Dopo aver presentato le principali **novità introdotte da HTML5**, si discuta il concetto di **semantic markup**.
Si discutano nel dettaglio le principali funzionalità introdotte con l'avvento di HTML5.

HTML5:

- è uno **standard per strutturare e presentare il contenuto del World Wide Web**.
- è **pragmatico**, parte dalla considerazione che i browser decidono cosa implementare e cosa no.
- fornisce **supporto** per tutto ciò che è **JavaScript**.
- aggiunge **semantica**, migliore definizione delle componenti delle pagine web, i tag HTML forniscono informazioni aggiuntive utili per motori di ricerca.

Lista di **funzionalità** sotto il cappello HTML5:

1. **Nuovi elementi strutturali**, semantica (ad es., <header>, <footer>, <nav>,)
2. **Form 2.0** e validazione client-side
3. Supporto nativo dei browser per audio e video (<video>, <audio>)
4. Canvas API e SVG
5. **Web storage** = memoria che si trova dentro il browser e permette di memorizzare i dati come coppia (chiave, valore).
6. Offline application
7. Geolocation
8. **Drag & Drop**
9. Web Workers
10. Nuove API di comunicazione (Server Sent Events, Web Sockets, ...)

La sintassi è:

- più **permissive** di XHTML per garantire compatibilità.
- **case insensitive**, maiuscole e minuscole non interferiscono con la validazione della pagina
- I tag di chiusura non sono richiesti.
- Le virgolette e i valori sono opzionali per gli attributi.

HTML5 introduce 28 nuovi elementi come <section>, <header>, <footer>, <nav>, etc.

Semantic markup = Associa un **significato** al contenuto.

- Definizione e scelta dell'elemento migliore per l'oggetto che si vuole rappresentare.
- Sintassi che da l'idea di cosa si sta facendo sia a un **uomo** che a una **macchina** (browser).
- Pagine facili da capire e modificare per sviluppatore, adatte per mostrare il contenuto in maniera **comprensibile** a utenti con disabilità.

Dopo aver descritto le caratteristiche principali a che cosa servono i **fogli di stile CSS**, mostrando la **sintassi** e un **esempio** di regola CSS, si discuta in dettaglio il **CSS box model**.

CSS, Cascading Style Sheet (fogli di stile a cascata), impone delle regole sulla visualizzazione della pagina.

I fogli di stile servono per una gestione del sito facilitata infatti:

- riducono il carico di lavoro
- permettono di controllare il layout di pagine web multiple in un colpo solo

Il risultato sono pagine più leggere e facili da modificare.

Una regola CSS è composta di due parti

```
h1 {
  color: white;
  background: red;
}
```

- h1= **selettore**, definisce la parte del documento cui verrà applicata la regola
- {...} = **blocco delle dichiarazioni**, formato da coppie:
  - **proprietà** = (seguito da ":") definiscono un aspetto dell'elemento da modificare (margini, colore di sfondo, …) secondo il valore espresso
  - **valore** (seguito da ";")

I fogli di stile possono essere:

- **esterni** = il codice CSS è definito in un file separato dal documento
- **interni** = il codice CSS è compreso in quello del documento HTML
- **linea** = quando lo stile è interno al tag HTML target

Nel CSS box model tutti gli elementi HTML sono visti come delle box. Il CSS box model racchiude ogni elemento in un box e vi associa:

- **Content** = il contenuto della box, dove testo e immagini risiedono
- **Padding** = libera un'area attorno al content (È trasparente)
- **Border** = un bordo che circonda padding e content
- **Margin** = libera un'area attorno al bordo (È trasparente)

Nell'ambito del linguaggio CSS, si descrivano le **regole di valutazione in cascata.**

1. Per prima cosa bisogna **selezionare tutte le dichiarazioni che applicano a un elemento/proprietà per il media type richiesto**.
2. Il primo ordinamento è fatto in base a **peso e origine.**
    a. Per dichiarazioni normali, gli style sheet dell'autore della pagina sovrascrivono gli style sheet dell'utente che sovrascrivono quelli di default.
    b. Per dichiarazioni "importanti" (!important), gli style sheet dell'utente della pagina sovrascrivono gli style sheet dell'autore che sovrascrivono quelli di default.
3. Il secondo ordinamento è fatto in base alla **specificity**. Selettori più specifici sovrascrivono quelli generali.
4. Il terzo ordinamento è fatto in base **all'ordine**. Se due regole hanno stesso peso, origine e specificity, vince l'ultimo specificato.

Si descriva il funzionamento della funzione JavaScript **getElementById**. Si presenti inoltre un **esempio** dove la funzione getElementById viene richiamata a seguito di un **evento onClick** su un tag a scelta.

Ritorna l'elemento con uno specifico id. Ritorna null se non esiste l'elemento.

```
<button onclick="getElementById('demo').innerHTML=Date()">
What is the time?
</button>

<p id="demo"></p>
```

Si descriva nel dettaglio le **caratteristiche di JSON**, discutendo almeno uno **scenario applicativo** e le **differenze con XML**.

JSON, (JavaScript Object Notation) è:

- un formato adatto all'interscambio di dati fra applicazioni client-server. JSON è **text-based open standard** leggero disegnato per lo **scambio di dati** human-readable.
- usato quando si scrivono applicazioni basate su JavaScript.
- usato per serializzare e trasmettere **dati strutturati** tra server e applicazioni web su una connessione di rete.
- una sorta di dialetto XML.
- facile da leggere e scrivere.
- formato di scambio basato su testo e **molto leggero (a differenza di XML)**
- indipendente dal linguaggio.

La sintassi JSON è considerata come un sottoinsieme della sintassi JavaScript:

- i dati sono rappresentati come coppie nome/valore
- le parentesi graffe contengono oggetti e ogni nome è seguito dal : (colon), le coppie nome/valore sono separate da , (comma)
- Parentesi quadre definiscono array e sono separate da , (comma)

JSON supporta due strutture dati:

- Insieme di coppie nome/valore
- Lista ordinate di valori (array)

Possibile confrontare JSON e XML sulla base di tre fattori principali:

- **verbosità** = XML più verboso di JSON. Più veloce scrivere JSON per esseri umani.
- **utilizzo degli array** = XML usato per descrivere dati strutturati che non includono array. JSON include array.
- **parsing** = attività di analisi sintattica che permette di interpretare una stringa. Il metodo eval di Javascript fa il parsing di JSON. Quando applicato a JSON, eval ritorna l'oggetto descritto. In XML il parsing è più difficile.

Si può inoltre aggiungere che:

- JSON è simile a XML ma più leggero
- sono entrambi formati human readable e indipendenti dal linguaggio
- ogni valore in XML richiede apertura e chiusura di un tag, in JSON solo il valore.

Somiglianze tra JSON e XML, entrambi:

- sono semplici e aperti
- supportano unicode
- rappresentano dati autodescrittivi
- sono interoperabili e indipendenti della lingua

A partire dal codice JSON, si discuta quali problematiche si potrebbero affrontare nella modellazione dello stesso file usando XML. Si confrontino inoltre brevemente vantaggi e svantaggi di una modellazione XML e una modellazione JSON

```
{"car": [ {
"Name":"chevrolet chevelle malibu",
"Miles_per_Gallon":18,
"Cylinders":8,
"Displacement":307,
"Horsepower":130,
"Origin":"USA",
"Available": ["USA", "Europe"] },
 {...}, {...}]
}
```

Possibile confrontare JSON e XML sulla base di tre fattori principali:

- **verbosità** = XML più verboso di JSON. Più veloce scrivere JSON per esseri umani.
- **utilizzo degli array** = XML usato per descrivere dati strutturati che non includono array. JSON include array.
- **parsing** = attività di analisi sintattica che permette di interpretare una stringa. Il metodo eval di Javascript fa il parsing di JSON. Quando applicato a JSON, eval ritorna l'oggetto descritto. In XML il parsing è più difficile.

In questo caso l'utilizzo di array è importante per il campo "Available" e sarebbe difficile replicarlo in XML.

Nell'ambito della programmazione web basata su HTML e JavaScript, si discuta in dettaglio il ruolo **dell'HTML DOM**. Si proponga inoltre un **esempio** del suo utilizzo.
Nell'ambito del linguaggio di scripting JavaScript, si descriva il ruolo dell'**HTML DOM per la modifica degli elementi** di una pagina web e si proponga un esempio del suo utilizzo.

HTML DOM = È uno standard object model e programming interface per HTML. Definisce:

- Elementi HTML come oggetti
- Proprietà degli elementi HTML
- I metodi per accedere agli elementi HTML
- Gli eventi per tutti gli elementi HTML

In altre parole, è uno standard che definisce come ottenere, cambiare, aggiungere o modificare elementi HTML.

window = è la radice della gerarchia DOM e **rappresenta la finestra corrente del browser**. I figli dell'oggetto window sono:

- **navigator** = il browser (in quanto applicazione). Ad esempio, per sapere quale browser si sta utilizzando (Explorer/ Netscape).
- **document** = il contenuto della finestra.
- **location** = informazioni sull'URL corrente. Ad esempio, per caricare nella finestra un URL differente.
- **history** = elenco degli URL visitati di recente. Ad esempio, per tornare alla pagina Web precedente.

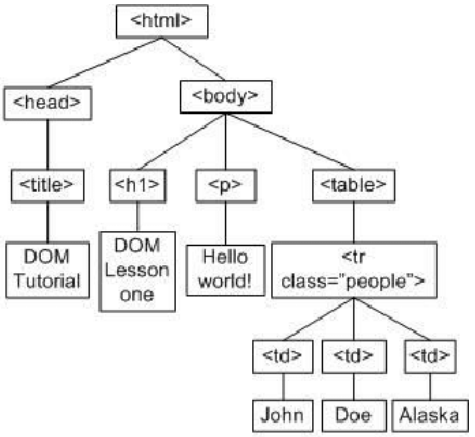
JavaScript usa HTML DOM per modificare tutti gli elementi di una pagina web. Quando una pagina è caricata il browser crea il DOM della pagina, la pagina viene rappresentata come un albero. DOM è definito dal browser (Definizione è fatta separatamente per Explorer, Mozilla, etc. e possono nascere incompatibilità).

Tramite il modello DOM a oggetti, JavaScript può:

- Cambiare tutti gli elementi e attributi HTML di una pagina
- Cambiare tutti gli stili CSS
- Rimuovere elementi e attributi HTML
- Aggiungere elementi e attributi HTML
- Reagire a eventi nella pagina
- Creare nuovi eventi

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
    <table>
      <tr class="people">
        <td>John</td>
        <td>Doe</td>
        <td>Alaska</td>
      </tr>
    </table>
  </body>
</html>
```

test.html



DOM Tree

```
window.document.getElementById("header");
```

Si presentino nel dettaglio i **modelli di servizio del paradigma cloud.**

Ci sono 3 modelli di servizio del paradigma cloud:

- IaaS (Infrastructure as a Service)
- PaaS (Platform as a Service)
- SaaS (Software as a Service)

IaaS (Infrastructure as a Service)

- Utente gestisce interamente la CPU, la memoria, lo storage, la rete, e le risorse di computazione aggiuntive.
- Utente in grado di installare ed eseguire codice generico incluso sistemi operativi e applicazioni.
- Utente non gestisce o controlla l'infrastruttura Cloud, mentre controlla sistemi operativi, storage e applicazioni installate.
- Offre risorse virtualizzate on demand.
- Fornisce diversi server con diversi sistemi operativi.

PaaS (Platform as a Service)

- Utente installa e crea applicazioni sviluppate tramite linguaggi di programmazione, librerie, tools, e servizi supportati dal provider.
- Utente mantiene il controllo sulla piattaforma installata.
- Utente non gestisce l'infrastruttura sottostante che include la rete, i sistemi operativi, il server o lo storage.
- Utente mantiene il controllo delle applicazioni installate e delle configurazioni dell'ambiente per hosting di applicazioni.
- Piattaforma installata sull'infrastruttura dove gli sviluppatori creano e installano applicazioni.
- Livello di astrazione più alto che rende la cloud programmabile

SaaS (Software as a Service)

- Utente utilizza le applicazioni di un cloud provider che sono eseguite sull'infrastruttura cloud.
- Il cliente non gestisce o controlla l'infrastruttura cloud, incluso la rete, il server, il sistema operativo, lo storage o anche le funzionalità specifiche dell'applicazione.

Nell'ambito delle infrastrutture cloud, si discutano almeno tre delle regole della cloudonomics.

Le 10 leggi della **Cloudonomics:**

1. Utility services cost less even though they cost more = Il costo per unità di tempo è maggiore. **Accesso on demand alle utility riduce il costo totale**.
2. On-demand trumps forecasting = **Capacità di allocazione e deallocazione quasi istantanea**. Previsioni spesso sbagliate, capacità di reagire prontamente permette grande guadagno.
3. The peak of the sum is never greater than the sum of the peaks = **La quantità di risorse è la somma dei picchi**. La cloud installa meno risorse (riallocazione delle risorse).
4. Aggregate demand is smoother than individual = **L'aggregazione di richieste da diversi clienti tende a ridurre le variazioni.** Cloud ottiene miglior efficienza.
5. Average unit costs are reduced by distributing fixed costs over more units of output = **Costi fissi sono maggiormente distribuiti.** Diminuisce il costo per unità in diversi ambiti: storage, bandwidth, etc.
6. Superiority in numbers is the most important factor in the result of a combat = **Superiorità numerica permette di vincere le battaglie**. Attacco DoS più difficile nella cloud.
7. Space-time is a continuum = **Cloud scalability permette decision making più rapido**.
8. Dispersion is the inverse square of latency = **Ridurre la latenza è fondamentale** per molte applicazioni. Più semplice in cloud.
9. Don't put all your eggs in one basket = Maggior affidabilità. **Replica su data center distribuiti**.
10. An object at rest tends to stay at rest = **I data center** aziendali sono installati nella sede dell'azienda. Nella cloud **sono installati dove è più vantaggioso**.

Nell'ambito dei servizi REST, si discutano in dettaglio le **operazioni CRUD**.

CRUD sta per **C**reate, **R**ead, **U**pdate, e **D**elete, che sono quattro operazioni di database primitive. A prima vista, queste operazioni si associano bene ai verbi HTTP usati più frequentemente in REST:

- Create → POST: create a sub resource. Utilizzato per supportare la creazione di una risorsa figlio, ma può anche modificare lo stato sottostante di un sistema.
- Read → GET: retrieve the current state of the resource. Recupera una rappresentazione di una risorsa, ma con semantica aggiuntiva disponibile.
- Update → PUT: initialize or update the state of a resource at the given URI. Aggiorna una risorsa usando una rappresentazione completa. Può essere utilizzato anche per creare una risorsa.
- Delete → DELETE: clear a resource, after the URI is no longer valid. Elimina una risorsa.

Con la PUT il **client** decide il nome della risorsa, con la POST il **server** decide il nome della risorsa.

Nell'ambito delle architetture REST si discutano le principali **differenze tra le operazioni POST e PUT**. Si presenti un **esempio** per ognuna delle operazioni.

POST vs PUT:

- POST usata per creare risorse subordinate, PUT usata per creare/modificare risorse
- Con la PUT il **client** decide il nome della risorsa.
- Con la POST il **server** decide il nome della risorsa per assicurare che l'id sia univoco.

Esempi:

- PUT /esami/{id}
- POST /esami

Si discutano in dettaglio **localstorage** e **sessionstorage,** presentando le principali **differenze con i cookie**.

Ci sono due tipi di **offline storage**:

- **session storage** = storage di sessione. Permette di memorizzare dati specifici a una finestra/tab. Permette di **isolare informazioni in ogni finestra**, se l'utente visita lo stesso sito in due finestre diverse, ogni finestra avrà il suo session storage individuale con dati separati e distinti. **Non è persistente**. Viene mantenuto solo per la durata della sessione utente in uno specifico sito, viene mantenuto per il tempo in cui la finestra/tab del browser è aperta e visualizza il sito.
- **local storage** = A differenza del session storage, il local storage permette di salvare dati persistenti sul computer dell'utente, attraverso il browser. Quando l'utente rivisita il sito successivamente, ogni dato salvato nel localstorage può essere **recuperato**.

Dati memorizzati come coppie chiave-valore (simili ai cookie).

Differenza tra localstorage e cookie. I localstorage possono essere scambiati a una prima occhiata per HTTP cookie ma ci sono alcune differenze chiave:

- Cookie sono letti **lato server**, mentre local storage sono disponibili solo **lato client.**
- Se l'esecuzione del codice lato server dipende sul valore di alcune variabili, i cookie sono la soluzione ideale. I cookie inviati con ogni richiesta al server portano **overhead sostanziali** in termini di banda usata.
- Local storage risiede sull'hard disk dell'utente, e viene letto a **zero costo**.
- Local storage fornisce molto spazio di memorizzazione, i Cookie memorizzano fino a 4KB di informazione mentre il Local storage memorizza fino a 5MB di informazione.

Si discuta in dettaglio il ruolo dell'oggetto **XMLHttpRequest** presentando un esempio del suo utilizzo.

XMLHttpRequest Object = usato per scambiare dati con il server in background.
Permette di:

- Aggiornare una pagina senza ricaricarla
- Richiedere i dati dal server **dopo** che la pagina è stata caricata
- Ricevere i dati dal server dopo che la pagina è stata caricata
- Inviare i dati al server in background xmlhttp = new XMLHttpRequest();

Nonostante il nome, tramite questo oggetto è possibile ricevere dati anche in **formati diversi** dall'XML, come il **JSON** o il formato binario.
XML Parser = converte un documento XML in un oggetto XML DOM, manipolabile con javascript.

```
function reqListener () {
  console.log(this.responseText);
}

var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://www.example.org/example.txt");
oReq.send();
```

DUBBI
JavaScript permette accesso e modifica di oggetti tramite:

- Oggetti basati su proprietà e funzioni…VERO
- Eventi scatenati da azioni degli utenti…VERO
- Formule booleane…VERO (cosa si intende?)

Emulazione vs Simulazione vs Virtualizzazione:

- Emulazione è la tecnica più costosa…VERO (il mio dubbio è se la simulazione è in realtà la tecnica più costosa)

Il modello di deployment cloud pubblico

- Fornisce prestazioni superiori a un modello di deployment cloud privato…VERO
- È equivalente a un modello di deployment cloud community…VERO

Un'applicazione REST è

- Distribuisce servizi con interfaccia di accesso custom…FALSO

L'architettura REST

- Distribuisco servizi con interfacce di accesso ad hoc…FALSO

In un RESTful web Service

- Le risorse vengono identificate attraverso file XML…VERO
- Le risorse vengono accedute attraverso un'interfaccia uniforme basata su HTTP operation…VERO

L’oggetto XMLHttpRequest

- È definito lato client…VERO