo tipo sia function. Solo se la condizione è vera potrà essere applicata la funzione passata agli altri argomenti.

- L'uso di funzioni di callback è molto frequente nella programmazione JavaScript

- Ad esse si ricorre ad esempio, nell'esecuzione di alcuni metodi e funzioni predefinite, nelle azioni asincrone come nelle chiamate HTTP o nella gestione di eventi

CALLBACK

- Consideriamo ad esempio il metodo forEach() degli array:

```
var numeri = [11, 3, 24];
numeri.forEach(function(valore, indice) {
    console.log(valore);
});
```

- La funzione passata come parametro al metodo forEach() viene eseguita per ciascun elemento dell'array, visualizzando nel nostro caso l'elenco degli elementi dell'array

- In questo esempio abbiamo passato una funzione anonima direttamente come parametro di forEach() invece che una variabile a cui è assegnata una funzione

- Un'alternativa è la definizione di una classica funzione con nome ed il passaggio del nome della funzione come callback:

```
function mostraValore(valore, indice) {
    console.log(valore);
}

var numeri = [11, 3, 24];
numeri.forEach(mostraValore);
```

CALLBACK

- Il metodo sort() consente di ordinare gli elementi di un array di tipo primitivo, cioè numeri e stringhe.

- Per ordinare ad esempio un array di oggetti dobbiamo ricorrere ad un funzione di callback da passare al metodo sort() che verrà internamente utilizzata per stabilire il criterio di confronto tra gli elementi dell'array

```
var persone = [{nome: "Mario",    cognome: "Rossi", professione: "impiegato"},
               {nome: "Giuseppe", cognome: "Verdi", professione: "operaio"},
               {nome: "Marco",    cognome: "Neri",  professione: "insegnante"},];

persone.sort(function(a, b) {
    if (a.cognome < b.cognome) return -1;
    if (a.cognome > b.cognome) return 1;
    return 0;
});
```

- Come possiamo vedere, ordiniamo un array di oggetti in base al valore della proprietà cognome

- La funzione di callback deve restituire il valore 1 se il primo parametro viene prima del secondo nell'ordinamento, -1 se accade il contrario e 0 se è uguale.

CALLBACK

- Dal momento che una funzione è un oggetto è possibile restituirla come valore di ritorno dell'esecuzione di un'altra funzione

```
var incrementatore = function(incremento) {
    return function(valore) {
        return incremento + valore;
    };
};
```

- Alla variabile incrementatore viene assegnata una funzione che restituisce un'altra funzione. La funzione restituita incrementa il suo argomento con l'argomento della funzione che la restituisce.

- In altre parole, la funzione incrementatore genera una funzione che incrementa un valore numerico di un numero predefinito

- Ad esempio, se vogliamo creare una funzione che incrementa di cinque il valore passato come parametro possiamo farlo in questo modo:

```
var incrementaDiCinque = incrementatore(5);
```

- A questo punto possiamo utilizzare la funzione incrementaDiCinque() per aumentare di cinque unità il valore passato come parametro:

```
console.log(incrementaDiCinque(4));  // 9
console.log(incrementaDiCinque(16)); // 21
```

CALLBACK

- L'interpretazione del codice JavaScript da parte di un engine avviene in uno specifico contesto di esecuzione, cioè in un ambiente in cui vengono risolti i riferimenti alle variabili
- Un contesto di esecuzione è composto dall'insieme delle variabili accessibili in un dato momento dell'esecuzione di un'istruzione, compresi gli eventuali argomenti di una funzione e l'oggetto this.
- Si tratta di un concetto estremamente importante in JavaScript ed è opportuno conoscerlo bene per comprendere alcuni meccanismi comunemente utilizzati ed evitare comportamenti indesiderati.
- Un classico esempio è la corretta risoluzione dell'oggetto this, che rappresenta l'oggetto a cui è associata una proprietà o un metodo.
- Consideriamo la definizione del seguente oggetto:

```
var persona = {
    nome: "Mario",
    cognome: "Rossi",

    nomeCognome: function () {
        return this.nome + " " + this.cognome;
    }
};
```

CALLBACK

- Il metodo nomeCognome() restituisce semplicemente la concatenazione delle due proprietà nome e cognome sfruttando l'oggetto this.

- Creiamo ora una funzione che prende per parametro il riferimento ad un altra funzione e tenta di lanciarla

```
// saluta() prende come parametro una funzione (nomepersona)
// e la esegue nel proprio contesto di esecuzione.
function saluta(nomePersona) {
    console.log("Buongiorno " + nomePersona());
}
```

- Facciamo in questo modo per invocare il metodo nomeCognome in un diverso contesto di esecuzione, dove this cambia di significato. Quindi invochiamo la funzione saluta nel seguente modo:

```
// usiamo come parametro il metodo persona.nomeCognome
saluta(persona.nomeCognome);
```

- In altre parole chiamando saluta(persona.nomeCognome) non invochiamo il metodo nomeCognome() ma lo passiamo alla funzione saluta(), all'interno della quale verrà poi invocato (console.log("Buongiorno " + nomePersona()))

CALLBACK

- Dal momento che nomeCognome() è invocato al di fuori del suo contesto di esecuzione, this non rappresenta più l'oggetto persona ma l'oggetto globale, nel quale non è presente alcuna definizione di variabili nome e cognome.
- Il risultato che avremo è quindi:

```
Buongiorno undefined undefined
```

- invece che "Buongiorno Mario Rossi".
- Il problema nasce dal fatto che la funzione di callback nomePersona() viene eseguita nel contesto di esecuzione della funzione saluta()
- In questo contesto di esecuzione, l'oggetto this non rappresenta l'oggetto persona ma, nel caso specifico, indica l'oggetto globale, che ad esempio in un browser corrisponde alla finestra corrente (window).
- Come facciamo quindi ad ottenere il nome e cognome dell'oggetto persona?

CONTESTO DI ESECUZIONE IN CHIAVE DOM

- Riprendiamo un oggetto persona fatto in questo modo:

```
var persona = {
    nome: "Mario",
    cognome: "Rossi",
    saluta: function() {
        alert("Buongiorno " + this.nome + " " + this.cognome);
    }
};
```

- e associamo all'evento "click" su un bottone il metodo saluta:

```
document.getElementById("pulsante").addEventListener("click", persona.saluta);
```

- La situazione è analoga quella vista in precedenza: poiché quando viene invocato il metodo persona.saluta il contesto di esecuzione sarà quello del singolo bottone, l'oggetto this sarà diverso e otterremo il messaggio "Buongiorno undefined undefined" invece di quello che ci aspetteremmo.

CLASSI IN ECMA 6

- Una classe in questo modello sintattico è un modo alternativo per la definizione di un costruttore. Possiamo ad esempio definire il costruttore dell'oggetto persona nel seguente modo:

```
class persona {
    constructor(nome, cognome) {
        this.nome = nome;
        this.cognome = cognome;
        this.email = "";
        this.indirizzo = "";
    }
    mostraNomeCompleto() {
        return this.nome + " " + this.cognome;
    }
}
```

- All'interno della classe definiamo il nostro costruttore tramite il nome di funzione riservato constructor e i metodi del nostro oggetto.

CLASSI IN ECMA 6

- Nonostante l'utilizzo della parola chiave class è da tener presente che quello che stiamo definendo non è in realtà una vera classe, ma un costruttore semanticamente equivalente a quello che abbiamo creato in precedenza tramite la definizione di una function. L'introduzione del costrutto class è solo una semplificazione sintattica, ma semanticamente JavaScript rimane un linguaggio senza classi.

- Possiamo creare un nuovo oggetto a partire dalla classe appena definita tramite l'operatore new, in modo del tutto identico a come abbiamo fatto con il costruttore definito tramite funzione:

```
var marioRossi = new persona("Mario", "Rossi");
```

GETTER E SETTER

```
class persona {
    constructor(nome, cognome) {
        this.nome = nome;
        this.cognome = cognome;
        this._email = "";
        this.indirizzo = "";
    }

    mostraNomeCompleto() {
        return this.nome + " " + this.cognome;
    }

    get email() { return this._email; }
    set email(value) {
        var emailRegExp = /\w+@\w+\.\w{2,4}/i;
        if (emailRegExp.test(value)) {
            this._email = value;
        } else {
            console.log("Email non valida!");
        }
    }
}
```

SOTTOCLASSI E EREDITARIETÀ

- Possiamo creare sottoclassi estendendo una classe nel seguente modo:

```
class programmatore extends persona {
    constructor(nome, cognome) {
        super(nome, cognome);
        this.linguaggiConosciuti = [];
    }
}
```

- La parola chiave extends consente di dichiarare che la nuova classe deriva dalla classe superiore
- All'interno del costruttore della classe possiamo fare riferimento alla superclasse o classe base proprio tramite la parola chiave super.
- L'introduzione delle classi in JavaScript semplifica la scrittura del codice e lo rende più accessibile a chi è abituato a lavorare ad oggetti tradizionale ma JavaScript non ha classi e che quello che definiamo con la parola chiave class non è altro che un costruttore di oggetti.

HASOWNPROPERTY

- Un oggetto può avere proprietà o metodi derivati da un prototipo, potremmo essere interessati a fare una distinzione tra quelli che sono i membri specifici dell'oggetto in questione e quelli che derivano invece da uno degli oggetti della propria catena di prototipi.

- Ci viene incontro in questo caso il metodo hasOwnProperty() che restituisce i nomi delle sole proprietà non derivate da un prototipo

```
for (p in persona) {
    if (typeof persona[p] != "function" && persona.hasOwnProperty(p))
        console.log(p);
}
```

- Possiamo analizzare la relazione tra un oggetto ed il suo prototipo sfruttando l'operatore instanceof:

```
marioRossi instanceof persona
```

- Questa espressione restituisce true se l'oggetto marioRossi è un'istanza del costruttore persona(), false altrimenti.

- Infine, il metodo Object.getPrototypeOf() ci consente di ottenere il prototipo di un oggetto:

```
var prototipo = Object.getPrototypeOf(marioRossi);
```

GEOLOCALIZZAZIONE

- Le Geolocation API in realtà non fanno parte della specifica HTML 5, anche se spesso sono associate a quest'ultima (persino articoli di stampa specialistica sbagliano). Gli strumenti di relativi alla geolocalizzazione sono oggetto di standardizzazione da parte del Geolocation Working Group.
- Questa tecnologia è stata sviluppata in concomitanza con HTML 5 ed è anch'essa una componente innovativa della programmazione web
- Molte applicazioni in rete hanno già cominciato a farne uso, soprattutto nel campo mobile.

GEOLOCALIZZAZIONE

- I principali metodi associati alla geoloacalizzazione sono getCurrentPosition e watchPosition, entrambi utili allo stesso scopo: ottenere la posizione corrente
- getCurrentPosition: rileva la posizione un'unica volta
- watchPosition, dopo la prima rilevazione, nel caso in cui la posizione abbia subito una variazione significativa, trasmette una notifica contenente le nuove informazioni. Inoltre watchPosition restituisce un identificativo numerico univoco che servirà a localizzare i successivi spostamenti del dispositivo
- Per terminare la lettura ripetuta delle coordinate è necessario invocare il metodo clearWatch che accetta come argomento tale identificativo. Entrambe le funzioni accettano tre argomenti: successCallback, errorCallback e options.

GEOLOCALIZZAZIONE: SUCCESSCALLBACK

- Funzione da eseguire nel caso in cui il rilevamento della posizione è andato a buon fine. Alla funzione è passato come argomento un riferimento all'oggetto Position. Quest'ultimo ha due proprietà: la data e l'ora del rilevamento della posizione contenute all'interno della proprietà timestamp e l'oggetto Coords che permette l'accesso alle seguenti proprietà:
    - **longitude**: informazioni di longitudine.
    - **latitude**: informazioni di latitudine.
    - **altitude**: informazioni relative all'altitudine. Se il dispositivo su cui è installato il browser non riesce a rilevare l'altitudine, il parametro sarà null.
    - **accuracy**: indica un valore in metri che rappresenta l'approssimazione dei valori di longitudine e latitudine.
    - **altitudeAccuracy**: indica un valore in metri che rappresenta l'approssimazione del valore di altitudine. Se il valore di altitude è null allora lo sarà anche quello di altitudeAccuracy.
    - **heading**: indica la direzione espressa in gradi. Il valore corrisponde a una rotazione oraria rispetto al nord. Se il dispositivo è stazionario, il valore sarà NaN.
    - **speed**: velocità espressa in metri al secondo. Se la velocità è zero heading sarà NaN.

GEOLOCALIZZAZIONE: ERRORCALLBACK

- Parametro facoltativo che rappresenta la funzione da eseguire nel caso in cui il rilevamento della posizione sia fallito. Alla funzione è passato come argomento un riferimento all'oggetto PositionError contenente le due proprietà message e code utili a comprendere la natura dell'erroree può assumere i seguenti valori interi:
  - **PERMISSION_DEINED**, l'utente non ha dato il consenso relativo alla rilevazione della posizione.
  - **POSITION_UNAVAILABLE**, Il tracciamento è fallito a causa di un errore relativo al fornitore dei dati.
  - **TIMEOUT**, il tempo prestabilito entro il quale il tracciamento avrebbe dovuto avere luogo è stato superato.

GEOLOCALIZZAZIONE: OPTIONS

- è un oggetto di tipo PositionOptions le cui tre proprietà enableHighAccuracy, timeout e maximumAge permettono di intervenire nel modo in cui avviene il tracciamento:
  - **enableHighAccuracy**: è una proprietà booleana che se impostata a true viene richiesto di determinare la posizione nel modo più accurato possibile. Le conseguenze però sono un maggior tempo di elaborazione e un aumento del consumo energetico.
  - **timeout**: è la soglia di tempo prefissata entro la quale si desidera una risposta. Se la soglia viene superata l'esecuzione fallisce. Il valore è espresso in millisecondi e nel caso in cui non venga impostata una soglia il valore è pari a zero.
  - **maximumAge**: è un intervallo di tempo espresso in millisecondi che, nel caso di acquisizioni della posizione ripetute, stabilisce quanto tempo deve trascorrere per l'aggiornamento dei valori.

REVERSE GEOCODING

- Con il termine Reverse geocoding s'intende una metodologia che permette, tramite le informazioni di latitudine e longitudine, di ricavare altre informazioni semplici da comprendere come il nome di una via o di una località

- Nel web un modo semplice per farlo potrebbe essere quello di sfruttare le potenzialità di Google Maps. Un esempio si trova alla fine di questa sezione.

GEOLOCALIZZAZIONE: PRIVACY

- Parlando di geolocalizzazione sicuramente il tema della privacy è un argomento da non sottovalutare
- Fortunatamente se un sito web ospita tale tecnologia, l'utilizzo è consentito solo col previo consenso dell'utente.
- Inoltre, dopo aver dato il consenso ad un determinato sito web, il browser deve permettere l'annullamento del consenso.

GEOLOCALIZZAZIONE

```
<!DOCTYPE html>
<html>
<body>
    <button onclick="getLocation()">Localizzami</button>
    <p id="demo"></p>
    <p id="demoUpdate"></p>
    <div id="mapholder"></div>
    <script>
        var x = document.getElementById("demo");
        var xUpdate = document.getElementById("demoUpdate");

        function getLocation() {
            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(showPosition, showError);
                navigator.geolocation.watchPosition(showPositionUpdate);
            } else {
                x.innerHTML = "Geolocation is not supported by this browser.";
            }
        }

        function showPosition(position) {
            x.innerHTML = "Latitude: " + position.coords.latitude +
                "<br>Longitude: " + position.coords.longitude;

            var latlon = position.coords.latitude + "," + position.coords.longitude;

            var img_url = "https://maps.googleapis.com/maps/api/staticmap?center=" +
                latlon + "&zoom=14&size=400x300&sensor=false&key=AIzaSyBu-916DdpKAjTmJNIgngS6HL_kDIKU0aU";
            document.getElementById("mapholder").innerHTML = "<img src='" + img_url + "'>";
        }
```

GEOLOCALIZZAZIONE

```
        function showPositionUpdate(position) {
            xUpdate.innerHTML = "Latitude: " + position.coords.latitude +
                "<br>Longitude: " + position.coords.longitude;
        }

        function showError(error) {
            switch (error.code) {
                case error.PERMISSION_DENIED:
                    x.innerHTML = "User denied the request for Geolocation."
                    break;
                case error.POSITION_UNAVAILABLE:
                    x.innerHTML = "Location information is unavailable."
                    break;
                case error.TIMEOUT:
                    x.innerHTML = "The request to get user location timed out."
                    break;
                case error.UNKNOWN_ERROR:
                    x.innerHTML = "An unknown error occurred."
                    break;
            }
        }
    </script>
</body>
</html>
```

WEB STORAGE

- Le WebStorage API permettono il salvataggio locale d'informazioni concernenti la navigazione web
- Nascono per risolvere due principali problematiche relative ai cookie: Lo spazio per la memorizzazione, che passa dai restrittivi 4KB a 5MB, e l'impossibilità di avere due sessioni distinte in un browser relative allo stesso dominio
- Prima del Web Storage di HTML 5 ci furono altri tentativi per la risoluzione di queste due problematiche senza però riscontrare successo perché basate su tecnologie proprietarie (come userData di Microsoft) o perché necessitavano l'installazione di componenti aggiuntivi (flash script).
- Fortunatamente tutti i browser supportano questo nuovo membro della specifica, di conseguenza molto probabilmente, in poco tempo, il Web Storage prenderà il sopravvento sui cookie.

WEB STORAGE: LOCAL

- localStorage: Mantiene il comportamento dei cookie essendo comune a tutte le finestre del browser che condividono lo stesso dominio
- I dati persistono anche dopo la chiusura del browser. Permette ad esempio di aggiungere articoli nel carrello e di poter completare l'acquisto nei giorni successivi
- L'utilizzo ideale quindi riguarda applicazioni in cui un processo può essere interrotto per poi essere ripreso anche a distanza di giorni.
- Entrambi sono inoltre stati studiati per ospitare molti più dati, fino a 5MB, e a differenza dei cookie i dati non sono scambiati sistematicamente tra client e server, generando richieste meno corpose.

WEB STORAGE: LOCAL

```html
<!DOCTYPE html>
<html>
<head>
    <script>
        function clickCounter() {
            if (typeof(Storage) !== "undefined") {
                if (localStorage.clickcount) {
                    localStorage.clickcount = Number(localStorage.clickcount) + 1;
                } else {
                    localStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Contatore: " + localStorage.clickcount;
            } else {
                document.getElementById("result").innerHTML = "Il browser non supporta il web storage...";
            }
        }
    </script>
</head>
<body>
    <p><button onclick="clickCounter()" type="button">Contatore</button></p>
    <div id="result"></div>
    <p>Clicca il pulsante per vedere il contatore aumentare.</p>
    <p>Chiudi il browser e prova a verificare se il contatore riprende a contare dall'ultimo numero.</p>
</body>
</html>
```

WEB STORAGE: SESSION

- I problemi legati ai cookie sono stati risolti con l'introduzione di due nuovi oggetti: sessionStorage
- Consente di avere un meccanismo di persistenza dei dati distinto per ogni sessione di navigazione in ogni finestra, o scheda, del browser eliminando i dati alla chiusura
- Usando sessionStorage ad esempio sarebbe quindi possibile coordinare l'apertura contemporanea di due distinti account GMail sullo stesso browser
- Questo oggetto è da utilizzare nel caso in cui le informazioni da memorizzare sono di carattere temporaneo, quindi per processi di breve durata.

WEB STORAGE: SESSION

```html
<!DOCTYPE html>
<html>
<head>
    <script>
        function clickCounter() {
            if (typeof(Storage) !== "undefined") {
                if (sessionStorage.clickcount) {
                    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
                } else {
                    sessionStorage.clickcount = 1;
                }
                document.getElementById("result").innerHTML = "Contatore: " + localStorage.clickcount;
            } else {
                document.getElementById("result").innerHTML = "Il browser non supporta il web storage...";
            }
        }
    </script>
</head>
<body>
    <p><button onclick="clickCounter()" type="button">Contatore</button></p>
    <div id="result"></div>
    <p>Clicca il pulsante per vedere il contatore aumentare.</p>
    <p>Chiudi il browser e prova a verificare se il contatore riprende a contare dall'ultimo numero.</p>
</body>
</html>
```

WEB STORAGE

- Entrambi gli oggetti, sessionStorage e localStorage, implementano l'interfaccia Storage che raggruppa un insieme di metodi comuni ai due oggetti
- La memorizzazione è consentita solo per valori testuali e avviene tramite una coppia chiave valore
- I metodi per accedervi sono: setItem() per valorizzare la chiave e getItem() per ottenerne il valore
- In realtà, in formato stringa, si possono anche memorizzare valori numerici interi o decimali che in seguito, con l'ausilio dei metodi parseInt() o parseFloat(), si possono riottenere in formato numerico.

JSON

- La possibilità di memorizzare dati o comunque di rappresentarli in maniera da poterli trasferire tra programmi anche diversi è fondamentale nella programmazione

- In questo ambito si può far rientrare la serializzazione, cioè il processo di trasformazione di un oggetto in un formato facilmente memorizzabile e/o trasmissibile, ed il relativo processo inverso, cioè la deserializzazione

JSON

- Negli ultimi tempi si sta sempre più affermando la rappresentazione di oggetti in JSON (JavaScript Object Notation)
- Come indica il nome, questo formato nasce dalla notazione letterale degli oggetti di JavaScript, ma è diventato ormai uno standard de facto indipendente dal linguaggio di programmazione e dalla piattaforma.
- Sostanzialmente JSON utilizza un sottoinsieme della notazione letterale degli oggetti di JavaScript per rappresentarli sotto forma di stringa

```
{nome: "Mario", cognome: "Rossi"}
```

- viene rappresentato in JSON con la seguente stringa:

```
'{nome: "Mario", cognome: "Rossi"}'
```

- Da notare che abbiamo utilizzato il singolo apice come delimitatore di stringa per non interferire con i doppi apici delle stringhe interne. Anche un array può essere rappresentato nella notazione JSON

```
[1, "test", true]
```

- viene rappresentato come

```
'[1, "test", true]'
```

JSON

- Naturalmente è possibile rappresentare array di oggetti ed oggetti che prevedono array o altri oggetti come proprietà. La notazione JSON però non prevede la rappresentazione dei metodi di un oggetto.
- Naturalmente l'utilizzo di questo formato per la serializzazione e deserializzazione di oggetti in JavaScript è del tutto immediato, dal momento che in linea di principio sarebbe sufficiente utilizzare l'eval() della stringa per ottenere l'oggetto rappresentato
- In pratica però è opportuno effettuare alcuni controlli per evitare errori e l'introduzione di codice indesiderato.
- A questo scopo, a partire dalla versione 5 di ECMAScript è stato introdotto un parser nativo per la serializzazione e deserializzazione in notazione JSON

JSON

- Il parser è implementato dall'oggetto JSON che prevede due metodi specifici: parse() e stringify().

- Il metodo parse() prende in input una stringa JSON e genera il corrispondente oggetto JavaScript:

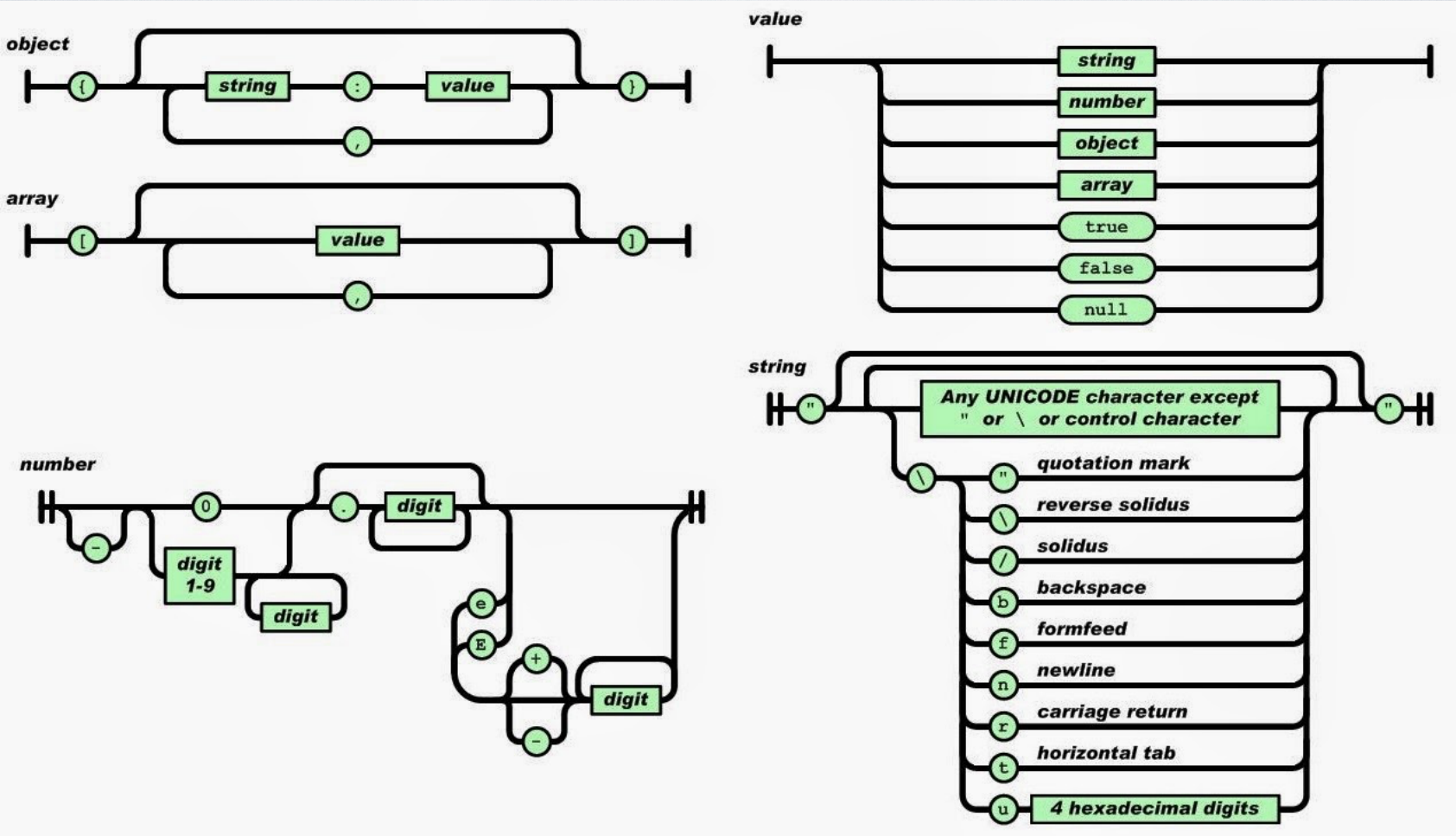
```
var marioRossi = JSON.parse('{nome: "Mario", cognome: "Rossi"}');
```

- Il metodo stringify() genera la rappresentazione JSON dell'oggetto passato come argomento:

```
var jsonMarioRossi = JSON.stringify({nome: "Mario", cognome: "Rossi"});
```

- Se l'oggetto da rappresentare contiene dei metodi, questi vengono semplicemente ignorati dal parser

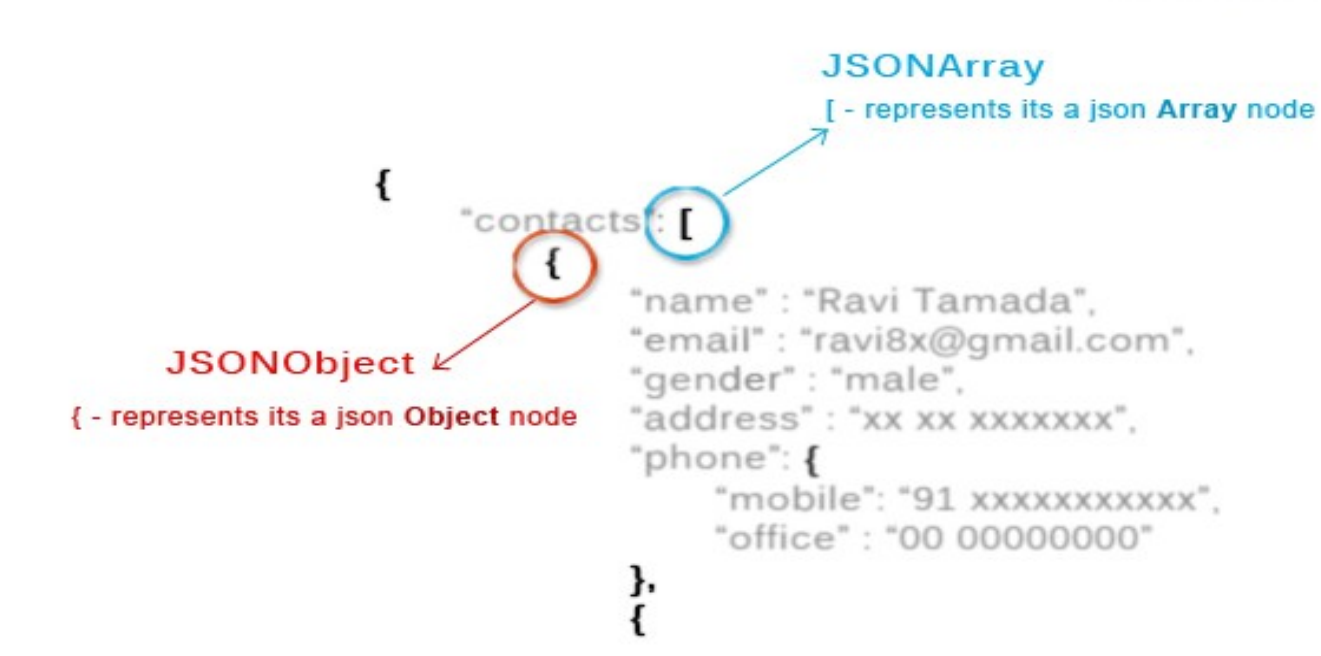
COME È UN JSON



FORMATI JSON

- I tipi di dati supportati da questo formato sono:
- **booleani** (true e false)
- **numeri:** interi, reali, virgola mobile
- **stringhe** racchiuse da doppi apici "
- **array:** sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre [ ]
- **array associativi**: sequenze coppie chiave-valore separate da virgole racchiuse in parentesi graffe

STRUTTURA FORMATO SON



ESEMPIO JSON

- esempio Json di un array di oggetti:

```
[
   {
      "nome":"Roberto",
      "eta":52
   },
   {
      "nome":"Antonio",
      "eta":32
   },
   {
      "nome":"Francesca",
      "eta":40
   }
]
```

WEB STORAGE: JSON

- Nel caso in cui si richieda la memorizzazione di un oggetto, si può utilizzare JSON (JavaScript Object Notation), un semplice formato utilizzato per lo scambio di dati
- La funzione JSON.stringfy() permette di ottenere una rappresentazione stringa di un oggetto
- la funzione inversa è JSON.parse() che permette di riottenere il formato originale

WEB STORAGE: JSON

```
<!DOCTYPE html>
<html>
<body>
<h2>Creare un oggetto da JSON</h2>

<p id="demo"></p>
<script>
var text = '{"employees":[' +
'{"firstName":"Antonio","lastName":"Lezzi" },' +
'{"firstName":"Mario","lastName":"Rossi" },' +
'{"firstName":"Giuseppe","lastName":"Verdi" }]}';

obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

SESSION STORAGE: ESEMPIO

```
<script type="text/javascript">
    var oggetto = {
        "nome": "Antonio",
        "cognome": "Lezzi",
        "email": "email@gmail.com"
    };
    stringaOggetto = JSON.stringify(oggetto);
    sessionStorage.setItem("credenziali", stringaOggetto);
    stringaOggettoRipristinato = sessionStorage.getItem("credenziali");
    var oggettoRipristinato = JSON.parse(stringaOggetto);
    alert(oggettoRipristinato["email"]);
</script>
```

- Nell'esempio si poteva procedere in modo analogo con l'oggetto localStorage.
- Per quanto riguarda la memorizzazione di variabili stringhe i metodi sessionStorage.setItem("chiave","valore") e temp=sessionStorage.getItem("chiave") si possono sostituire rispettivamente con sessionStorage.chiave="valore" e temp=sessionStorage.chiave.

WEB STORAGE: INTERCETTARE MODIFICHE

- Nel caso in cui fosse necessario rilevare eventuali modifiche alle variabili memorizzate, si può intercettare l'evento storage, notificato ogni qualvolta i dati sono modificati
- Questo evento mette a disposizione una serie di proprietà, accessibili solamente in lettura, contenenti alcune informazioni relative alla modifica effettuata:
  - **key**: la chiave dell'oggetto modificato.
  - **oldValue**: il valore associato alla chiave prima della modifica.
  - **newValue**: il valore associato alla chiave dopo la modifica.
  - **url**: l'indirizzo del documento in cui ha avuto luogo la modifica.
  - **storageArea**: riferimento all'oggetto, che rappresenta i dati, nel quale è stata rilevata la modifica.

WEB STORAGE: ESEMPIO EVENTO

```
<script type="text/javascript">
    if (window.addEventListener) {
        window.addEventListener("storage", modificaDati, false);
    } else if (window.attachEvent) { //per Internet Explorer 6,7 e 8
        Window.attachEvent("onstorage", modificadati);
    }

    function modificaDati(event) {
        var dataStorage = event || window.event;
        alert("chiave: " + dataStorage.key + " – valore:" +
dataStorage.newValue);
    }
</script>
```

WEB STORAGE: PRIVACY

- I cookie in passato sono stati visti come una minaccia alla privacy, molti utenti hanno deciso di disabilitarli con la conseguente riduzione dei servizi online.
- Senza cookie ad esempio non si riesce ad utilizzare Gmail, Twitter e molte altre applicazioni web. Il Web Storage amplificando le potenzialità dei cookie rende ancora più sensibile il tema della privacy
- Ad esempio se un ente in grado di distribuire i suoi contenuti su più siti assegnasse un identificativo univoco salvato nell'area di storage locale di un utente, potrebbe essere in grado di tracciare i suoi movimenti nel web
- Così facendo sarebbe più semplice focalizzare i suoi interessi in modo tale da poter predisporre della pubblicità mirata.

SELEZIONARE ELEMENTI

- Possiamo ad esempio accedere agli elementi della pagina utilizzando diversi criteri di selezione.
- Se abbiamo un elemento con un valore per l'attributo id, come quello del seguente esempio:

```
<p id="mioParagrafo">Questo &egrave; un paragrafo</p>
```

- possiamo utilizzare il metodo getElementById() dell'oggetto document come mostrato dal seguente codice:

```
var p = document.getElementById("mioParagrafo");
```

- È un metodo uno più utilizzato per la gestione del DOM, restituisce un oggetto che rappresenta il nodo di tipo elemento che ha l'attributo id con il valore specificato
- Se l'elemento non esiste viene restituito il valore null, mentre se esistono più elementi con lo stesso id viene restituito il primo individuato.

SELEZIONARE ELEMENTI

- Il metodo getElementsByName() restituisce l'elenco dei nodi della pagina il cui valore dell'attributo name corrisponde a quello del parametro.
- Restituisce un elenco di oggetti NodeList, cioè una struttura dati simile ad un array contenente nodi del DOM.
- È anche possibile individuare gli elementi di una pagina in base al loro tag utilizzando il metodo getElementsByTagName()

```
var listaParagrafi = document.getElementByTagName("p");
```

- In questo caso il metodo restituisce sotto forma di NodeList l'elenco dei nodi corrispondenti al tag specificato come parametro. Specificando come parametro la stringa "*", getElementsByTagName() restituisce l'elenco di tutti i nodi della pagina.
- Se utilizziamo il metodo getElementByClassName() possiamo ottenere l'elenco dei nodi a cui è stato assegnato un determinato valore come attributo class.

QUERYSELECTOR

- Tra le novità più introdotte più di recente nelle specifiche del DOM c'è la possibilità di selezionare gli elementi di una pagina utilizzando i selettori CSS
- Sono due i metodi che consentono questo approccio: querySelector() e querySelectorAll()
- Il primo restituisce il primo elemento trovato, l'altro l'elenco di tutti gli elementi individuati dal selettore. Ad esempio, il seguente codice restituisce l'elenco dei di classe messaggio:

```
var divList = document.querySelectorAll("div.messaggio");
```

- Il seguente esempio, invece, seleziona l'elemento con id mioParagrafo, risultando quindi alternativo a getElementById():

```
var p = document.querySelector("#mioParagrafo");
```

INNER HTML

- Una volta individuato l'elemento o gli elementi presenti su una pagina, possiamo modificarne il contenuto o altre caratteristiche sfruttando proprietà e metodi specifiche dei nodi di tipo elemento

- Ad esempio, la proprietà innerHTML rappresenta il contenuto HTML di un elemento ed è accessibile sia in lettura che in scrittura. Il seguente codice modifica il contenuto di un paragrafo:

```
var p = document.getElementById("mioParagrafo");
p.innerHTML = "Testo del <b>paragrafo</b>";
```

ALTRI METODI

- Altri metodi dei nodi di tipo elemento ci consentono di analizzare e modificare i suoi attributi, come quelli riepilogati nella seguente tabella:
- **hasAttribute(attrName)** : restituisce true se l'elemento ha l'attributo specificato come parametro
- **hasAttributes()** : restituisce true se l'elemento ha almeno un attributo valorizzato
- **getAttribute(attrName)** : restituisce il valore dell'attributo specificato come parametro
- **setAttribute(attrName, value)** : imposta un valore per un attributo

ATTRIBUTI E PROPRIETÀ

- Il seguente esempio mostra come assegnare un'immagine di default agli elementi <img> di una pagina privi di un'immagine associata:

```
var imgList = document.getElementsByTagName("img");
for(var i=0; i < imgList.length; i++){
    if (!imgList[i].hasAttribute("src"))
        imgList[i].setAttribute("src", "default.png");
}
```

- Alcuni attributi sono direttamente accessibili via JavaScript come proprietà dell'elemento. È ad esempio il caso degli attributi `id`, `name`, `src` e `href`. Alla luce di questa possibilità, possiamo riscrivere il codice precedente nel seguente modo:

```
var imgList = document.getElementsByTagName("img");
for(var i=0; i < imgList.length; i++){
    if (!imgList[i].src)
        imgList[i].src = "default.png";
}
```

ATTRIBUTI E PROPRIETÀ

- Dal momento che un elemento del DOM è sostanzialmente un oggetto JavaScript, possiamo interagire con esso come un qualsiasi altro oggetto

```
var img = document.getElementById("miaImmagine");
img.miaProprieta = "nuovo valore";
```

- L'aggiunta di una nuova proprietà all'elemento del DOM non ha alcuna ripercussione sull'HTML, ma può risultare utile per fare delle personalizzazioni tramite JavaScript, si può assegnare ad una immagine informazioni testuali o numeriche per essere poi recuperate in un secondo momento

- Si deve fare distinzione tra le proprietà e gli attributi di un elemento:

- una proprietà aggiunta ad un elemento del DOM non è gestibile tramite getAttribute() e setAttribute(), dal momento che non diventa un attributo HTML.

- gli attributi non sono case-sensitive, mentre le proprietà di un oggetto JS lo sono

- I valori assegnabili agli attributi sono soltanto stringhe, alle proprietà possiamo assegnare qualsiasi tipo di dato.

```
img.getAttribute("src");
img.getAttribute("SRC");
```

ATTRIBUTI E PROPRIETÀ

- Abbiamo detto che per alcuni attributi è prevista una corrispondente proprietà dell'elemento del DOM, la proprietà riflette generalmente il valore dell'attributo, ma va tenuto presente che non sono esattamente la stessa cosa.

- Prendiamo ad esempio le proprietà src di un elemento <img> e href di un elemento <a>. Mentre il valore dell'attributo restituito da getAttribute() restituisce sempre il valore presente nell'HTML, il valore restituito dalle proprietà src e href è l'URL completo alla risorsa, salvo errata interpretazione delle specifiche W3C da parte di qualche browser. Quindi, se abbiamo il seguente codice HTML:

```
<img id="miaImmagine" src="default.png" />
```

- le seguenti istruzioni restituiscono valori diversi:

```
var img = document.getElementById("miaImmagine");
console.log(img.src); //"http://www.site.it/default.png"
console.log(img.getAttribute("src")); //"default.png"
```

ATTRIBUTI E PROPRIETÀ

- Un altro caso tipico è la proprietà checked di un elemento <input>, la proprietà ha un valore booleano che indica se l'elemento è selezionato o meno, l'attributo corrispondente restituisce la stringa "checked" o una stringa vuota.

- Un aspetto più insidioso è legato all'interazione con l'utente. Consideriamo un elemento <input> con un suo valore predefinito:

```
<input id="txtNome" type="text" value="Mario" />
```

- Inizialmente i valori dell'attributo value e della corrispondente proprietà coincidono, se l'utente cambia il valore della casella di testo avremo valori differenti tra proprietà e attributo

- In particolare avremo che il valore della proprietà value sarà sincronizzata con il nuovo valore inserito dall'utente, mentre l'attributo continuerà ad avere il valore iniziale

NAVIGAZIONE DEI NODI

- Alcuni metodi del DOM ci consentono di analizzare e muoverci all'interno della struttura di un documento

- La proprietà childNodes di un elemento contiene l'elenco dei nodi figli dell'elemento sotto forma di NodeList

```
<div id="mainDiv">
    <h1>Titolo</h1>
    <p>Un paragrafo</p>
    <p>Un altro paragrafo</p>
</div>
```

- Possiamo scoprire il contenuto degli elementi figli del principale utilizzando il seguente codice JavaScript:

```
var div = document.getElementById("mainDiv");

for (var i = 0; i < div.childNodes.length; i++) {
    console.log(div.childNodes[i].innerHTML);
}
```

NAVIGAZIONE DEI NODI

- Sono previsti dei metodi per semplificare la navigazione tra i nodi figlio evitando di ricorrere ad un ciclo
- i metodi firstChild() e lastChild() che restituiscono rispettivamente il primo figlio e l'ultimo figlio di un elemento.
- Il metodo parentNode() restituisce il nodo genitore dell'elemento corrente, permettendoci di risalire la struttura gerarchica dell'albero del DOM.
- La navigazione del DOM può avvenire non solo verticalmente , analizzando i figli ed il genitore di un elemento, ma anche allo stesso livello dell'albero, scoprendo quali sono i nodi fratelli dell'elemento corrente
- I metodi nextSibling() e previousSibling() restituiscono il nodo fratello successivo e precedente nella struttura del DOM.
- Non è previsto un metodo per ottenere tutti i fratelli del nodo corrente, ma è possibile aggirare il problema utilizzando il seguente codice JavaScript:

```
var me = document.getElementById("mainDiv");
var allSiblings = me.parentNode.childNodes;
var mySiblings = [];

for (i = 0; i < allSiblings.length; i++) {
    if (allSiblings[i] !== me) {
        mySiblings.push(allSiblings[i]);
    }
}
```

NAVIGAZIONE DEI NODI

- Il DOM ci consente di modificare la struttura di un documento, possiamo ad esempio creare elementi ed attributi e aggiungerli al DOM avendo un effetto immediato sulla pagina HTML.

```
var mainDiv = document.getElementById("mainDiv");
var img = document.createElement("img");
var srcAttr = document.createAttribute("src");

srcAttr.value = "default.png";
img.setAttributeNode(srcAttr);

mainDiv.appendChild(img);
```

INSERT BEFORE

- Il metodo insertBefore() per inserire un nodo prima di un altro (ad esempio in un elenco). Il metodo prevede due paremetri: il nodo da inserire ed il nodo prima del quale inserirlo nella lista.
- document.body.insertBefore(img, mainDiv);
- Anche se non esiste un metodo per inserire un nodo dopo un altro (un ipotetico metodo insertAfter), possiamo ovviare a questa mancanza utilizzando il seguente approccio:
- document.body.insertBefore(img, mainDiv.nextSibling);
- Nel caso in cui non esista un fratello successivo, mainDiv.nextSibling è null e il nodo viene aggiunto come ultimo elemento, con lo stesso risultato di appendChild().

REPLACE E REMOVE

- Il metodo replaceChild() sostituisce uno dei figli di un elemento con un nuovo nodo. Facendo riferimento all'esempio di prima, possiamo sostituire il primo un nodo figlio del div mainDiv con il nodo <img> appena creato:
- mainDiv.replaceChild(img, mainDiv.firstChild());
- I metodi removeChild() e removeAttribute() consentono di eliminare elementi ed attributi dal DOM. Ad esempio, per rimuovere il primo nodo figlio del mainDiv eseguiamo la seguente istruzione:
- mainDiv.removeChild(mainDiv.firstChild());
- mentre per rimuovere l'attributo class di un elemento scriviamo il codice seguente:
- mainDiv.removeAttribute("class");

GESTIONE EVENTI

- Gli eventi sono ci consentono di gestire il comportamento delle nostre applicazioni al verificarsi di una certa interazione dell'utente, oppure di rispondere alle sollecitazioni che vengono da altre applicazioni o dal sistema stesso.
- Per intercettare gli eventi che vengono scatenati, utilizziamo il meccanismo degli handler (o dei listener). Possiamo definire come handler una funzione di callback che viene associata ad un certo evento.
- I meccanismi per associare eventi e handler sono sostanzialmente tre:
  - Via codice, sfruttando la funzione addEventListener().
  - All'interno del markup HTML, sfruttando speciali attributi dei tag;
  - Nel codice JavaScript, associando specifiche proprietà degli elementi del DOM;

GESTIONE EVENTI

- La funzione addEventListener() è un metodo esposto dagli elementi del DOM e rappresenta la più comune tra le modalità usate per associare un evento al rispettivo handler

```
elemento.addEventListener(evento, callback, [useCapture]);
```

- evento: È una stringa che definisce l'evento che vogliamo gestire (es. "load" o "click")
- callback: qui inseriamo la chiamata a callback che definisce l'hanlder vero e proprio
- useCapture: Serve per "forzare" la priorità di gestione di un certo evento. Questo nei casi in cui

    1. abbiamo associato diversi handler allo stesso evento (allora il primo diventa quello con useCapture = true)
    2. serve dare priorità all'handler posizionato su un certo livello dell'albero (normalmente gli handler sono eseguiti e propagati dalle foglie verso la radice del DOM)
    3. Per default è impostato a false.

GESTIONE EVENTI

- Supponiamo di avere il classico link:

```
<a href="#" id="test">test</a>
```

- Possiamo definire una funzione di callback che gestisca l'evento click sul link e associarla all'evento direttamente da codice (a livello di logica applicativa):

```
<script>
// definisce la funzione callback
function clickOnTest(event){
    console.log('Click su test');
}

// ottiene l'elemento 'test'
var test = document.getElementById('test');

// associa l'evento click alla callback
test.addEventListener('click', clickOnTest);
</script>
```

- Naturalmente avremmo ottenuto lo stesso risultato utilizzando una funzione implicita:

```
test.addEventListener('click', function(event) {
    console.log('Click su test');
});
```

GESTIONE EVENTI

- Il parametro event, si tratta di un oggetto che contiene alcune informazioni circa l'evento che è stato scatenato: cose come il bottone del mouse che è stato premuto o il tasto della tastiera che abbiamo schiacciato.

- addEventListener permette una gestione più raffinata degli handler, consente di definirne e gestirne più di uno per lo stesso evento e funziona con qualunque elemento del DOM (non solo con gli elementi HTML).

- Altro vantaggio sta nel fatto che possiamo rimuovere l'associazione tra evento e handler utilizzando la funzione removeEventListener che prende gli stessi parametri di addEventListener ma ha la funzione opposta

```
test.removeEventListener('click', clickOnTest);
```

HANDLER ASSOCIATI

- Possiamo anche inserire la chiamata alla nostra funzione di callback direttamente nella definizione di un tag HTML

```
<a href="#" onclick="alert('ciao')">Link</a>
```

- In questo caso l'evento onclick è un attributo che prende come valore la chiamata alla relativa callback

- Modalità è sempre meno utilizzata e poco consigliabile perché favorisce troppo l'accoppiamento tra layout della pagina (HTML) e logica applicativa (JavaScript)

- In un'ottica di buona manutenzione del software è sempre meglio tentare di tenere le due cose separate.

THIS E HANDLER

- In questa modalità quando la funzione di callback viene chiama, questa acquisisce lo scope dell'elemento del DOM cui è associata. Il risultato è che l'oggetto this assume il valore dell'elemento

```
<a href="#" onclick="alert(this.innerHTML)">Ciao</a>
```

- Cliccando sul link otterremo una finestra che riporterà "Ciao" il contenuto dell'elemento!

- Possiamo associare gli handler degli eventi direttamente alle relative proprietà degli oggetti del DOM, con una sintassi simile a questa:

```
Oggetto.evento = handler;
```

- Per esempio:

```
document.images[0].onload = alert('Immagine caricata');
```

- Anche questa modalità è sempre meno utilizzata, più che altro perché risulta scomoda

THIS E HANDLER

- Nella gestione di un evento potremmo essere interessati ad ottenere informazioni specifiche su di esso e sull'elemento HTML su cui esso si è verificato. Immaginiamo ad esempio di avere assegnato un unico gestore dell'evento clic a tutti i pulsanti di una pagina:

```
var buttons = document.getElementsByTagName("button");
var handler = function() { console.log("Clic..."); };
for(var i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", handler);
}
```

- Come si fa a sapere quale pulsante è stato cliccato? Un possibile approccio consiste nel far riferimento all'oggetto this all'interno della funzione che gestisce l'evento:

```
var buttons = document.getElementsByTagName("button");
var handler = function() { console.log("Clic su " + this.innerHTML); };
for(var i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", handler);
}
```

- L'oggetto this rappresenta normalmente l'elemento su cui si è verificato l'evento corrente, l'oggetto this non ci consente di avere informazioni specifiche sull'evento.

THIS E HANDLER

- Il DOM prevede che ad ogni gestore di eventi venga passato come parametro l'oggetto event contenente informazioni su di esso. Alcune di queste informazioni sono generiche, cioè presenti per qualsiasi tipo di evento, altre sono specifiche per il particolare evento.
- Le informazioni specifiche quando esploreremo i principali eventi che possiamo gestire
- Per il momento segnaliamo la presenza della proprietà target che rappresenta l'elemento su cui si è verificato l'evento, in maniera indipendente da altri fattori come ad esempio il flusso degli eventi e quindi con maggiori garanzie rispetto all'oggetto this
- Il gestore dell'evento clic dell'esempio precedente andrebbe dunque riscritto nel seguente modo:

```
var handler = function(e) { console.log("Clic su " + e.target.innerHTML); };
```

- dove il parametro e indica l'oggetto event associato al clic e la sua proprietà target è il pulsante cliccato.
- Insieme all'elemento su cui si è verificato l'evento possiamo individuare anche il tipo di evento tramite la proprietà type. Questa proprietà restituisce una stringa corrispondente al nome dell'evento:

```
var button = document.getElementById("btnClicca");
var handler = function(e) { console.log("Evento: " + e.type); };
button.addEventListener("click", handler);
```

INTEGRAZIONE CON IL BROWSER

- Per scrivere a video, pagina in cui si trova lo script, è possibile utilizzare il metodo document.write(), le stringhe da visualizzare possono essere concatenate utilizzando l'operatore di concatenazione '+'

```
<html>
<head>
<script type="text/javascript">
<!-- Nasconde lo script ai browser obsoleti
// Questa funzione genera una riga orizzontale della larghezza scelta.
function bar(widthPct) {
  document.write("<hr align='left' width='" + widthPct + "%' />");
}
// Questa funzione genera del testo preceduto da un titolo di dato livello.
function output(headLevel, headText, text) {
  document.write("<h" + headLevel + ">" + headText + "</h" + headLevel + "><p>" + text + "</p>");
}
// Fine del codice da nascondere -->
</script>
</head>
<body>
<script type="text/javascript">
<!-- Nasconde lo script ai browser obsoleti
bar(25)
output(2, "JavaScript Rules!", "Usare JavaScript è facile...")
// Fine del codice da nascondere -->
</script>
<p>Questo è normale HTML, a differenza di quello generato dal codice sopra.</p>
</body>
</html>
```

MVC E MVVM

- MVC (Model-View-Controller) è un pattern che si pone l'obiettivo di disaccoppiare l'interfaccia utente dal modello dei dati in modo da ottenere un'architettura più flessibile.
- **Model**: Rappresenta i dati da gestire tramite l'interfaccia utente
- **View**: Corrisponde all'insieme degli elementi dell'interfaccia utente che mostrano i dati del Model, cose come label, caselle di testo, immagini, etc.
- **Controller**: È il componente che fa da collante tra il Model e la View e definisce le funzionalità dell'applicazione; in pratica il Controller implementa la relazione tra il Model e la View.
- Per fissare meglio le idee, immaginiamo di dover creare un'interfaccia Web per la gestione di dati anagrafici. I ruoli del pattern MVC andranno così suddivisi:
    - il Model sarà costituito dall'oggetto che rappresenta una persona;
    - la View dall'HTML che mostra i dati della persona e ne permette l'interazione con l'utente, come ad esempio il salvataggio dopo una modifica;
    - il Controller da un oggetto che si occuperà di fornire i dati da mostrare alla View e di gestire i comandi impartiti dall'utente.

MVC E MVVM

```
<label for="txtNome"><input id="txtNome" type="text" value=""/><br/>
<label for="txtCognome"><input id="txtCognome" type="text" value=""/><br/>
<button id="btnSalva"/>Salva</button><br/>
```

- che sarà gestita dal seguente codice JavaScript:

```
var model = { nome: "Mario", cognome: "Rossi" };
var view = {
        txtNome: document.getElementById(“txtNome"),
        txtCognome: document.getElementById(“txtCognome"),
        btnSalva: document.getElementById("btnSalva")
};
var controller = {
    init: function() {
        view.txtNome.value = model.nome;
        view.txtCognome.value = model.cognome;
        view.btnSalva.onclick = controller.salva;
    },
    salva: function() {
        model.nome = view.txtNome.value;
        model.cognome = view.txtCognome.value;
        //invia il model al server
        invia(model);
    }
};
```

MVC E MVVM

- Per la definizione dell'oggetto view abbiamo fatto ricorso all'oggetto document, che fa parte del DOM (Document Object Model) il modello a oggetti delle pagine HTML
- L'oggetto controller ha due metodi:
  - init(), che inizializza la view associando a ciascun elemento grafico il corrispondente valore dell'oggetto model;
  - salva(), che assegna all'oggetto model i valori prelevati dall'oggetto view ed invia il model al server tramite un'ipotetica funzione invia().
- Notiamo che la funzione init() si occupa anche di assegnare al click sul pulsante btnSalva l'esecuzione del metodo salva() del controller.
- Questa architettura rende indipendente il Model dalla View, facendo sì che eventuali modifiche implementative sulla View non interferiscano sul Model e viceversa.

MVC E MVVM

- Una variante del pattern MVC è MVVM, Model View ViewModel.
- Propone un ruolo più attivo della View rispetto a MVC: la View è in grado di gestire eventi, eseguire operazioni ed effettuare il data-binding
- In questo contesto, quindi, alcune delle funzionalità del Controller vengono inglobate nella View, la quale si appoggia su un'estensione del Model: il ViewModel.
- Il ViewModel è quindi un Model esteso con funzionalità per la manipolazione dei dati e per l'interazione con la View

EVENTI

- La sintassi per richiamare un evento in uno script è la seguente: Oggetto.evento = handler; Esistono varie categorie di eventi:

1. Eventi attivabili dai tasti del mouse
2. Eventi attivabili dai movimenti del mouse
3. Eventi attivabili dal trascinamento del mouse (drag and drop)
4. Eventi attivabili dall'utente con la tastiera
5. Eventi attivabili dalle modifiche dell'utente
6. Eventi legati al "fuoco"
7. Eventi attivabili dal caricamento degli oggetti
8. Eventi attivabili dai movimenti delle finestre
9. Eventi legati a particolari bottoni
10. Altri e nuovi tipi di eventi

EVENTI: TASTI MOUSE

- Lista eventi:

  1. **onClick**: attivato quando si clicca su un oggetto;
  2. **onDblClick**: attivato con un doppio click;
  3. **onMouseDown**: attivato quando si schiaccia il tasto sinistro del mouse;
  4. **onMouseUp**: attivato quando si alza il tasto sinistro del mouse precedentemente schiacciato;
  5. **onContextMenu**: attivato quando si clicca il tasto destro del mouse aprendo il ContextMenu.

- Il doppio click è un evento che ingloba gli altri e, per la precisione, attiva in successione onmousedown, onmouseup, onclick.

- **Tag di applicazione**

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

EVENTI: MOVIMENTO MOUSE

- Lista eventi:

  1. **onMouseOver**: attivato quando il mouse si muove su un oggetto;

  2. **onMouseOut**: attivato quando il mouse si sposta da un oggetto;

  3. **onMouseMove**: si muove il puntatore del mouse, ma poiché questo evento ricorre spesso (l'utilizzo del mouse è frequente), non è disponibile per default, ma solo abbinato con la cattura degli eventi, che si spiegherà in seguito.

- **Tag di applicazione**

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

EVENTI: TRASCINAMENTO MOUSE

- Lista eventi:

1. **onDragDrop**: evento attivato quando un utente trascina un oggetto sulla finestra del browser o quando rilascia un file sulla stessa;
2. **onMove**: attivato quando un oggetto muove una finestra o un frame;
3. **onDragStart**: evento attivato appena l'utente inizia a trascinare un oggetto;
4. **onDrag**: attivato quando il mouse trascina un oggetto o una selezione di testo nella finestra dello stesso browser o anche di un altro o anche sul Desktop;
5. **onDragEnter**: attivato appena l'utente trascina un oggetto su un obiettivo valido dello stesso o di un altro browser;
6. **onDragOver**: attivato quando l'utente trascina un oggetto su un obiettivo valido ad ospitarlo, ed è simile all'evento precedente, ma viene attivato dopo quello;

EVENTI: TRASCINAMENTO MOUSE

7. **onDragLeave**: attivato quando l'utente trascina un oggetto su un obiettivo adatto per ospitarlo, ma non vi viene rilasciato;

8. **onDragEnd**: attivato quando l'utente rilascia l'oggetto al termine del trascinamento.

9. **onDrop**: attivato quando il mouse si alza il tasto del mouse in seguito ad un'operazione di trascinamento

- **Tag di applicazione**

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

EVENTI: TASTIERA

- Lista Eventi:

1. **onKeyPress**: evento attivato quando un utente preme e rilascia un tasto o anche quando lo tiene premuto;
2. **onKeyDown**: attivato quando viene premuto il tasto;
3. **onKeyUp**: evento attivato quando un tasto, che era stato premuto, viene rilasciato;
4. **onHelp**: attivato quando un utente preme il tasto F1;

EVENTI: MODIFICHE UTENTE

- **onChange**: Attivato quando il contenuto di un campo di un form o modulo è modificato o non è più selezionato. Viene utilizzato anche con gli oggetti FileUpload, Select, Text, TextArea

```
<input type="text" value="Enter email address" name="userEmail" onChange=validateInput(this.value) />
<script type="text/javascript">
  this.myForm.userEmail.focus();
  this.myForm.userEmail.select();

  function validateInput()
  {
    userInput = new String();
    userInput = this.myForm.userEmail.value;
    if (userInput.match("@"))
      alert("Thanks for your interest.");
    else
      alert("Please check your email details are correct before submitting");
  }
</script>
```

- **onCellChange**: Attivato quando si modifica un elemento in un Database, per questa sua caratteristica ha un uso non propriamente legato a JavaScript;

EVENTI: MODIFICHE UTENTE

- **onPropertyChange**: Evento attivato quando cambia la proprietà di un elemento;
- **onReadyStateChange**: Evento attivato quando lo stato del caricamento di un elemento cambia, l'evento è utile, ad esempio, per verificare che un elemento sia stato caricato.
- **Tag di applicazione**

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

EVENTI: FUOCO

- **onFocus**: Questo handler è l'opposto di onBlur per cui si attiva quando l'utente entra in un campo;

- **onBlur**: Viene attivato quando il puntatore del mouse o il cursore esce dalla finestra corrente utilizzando il mouse o il carattere TAB. Applicato ai moduli, invece, tale handler si avvia se si esce dal campo il cui tag contiene il controllo;

```
Enter email address <input type="text" value="" name="userEmail" onBlur=addCheck()>
<script type="text/javascript">
function addCheck(){
  alert("Please check your email details are correct before submitting")
}
</script>
```

- **onSelect**: Attivabile quando si seleziona del testo all'interno di una casella di testo sia col mouse sia tenendo premuto SHIFT e selezionando con i tasti Freccia;

- **onSelectStart**: Si attiva quando si inizia a selezionare un evento;

EVENTI: FUOCO

- **onBeforeEditFocus**: Si attiva con un doppio click o con un click su un oggetto che ha già la selezione, quando questo è in DesignMode;
- **onLoseCapture**: Si attiva quando un oggetto perde la cattura del mouse.
- **Tag di applicazione**

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

EVENTI: CARICAMENTO OGGETTI

- **onLoad**: Questo handler funziona nel caricamento di oggetti, per lo più finestre e immagini;
- **onUnload**: È l'opposto del precedente e funziona quando si lascia una finestra per caricarne un'altra o anche per ricaricare la stessa (col tasto refresh);
- **onAbort**: L'Handler dell'evento onAbort permette di specificare del codice nel caso in cui l'utente blocchi il caricamento di un oggetto, o che si blocchi il caricamento di un'immagine. Questo handler usa le seguenti proprietà dell'evento.

```
<IMG NAME = "myPic" SRC = "images/myPic.gif" onAbort = "alert('Loading of image aborted!')">
```

- **onError**: Si attiva quando il caricamento di un oggetto causa un errore, ma solo se questo è dovuto ad un errore di sintassi del codice e non del browser così funziona su un link errato di un'immagine della pagina, ma non su un link errato di caricamento di una pagina intera. Opera non gestisce questo evento, ormai obsoleto: per una corretta gestione degli errori si utilizza il costrutto try ... catch;

EVENTI: CARICAMENTO OGGETTI

- **onBeforeUnload**: Questo handler funziona allo stesso modo di onUnload ma si carica in un momento prima;
- **onStop**: Questo handler funziona quando si ferma il caricamento della pagina con il tasto stop del browser e dovrebbe funzionare anche allo stesso modo di onUnload caricandosi prima di questo ma dopo onBeforeUnload.
- **Tag di applicazione**

1. onLoad Questo gestore è usato con i tag <BODY> e <FRAMESET> e da JavaScript 1.1anche con <IMG> mentre in Explorer occorre aggiungere anche i tag <SCRIPT>, <LINK>, <EMBED>, <APPLET>. In JavaScript 1.2 in Netscape si aggiunge anche il tag <LAYER>.
2. onUnload Questo gestore è usato con i tag <BODY> e <FRAMESET> anche in Internet Explorer.
3. onAbort Questo gestore è usato solo con il tag<IMG> anche in Internet Explorer.
4. onError Questo gestore è usato solo con il tag<IMG> e con Window mentre in Internet Explorer anche con <OBJECT> e <STYLE>.
5. onBeforeUnload Questo gestore è usato con i tag <BODY> anche in Internet Explorer.
6. onStop Questo gestore è usato con i tag <BODY> anche in Internet Explorer.

ESERCIZI

- Esercizio 1
  - Creare una funzione caricata in avvio (onLoad su body) che cambi il colore di sfondo ongli 15 secondi (window.setTimeout(comando,millisecondi);
  - Il colore di sfondo (document.bgColor) viene preso da un array.
  - Quando tutto l'array è stato visitato, ricomincia.
  - Inserire due bottoni, uno per fermare e l'altro per continuare l'esecuzione (onClick)
- Esercizio 2
  - Fare una tabella in una pagina che permetta di impostare un bookmark a seconda del browser.
  - La proprietà document.title contiene il titolo della pagina.
  - Per explorer: window.external.AddFavorite(bookmarkurl,bookmarktitle).
  - Per netscape: testo che dice "Premere CTRL+D per salvare la pagina nei segnalibri".

ESERCIZI

- Esercizio 3
    - Cambiare il colore di un testo non in un link, ne in un pulsante di un form passando il mouse sopra un elemento sensibile.
    - Utilizzare i fogli di stile.
    - Per risolvere considerare il fatto che tramite il dom di javascript è possibile accedere alle proprità di qualunque elemento con un nome della pagina tramite la sintassi nomeelemento.proprietà = valore (in questo caso nomeelemento.className=nomeclasse
- Esercizio 4
  Scrivere un documento HTML contenente una form contenente i seguenti campi:
  Aggiungere al documento HTML una funzione JavaScript che esegue i seguenti controlli:
  • verifica che il cognome non sia vuoto;
  • verifica che il nome non sia vuoto;
  • verifica che la matricola sia un numero;
  • verifica che sia stata selezionata una regione;
  • verifica che o l'email o il telefono siano non vuoti.
  • richieste particolari (area di testo editabile di 12 righe per 60 colonne)
  • bottone di invio e bottone di reset
    - Salvare i dati nel sessionStorage e nel DB, infine mostrare una lista di elementi, che se cliccati mostrano il dettaglio

Dott. Antonio Giovanni Lezzi