Introduzione HTML

Struttura di una pagina

Document Object Model HTML

Il DOM ci consente di navigare ed operare sulla pagina e sugli elementi che la compongono a tempo di esecuzione usando **javascript**; questo aspetto ci da un riferimento su come sia possibile intervenire sul documento per "modificarlo" anche a tempo di esecuzione.

Vediamo un elemento radice, ed ogni nodo in questo albero è un elemento che abbiamo visto in precedenza. All'interno di ogni elemento possono essere posti altri elementi: ad esempio nel <body> è possibile inserire degli Headers <h1> o testo.

Elementi basilari

Gli heading possono rappresentare diversi livelli di intestazione. Un elemento importante da notare in questa immagine è il fatto che per ogni elemento possiamo introdurre la possibilità di associare (a quell'elemento) un **identificatore**. Possiamo poi (usando javascript) operare su quell'elemento **anche a tempo di esecuzione**.

Abbiamo anche altri elementi:

Possiamo anche creare delle tabelle:

Liste:

Blocchi: in HTML un block è un elemento che occupa una linea, ovvero dall'inizio (in alto) della pagina fino alla fine (in basso) o fino a quanto possibile; quindi se aggiungiamo diversi blocchi in successione questi vengono disposti verticalmente.

Span: gli span sono come i blocchi, ma vengono disposti orizzontalmente.

Forms: abbiamo già usato i form per raccogliere input che vengono inviati al server; infatti servono proprio a questo.

Fieldset: ci permette di raggruppare degli elementi, usando il tag:

Ad esempio possiamo costruire un form dove alcuni elementi vengono raggruppati grazie al fieldset:

Renderizzando la paggina otterremo:

> Quindi il Form è uno, ma abbiamo diversi elementi raggruppati.

Introduzione Javascript

Javascript è un linguaggio di scripting; attraverso javascript possiamo fare delle manipolazioni a tempo di esecuzione. Attraverso js potremmo anche effettuare delle modifiche pericolose: essendo js dello script recuperato dal server, è un codice che garantisce si flessibilità, ma potrebbe anche introdurre dei problemi di sicurezza; dobbiamo quindi evitare che uno script possa effettuare delle operazioni dannose sul client.

JS viene usato per intervenire a tempo di esecuzione sul browser, e quindi sugli elementi che il browser ospita. Possiamo sostituire il testo di una pagina con dell'altro testo, inserire del testo dove prima non era previsto, oppure cambiare il font/colore di un dato testo. Possiamo fare che una componente JS vada a recuperare un elemento di una pagina, per poterli utilizzare successivamente.

Abbiamo parlato della possibilità di manipolare il documento e per operare sul DOM (documento che rappresenta la pagina HTML all'interno del browser) possiamo usare l'oggetto predefinito **document** ed a partire da questo oggetto possiamo usarlo per invocare un metodo che ci permette di recuperare un oggetto dalla pagina HTML:

```
document.getElementById('demo').innerHTML = 'Hello Javascript'
```

con quest istruzione recuperiamo l'elemento della pagina HTML attraverso il suo nome che avevamo previsto all'interno del tag. Oltre a recuperare l'elemento, grazie ad `.innerHTML = ...` stiamo inserendo una stringa con valore `"Hello Javascriot"`.

Possiamo anche recuperare un elemento immagine con: `document.getElementById('myImage').src='some_pic.jpg'` invece di usare la source (.src) prevista nel codice HTML dell'elemento, andiamo a modificarla con la stringa 'some_pic.jpg'.

Possiamo nascondere un elemento con `document.getElementById('demo').style.display="none"`.

Semplice script

Attraverso il tag **<script> e </script>** possiamo specificare uno script di tipo MIME `text/javascript` e collocare lo script direttamente all'interno del body.

Embedding JavaScript

Non è detto che il codice debba essere inserito all'interno del documento HTML, ma è possibile recuperare del codice JS usando un file separato ed andando a recuperare il file come risorsa addizionale:

Con l'attributo **src="..."** specifichiamo il nome del file (assumiamo che la risorsa sia presente sullo stesso server web della pagina involucro HTML). All'atto dell'interazione browser-server il client scarica il file js e lo esegue.

🏁 1:17

Alert(), confirm() e prompt()

Possiamo inserire delle interazioni tra utente e browser/pagina web con i metodi:

Identificatori

JS è un linguaggio che ci consente di eseguire del codice all'interno del browser; di conseguenza si prevedono degli identificatori simili a quelli presenti in altri linguaggi. Questi identificatori devono iniziare con una lettera, simbolo '$' o '_'. Non possiamo usare degli identificatori riservati al linguaggio.

Tipi di dati

JS non è un linguaggio tipizzato, di conseguenza **una variabile non ha un tipo staticamente definito**. Il tipo della variabile viene assegnato a tempo di esecuzione, di conseguenza i tipi di dati sono variabili:

All'interno di questo script la variabile x prima è di tipo intero, poi diventa di tipo stringa.

Array

Gli arrays sono simili a quelli presenti in Java, infatti anche in questo caso definiamo un array partendo dall'esplicitazione degli elementi costituenti l'array stesso: `var languages = ["Java", "C", JavaScript];` Possiamo anche avere un array **omogeneo**, ovvero contenente **tipi diversi** (ad esempio contiene stringhe, interi, oggetti, ecc.)

Oggetti

Gli oggetti in JS sono **collezioni di proprietà** e possono avere anche delle funzioni (simili ai metodi):

> Possiamo definire un oggetto all'atto di assegnazione dello stesso ad una variabile

Funzioni

Le funzioni in JS vengono dichiarate in questo modo:

Siccome non esiste il concetto di classe, possiamo definire le funzioni liberamente. Non dobbiamo prevedere ne il **tipo di parametro** ne il **valore di ritorno**.

In JS è possibile gestire i parametri di una funzione come un array, invece di specificarli come nell'immagine precedente.

> Questa funzione addiziona tutti i parametri passati e ritorna la somma

Se usiamo **arguments** possiamo quindi gestire i parametri come un array, selezionando uno specifico parametro usando `arguments[i]`. Di conseguenza possiamo avere un **numero indefinito di parametri**.

Funzioni incorporate in JS

- **eval(expression)** L'espressione può essere sia numerica che la visualizzazione di un alert.
  - eval("3+4") ritorna 7
  - eval("alert('Hello')") chiama la funzione alert('hello')
- **ifFinite(x)** verifica se il numero passato è finito
- **isNaN(x)** verifica se il numero passato è un numero.
- **parseInt(s)**
- **parseInt(s,radix)**: convertiamo una stringa che contiene nella parte iniziale un numero:
  - parseInt("3 chances") ritorna 3
  - parseInt(" 5 alive") ritorna 5
  - parseInt("come stai fra?") ritorna NaN
  - parseInt("17" , 8) ritorna 15 (boh)

Eventi

Nei documenti HTML possiamo catturare degli eventi causati dall'interazione dell'utente con il browser. Ad esempio un evento potrebbe essere il click su di un link, o su di un button.

Questi eventi possono essere catturati ed associati ad handler, un po' come accade nella programmazione delle interfacce grafiche di java. Se possiamo caratterizzare degli eventi, possiamo associarvi un **handler** in modo da eseguire del codice quando questo evento avviene.

All'evento **onClick** viene associato l'handler **alert()** (che è una funzione js)

Event Handlers JS

Esempio onClick

Vediamo che viene usata la funzione **warnUser()** (JS) come reazione ad un evento.

Creazione degli oggetti

Esistono diversi modi per creare oggetti:

Prima opzione

Possiamo usare l'operatore **new Object()** per creare un nuovo oggetto. Possiamo poi associare alla variabile che referenzia l'oggetto delle **proprietà**.

Possiamo anche definire delle funzioni dell'oggetto sempre tramite la variabile che lo referenzia con `person.sayHi = function(){...}`

Seconda opzione - Literal Notation

Questa notazione ci consente anche di capire i legami tra JS e servizi REST:

firstname è una proprietà a cui assegnamo un valore attraverso ':', così come le altre proprietà. Abbiamo inoltre la funzione **sayHi** che come nelle proprietà assegnamo un valore **funzione** con ':'.

Possiamo anche avere delle **proprietà innestate:**

Costruttore dell'oggetto

Per poter definire un template da cui creare più istanze di oggetti (una sorta di classe) possiamo scrivere una funzione vista come un costruttore:

Implicitamente assumiamo che, usando **this.**, queste variabile saranno gestite in maniera separata in istanze diverse degli oggetti. Di conseguenza possiamo usare la funzione come oggetto, invocando **new Person()**. Andando ad invocare **sayHi** sui diversi oggetti otterremo "risultati" diversi, proprio perchè questi sono **due istanze diverse dello stesso oggetto** (template di oggetto).

Oggetti incorportati in JS

Gestione delle eccezioni in JS

Le eccezioni possono essere gestite in modo simile a quello già visto in Java, usando la struttura di controllo **try/catch**

All'interno della sezione script prevediamo il costrutto try/catch che "controlla" un elemento di HTML tramite il suo id.

L'oggetto XMLHttpRequest

Possiamo usare all'interno di programmi JS un oggetto particolare **XMLHttpRequest** che funziona come il client che abbiamo visto negli esempi dell'esercitazione 10, ovvero possiamo costruire dei messaggi di richeista HTTP direttamente all'interno del client (programma).

00:13