

Linguaggio di **scripting** (interpretato e debolmente tipato) orientato agli **eventi** e (pseudo-orientato) agli oggetti. L'orientamento agli oggetti non è completo in quanto non sono supportate l'**ereditarietà** e il **polimorfismo**.

L'interprete è il "Javasctipt engine" che è incluso nel **browser** usato (firefox usa "spidermonkey").

1 Tag <script>

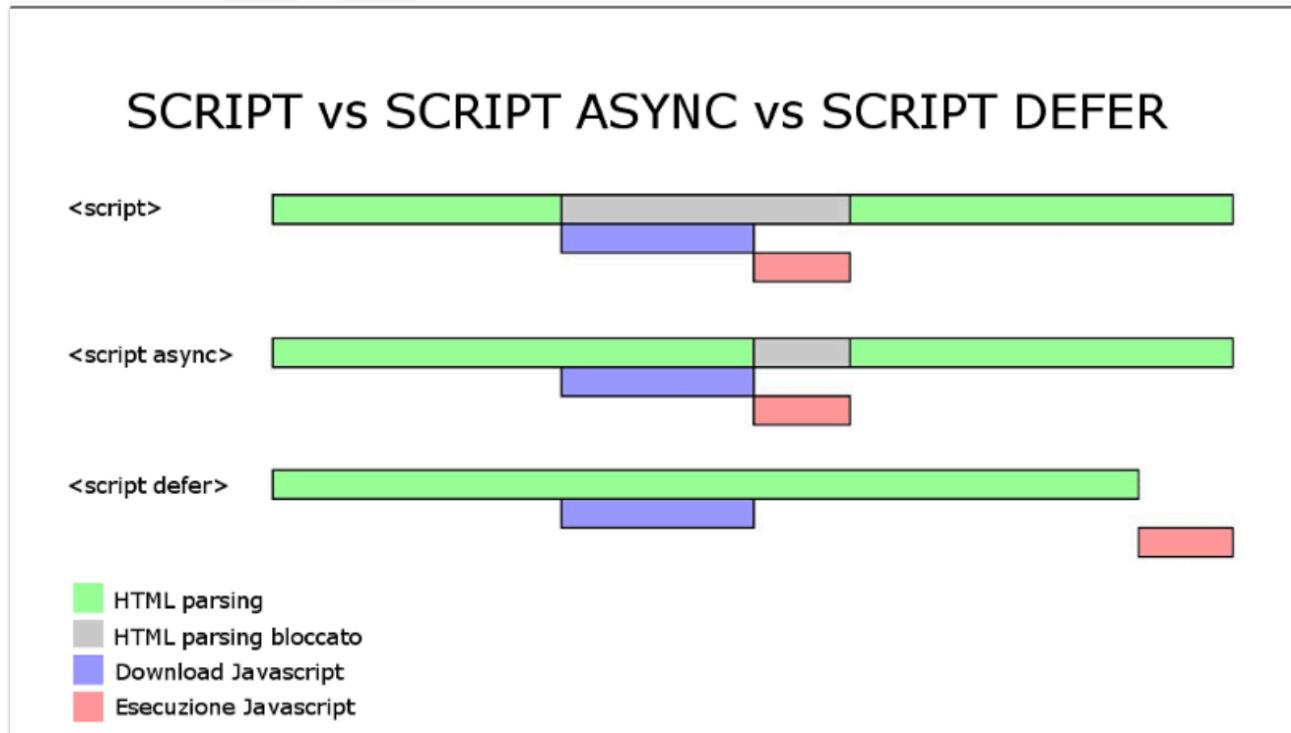
Si può inserire in pagine html usando il tag `<script></script>` che prevede 4 attributi>

- `type` : descrive il **mime type**. Eg: `type="text/javascript"`
- `src` : indica il **percorso** di un file javascript. Eg: `src="codice/menu.js"`
- `defer` : attributo booleano che indica al browser **caricare lo script in background** ed **eseguirlo** solamente quando tutto l'**HTML è stato caricato**;
- `async` : lo script è completamente **indipendente**, viene scaricato in background ed **eseguito quando pronto**. Script che si trovano in posizioni diverse del documento possono quindi essere eseguiti in ordine differente.

1.1 Problema del caricamento

La presenza di uno script nell'HTML, blocca la **costruzione del DOM**: il browser deve prima scaricare ed eseguire lo script e successivamente continuare a costruire il DOM. Sorgono 2 problemi:

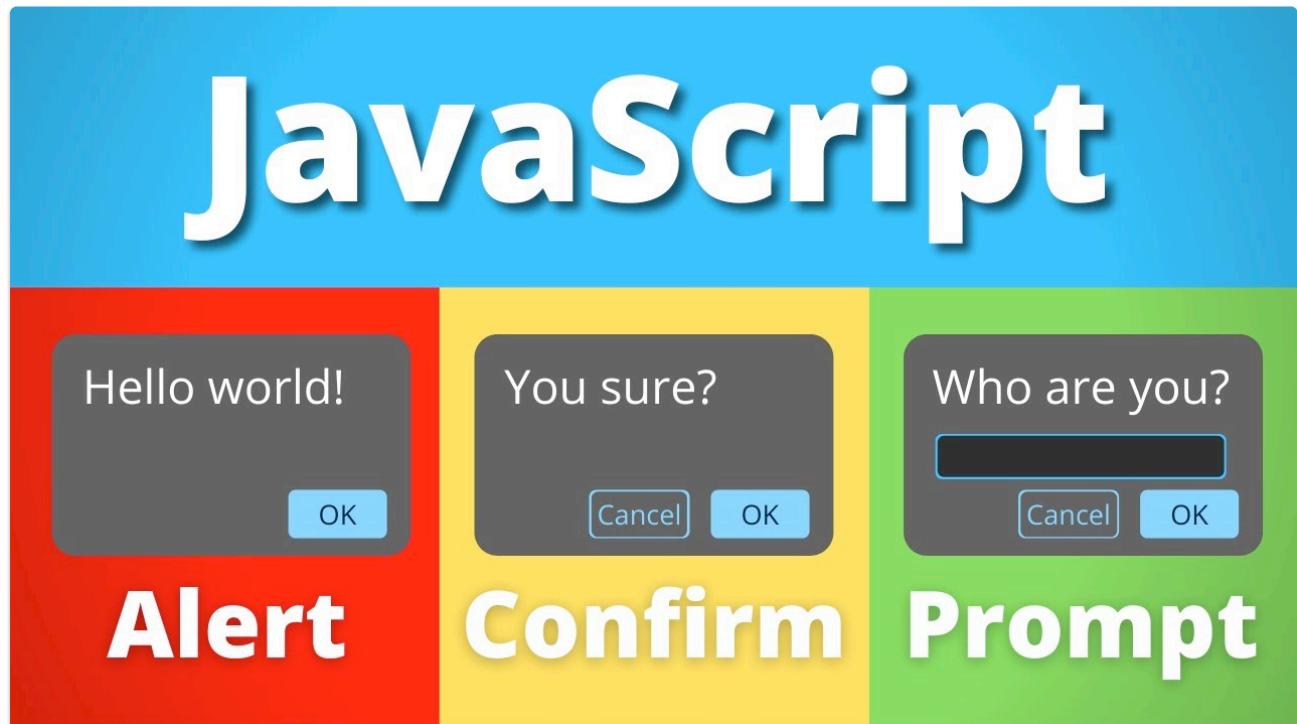
- Script non possono vedere elementi ancora non presenti nel DOM;
- Uno script pesante posizionato in cima ad una pagina HTML può **bloccare** il caricamento della pagina;
Una soluzione banale è quella di **posizionare lo script al fondo della pagina**, altrimenti (e meglio) bisogna usare gli attributi `async` e `defer`.



2 Elementi del linguaggio

2.1 Interazione con l'utente

- `alert(<stringa>)` : mostra una finestra di dialogo all'utente ed un pulsante per chiuderla;
- `confirm(<stringa>)` : mostra una finestra di dialogo all'utente e 2 pulsanti, uno per confermare l'azione (restituisce `true`), l'altro per annullarla (restituisce `false`). Un esempio è la finestra di accettazione dei cookie.
- `prompt(<stringa>, default)` : Mostra una finestra di dialogo con un messaggio, una linea testuale di input e due bottoni uno per confermare l'azione l'altro per annullare l'azione. Restituisce la stringa inserita dall'utente.



2.2 Variabili

	Scope	Ridichiarabilità	Aggiornabilità	Inizializzazione/Hoisting
<code>var</code>	globale, locale, funzione	SI	SI	Undefined
<code>let</code>	locale	NO	SI	Not Initialized
<code>const</code>	locale	NO	NO	Not Initialized

Le variabili `var` presentano una debolezza relativa al fatto che possono essere ridichiarate ed aggiornate:

```
var greeter = "hey hi";
var times = 4;
if (times > 3)
    var greeter = "say Hello instead";
console.log(greeter) // "say Hello instead"
```

È possibile definire una nuova variabile, con lo stesso identificatore di quella precedente, perdendo di fatto valore e riferimento. Questo problema non si presenta con `let` e `const`, un'eventuale ri-dichiarazione genera un [errore](#).

```
let greeting = "say Hi";
let times = 4;
if (times > 3) {
    let hello = "say Hello instead";
    console.log(hello); // "say Hello instead"
```

```
}
```

```
console.log(hello) // hello is not defined
```

2.2.1 Hoisting

Meccanismo che **sposta la dichiarazione di variabili** e funzioni all'inizio del loro **ambito di visibilità**, prima dell'esecuzione del codice:

```
console.log (greeter);  
var greeter = "say hello"
```

È interpretato come:

```
var greeter;  
console.log(greeter); // greeter is undefined  
greeter = "say hello"
```

Le variabili `Not Initialized` non sono inizializzate e pertanto se lette generano un errore.

2.3 Funzioni

```
function nome_funzione (argomenti){  
    corpo ....  
}
```

Alla chiamata della funzione, il numero dei parametri passato può **differire** dal numero presente nella dichiarazione:

- passiamo un numero **inferiore** di parametri: i mancanti avranno valore `undefined`;
- passiamo un numero **superiore** di parametri: i parametri extra sono ignorati;

2.3.1 Funzioni anonime

Il nome della funzione non è obbligatorio, in caso di sua assenza si parla di "**funzioni anonime**".

Vantaggi

- Riduzione inquinamento namespace: non vengono create variabili globali e pertanto il namespace rimane pulito;
- Codice modulare: creare moduli di codice indipendenti che possono essere facilmente riutilizzati.
- Assegnazione dinamica: possono essere assegnate dinamicamente a variabili e invocate in base a determinate condizioni o eventi.

Nonostante manchi il nome della funzione, **assegnandola ad una variabile** è possibile invocarla in qualunque punto del codice usando la variabile stessa.

```
let saluto = function() {  
    console.log("Ciao, benvenuto!");  
};  
saluto(); //Stampa: Ciao, benvenuto!
```

Un altro utilizzo è relativo alla creazione di uno **scope isolato** per evitare conflitti con variabili globali. Sono dette IIFE (Immediately Invoked Function Expression), funzioni anonime invocate immediatamente dopo la loro definizione:

```
(function() {
    let nome = "Alice";
    console.log(`Ciao, ${nome}!`);
})();
// Stampa: Ciao, Alice!
```

2.3.1.1 Arrow functions

Sono particolari funzioni anonime create con l'obiettivo di semplificare ed abbreviare la sintassi:

Arrow function

```
let func = (arg1, arg2, ..., argN) => {body};
```

```
var somma = function(x, y) {
    return x + y;
};
```

Puó essere riscritta come:

```
var somma = (x, y) => { return x + y};
```

Se il corpo é formato dalla sola istruzione di `return`, é possibile omettere la keyword `return`:

```
var somma = (x, y) => x + y;
```

2.4 Oggetti

JavaScript non è un classico linguaggio orientato agli oggetti in quanto non supporta l'**ereditarietà** e il **polimorfismo**.

Un oggetto é una **collezione di valori** sotto un unico nome (**proprietá** o named values), simili alle strutture in C. L'accesso alla proprietà é effettuato mediante il `.`:

```
immagine.altezza  
immagine.larghezza
```

Oppure considerando l'oggetto come un **array associativo**:

```
var p = {x:3, y:4};  
vx = p["x"];
```

Un nuovo oggetto puó essere creato mediante `new Object()`. L'oggetto creato puó anche avere dei metodi:

- per chiamare il metodo si usa la dot-notation `.` sull'oggetto che contiene il metodo;
- per accedere agli attributi del metodo si usa la keyword `this`.

2.4.1 Array

Sono oggetti eterogenei e non occorre specificare la dimensione durante la creazione:

```
var array = new Array()
```

Le posizioni non assegnate dell'array presentano valore `undefined`.

Method	Description	Returns
<code>concat(<otherArray>)</code>	Concatenates the contents of the array with the array specified by the argument. Multiple arrays can be specified.	<code>Array</code>
<code>join(<separator>)</code>	Joins all of the elements in the array to form a string. The argument specifies the character used to delimit the items.	<code>string</code>
<code>pop()</code>	Treats an array like a stack, and removes and returns the last item in the array.	<code>object</code>
<code>push(<item>)</code>	Treats an array like a stack, and appends the specified item to the array.	<code>void</code>
<code>reverse()</code>	Reverses the order of the items in the array in place.	<code>Array</code>
<code>shift()</code>	Like <code>pop</code> , but operates on the first element in the array.	<code>object</code>
<code>slice(<start>,<end>)</code>	Returns a sub-array.	<code>Array</code>
<code>sort()</code>	Sorts the items in the array in place.	<code>Array</code>
<code>unshift(<item>)</code>	Like <code>push</code> , but inserts the new element at the start of the array.	<code>void</code>

3 DOM - Document Object Model

Il DOM è un **modello ad oggetti** (insieme di oggetti che rappresentano gli elementi del documento HTML) che fornisce l'accesso agli elementi del documento HTML, permettendo di gestire e manipolare il documento HTML da Javascript. Rappresenta la **connessione tra JS e HTML**. Il DOM rispetta la gerarchia degli elementi HTML, ossia la gerarchia ad albero.

Tale modello viene generato dal browser quando effettua il parsing del file HTML per visualizzarlo. Ogni elemento HTML diventa un oggetto del DOM (e quindi un oggetto javascript), si parla di "**HTML Element object**". L'oggetto radice (gateway) è `Document` (raggiungibile tramite la variabile `document`), rappresenta l'intero documento HTML e consente di accedere a tutte le funzionalità del DOM.

3.1 Alcuni metodi ed oggetti del DOM

- `writeln`: metodo dell'oggetto `Document` che consente di scrivere HTML nel documento corrente
- `documentURL`: restituisce l'informazione relativa alla URL del documento corrente
- `document.location`: restituisce un oggetto `Location` che fornisce informazioni dettagliate sull'indirizzo del documento corrente e consente di navigare verso altri documenti

Location Object Properties

Property	Description
<code>hash</code>	Sets or returns the anchor part (#) of a URL
<code>host</code>	Sets or returns the hostname and port number of a URL
<code>hostname</code>	Sets or returns the hostname of a URL
<code>href</code>	Sets or returns the entire URL
<code>origin</code>	Returns the protocol, hostname and port number of a URL
<code>pathname</code>	Sets or returns the path name of a URL
<code>port</code>	Sets or returns the port number of a URL
<code>protocol</code>	Sets or returns the protocol of a URL
<code>search</code>	Sets or returns the querystring part of a URL

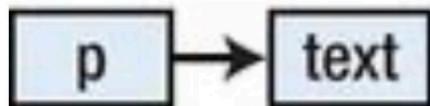
Location Object Methods

Method	Description
<code>assign()</code>	Loads a new document
<code>reload()</code>	Reloads the current document
<code>replace()</code>	Replaces the current document with a new one

- `window`: oggetto che rappresenta una finestra aperta nel browser. Il browser crea automaticamente un oggetto finestra per ogni `iframe` presente nel documento

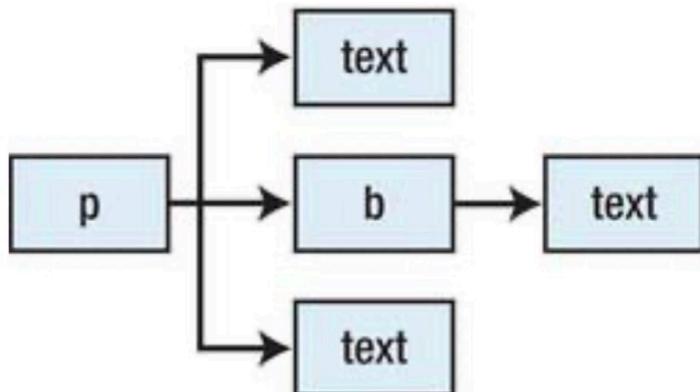
3.1.1 Text element e modifica del testo

Il testo contenuto in un elemento HTML, è visto in JS come un elemento di tipo Text, figlio dell'elemento che contiene il testo.



Per modificare un testo bisogna quindi accedere al primo figlio (`firstChild`) dell'elemento che contiene il testo (`<p>`).

```
...
<p id="textblock" class="fruit numbers" data-fruit="apple" data-sentiment="like">
    There are lots of different kinds of fruit - there are over <b>500</b> varieties
    of banana alone. By the time we add the countless types of apples, oranges,
    and other well-known fruit, we are faced with thousands of choices.
</p>
...
```



Metodi per la modifica del testo:

Member	Description	Returns
appendData(<string>)	Appends the specified string to the end of the block of text	void
data	Gets or sets the text	string
deleteData(<offset>, <count>)	Removes the text from the string; the first number is the offset, and the second is the number of characters to remove	void
length	Returns the number of characters	number
splitText(<number>)	Splits the existing Text element into two at the specified offset (see the “Inserting an Element into a Text Block” section later in this chapter for a demonstration of this method)	Text
substringData(<offset>, <count>)	Returns a substring from the text	string

3.1.2 Modifica del DOM

È possibile modificare l'intera struttura HTML creando, eliminando o [modificando](#) direttamente elementi nel [DOM](#):

- createElement(<tag>) : crea un nuovo elemento di tipo <tag>
- createTextNode(<text>) : crea un nuovo oggetto Text con il contenuto <text>

Creare un nuovo oggetto, bisogna aggiungerlo al DOM:

- appendChild(HTMLElement) : aggiunge l'elemento HTML come figlio dell'elemento su cui il metodo è stato invocato;
- insertBefore(<newElem>, <childElem>) : Inserisce il nuovo elemento newElem prima dell'elemento childElem

ilNodo. insertBefore(nuovoFiglio, vecchioFiglio)



- replaceChild(newHtmlElement, oldHtmlElement) : Inserisce il nuovo elemento newElem prima dell'elemento childElem
- removeChild(HtmlElement) : cancella un figlio. Il metodo restituisce il nodo rimosso.

- `cloneNode(boolean)` : effettua la copia di un nodo particolare. Il valore booleano indica se i figli del nodo devono essere copiati (`true`) oppure deve essere copiato solo il nodo stesso senza figli (`false`)
- `innerHTML` : Contiene l'HTML del figlio dell'elemento su cui è stato chiamato e consente pertanto la sua modifica;
- `outerHTML` : Contiene l'HTML del nodo a cui appartiene e di tutti i suoi figli e consente la sua modifica

3.1.2.1 CSS API

Il DOM fornisce un [API](#) per poter [controllare lo stile CSS con JavaScript](#). Per modificare lo stile bisogna per prima cosa ottenere un [riferimento all'elemento](#) da modificare e successivamente utilizzare la proprietà `style` per ottenere un oggetto "Style", sul quale è possibile impostare i classici attributi css.

3.1.3 Selezione ed accesso degli elementi del DOM

Esistono vari modi per [selezionare](#) ed accedere agli elementi (`HTMLElement`) del DOM:

1. Usando le [proprietà di document](#);
2. Usando `document` come [array](#) di elementi
3. Usando metodi per la [ricerca](#) di elementi
4. Navigando nel DOM Tree

Una volta selezionato un elemento, possiamo accedere ai suoi attributi e metodi.

Property	Description	Returns
<code>checked</code>	Gets or sets the presence of the checked attribute	<code>boolean</code>
<code>classList</code>	Gets or sets the list of classes to which the element belongs	<code>DOMTokenList</code>
<code>className</code>	Gets or sets the list of classes to which the element belongs	<code>string</code>
<code>dir</code>	Gets or sets the value of the dir attribute	<code>string</code>
<code>disabled</code>	Gets or sets the presence of the disabled attribute	<code>boolean</code>
<code>hidden</code>	Gets or sets the presence of the hidden attribute	<code>boolean</code>
<code>id</code>	Gets or sets the value of the id attribute	<code>string</code>
<code>lang</code>	Gets or sets the value of the lang attribute	<code>string</code>
<code>spellcheck</code>	Gets or sets the presence of the spellcheck attribute	<code>boolean</code>
<code>tabIndex</code>	Gets or sets the value of the tabindex attribute	<code>number</code>
<code>tagName</code>	Returns the tag name (indicating the element type)	<code>string</code>
<code>title</code>	Gets or sets the value of the title attribute	<code>string</code>

- `attributes` : restituisce un array di attributi (oggetto Attr) applicati all'elemento;
- `getAttribute(<name>)` : Restituisce il valore dell'attributo `<name>`
- `hasAttribute(<name>)`: Restituisce true se l'elemento ha l'attributo specificato
- `removeAttribute(<name>)`: Rimuove l'attributo specificato
- `setAttribute(<name> , <value>)`: Aggiunge l'attributo `<name>` con valore `<value>`

3.1.3.1 Accesso mediante le proprietà di document

Si può accedere agli elementi del DOM usando la **dot notation** sull'oggetto `document`.

Property	Description	Returns
<code>activeElement</code>	Returns an object representing the currently focused element	<code>HTMLElement</code>
<code>body</code>	Returns an object representing the <code>body</code> element	<code>HTMLElement</code>
<code>embeds</code> <code>plugins</code>	Returns objects representing all the <code>embed</code> elements	<code>HTMLCollection</code>
<code>forms</code>	Returns objects representing all the <code>form</code> elements	<code>HTMLCollection</code>
<code>head</code>	Returns an object representing the <code>head</code> element	<code>HTMLHeadElement</code>
<code>images</code>	Returns objects representing all the <code>img</code> elements	<code>HTMLCollection</code>
<code>links</code>	Returns objects representing all the <code>a</code> and <code>area</code> elements in the document that have <code>href</code> attributes	<code>HTMLCollection</code>
<code>scripts</code>	Returns objects representing all the <code>script</code> elements	<code>HTMLCollection</code>

Esempio

```
document.body : restituisce l'elemento (HTMLElement) body della pagina...
```

3.1.3.2 Accesso usando `document` come array

Document è visto come un **array associativo** che contiene gli elementi del documento HTML. La chiave di tali elementi nell'array è l' `id` dell'elemento HTML o l'attributo `name`.

Esempio

```
document['apple']
```

Restituisce gli elementi con `name=apple` o `id=apple` nello stesso ordine in cui vengono trovati all'interno della pagina HTML.

```
document.apple ha lo stesso significato di document['apple']
```

3.1.3.3 Accesso usando metodi per la ricerca di elementi

Proprietà	Descrizione	Return
getElementById(<id>)	elemento con l'id specificato	HTMLElement
getElementsByClassName(<class>)	elementi appartenenti alla classe specificata	HTMLElement[]
getElementsByName(<name>)	elementi con il "name" specificato	HTMLElement[]
getElementsByTagName(<tag>)	elementi del tipo specificato (tag)	HTMLElement[]
querySelector(<selector>)	primo elemento che viene selezionato dal Selettore CSS	HTMLElement
querySelectorAll(<selector>)	tutti gli elementi selezionati dal Selettore CSS	HTMLElement[]

I metodi di ricerca possono essere combinati tra loro.

3.1.4 Accesso mediante la navigazione nel DOM tree

Considerando il DOM come un albero, è possibile [navigare](#) fra i suoi elementi con i seguenti metodi:

Proprietà	Descrizione	Return
childNodes	insieme degli elementi figlio	HTMLElement[]
firstChild	il primo elemento figlio	HTMLElement
hasChildNodes()	vero se l'elemento corrente ha figli	boolean
lastChild	ultimo elemento figlio	HTMLElement
nextSibling	prossimo elemento adiacente dopo l'elemento corrente	HTMLElement
parentNode	elemento genitore	HTMLElement
previousSibling	elemento adiacente precedente l'elemento corrente	HTMLElement

4 Programmazione Event-Driven

Javascript è un linguaggio [event-driven](#): l'interprete javascript monitora costantemente un insieme di eventi (generati dall'utente e dal browser), con il quale è possibile interagire. Ad ogni evento è infatti associato un

gestore che consente associare automaticamente (al verificarsi di un evento), la chiamata di una **funzione js**.

Eventi in Javascript

- Abort
- Blur
- Change
- Click
- DblClick
- Error
- Focus
- KeyDown
- KeyPress
- KeyUp
- Load
- MouseDown
- MouseMove
- MouseOut
- MouseOver
- MouseUp
- Move
- Reset
- Resize
- Select
- Submit
- Unload

4.1 Come gestire gli eventi

1. Metodo Inline;
2. Metodo registration handler;
3. Event Listener

4.1.1 Metodo Inline

Il codice di gestione dell'evento viene inserito inline direttamente nell'**html**, come **attributo**:

```
<a id="link" href="#" onEvent="myFunction(); return false;">Where</a>
```

Dove `onEvent` rappresenta un generico evento (per esempio `onclick`).

4.1.2 Metodo Registration Handler

Si specifica tutto nello script js senza dover modificare il codice html.

```
var pElems = document.getElementsByTagName("p");
for (var i = 0; i < pElems.length; i++) {
    pElems[i].onmouseover = handleMouseEvent;
    pElems[i].onmouseout = handleMouseEvent;
}
function handleMouseEvent(e) {
    if (e.type == "mouseover") {
        e.target.style.background='white';
        e.target.style.color='black';
    } else {
        e.target.style.removeProperty('color');
        e.target.style.removeProperty('background');
    }
}
```

Il parametro `e` è un oggetto di tipo `Event`, creato dal browser al momento in cui viene scatenato l'`evento`.

Event e

Name	Description	Returns
<code>type</code>	The name of the event (e.g., <code>mouseover</code>).	<code>string</code>
<code>target</code>	The element at which the event is targeted.	<code>HTMLElement</code>
<code>currentTarget</code>	The element whose event listeners are currently being invoked.	<code>HTMLElement</code>
<code>eventPhase</code>	The phase in the event life cycle.	<code>number</code>
<code>bubbles</code>	Returns <code>true</code> if the event will bubble through the document, <code>false</code> otherwise.	<code>boolean</code>
<code>cancelable</code>	Returns <code>true</code> if the event has a default action that can be cancelled, <code>false</code> otherwise.	<code>boolean</code>
<code>timeStamp</code>	The time at which the event was created, or <code>0</code> if the time isn't available.	<code>string</code>
<code>stopPropagation()</code>	Halts the flow of the event through the element tree after the event listeners for the current element have been triggered.	<code>void</code>
<code>stopImmediatePropagation()</code>	Immediately halts the flow of the event through the element tree; untriggered event listeners for the current element will be ignored.	<code>void</code>
<code>preventDefault()</code>	Prevents the browser from performing the default action associated with the event.	<code>void</code>
<code>defaultPrevented</code>	Returns <code>true</code> if <code>preventDefault()</code> has been called.	<code>boolean</code>

Utilizzando questo approccio tuttavia, non è possibile assegnare più handlers ad uno stesso evento su un elemento.

4.1.3 Event Listeners

Il metodo `addEventListener` di `HTMLElement`, consente di assegnare più handlers allo stesso evento, superando il limite dei "registration handlers".

```
var pElems = document.getElementsByTagName("p");
for (var i = 0; i < pElems.length; i++) {
    pElems[i].addEventListener("mouseover", handleMouseOver);
    pElems[i].addEventListener("mouseout", handleMouseOut);
}
document.getElementById("pressme").onclick = function() {
    elm = document.getElementById("block2");
    elm.removeEventListener("mouseout", handleMouseOut);
}
```

Il metodo `removeEventListener` consente di rimuovere il listener di un determinato evento, rompendo l'associazione tra l'evento e la funzione.

4.2 Propagazione degli eventi: Event Bubbling

Fenomeno legato al DOM per cui, quando si genera un `evento`, questo si `propaga` ai suoi elementi genitori ed `antenati` all'interno del DOM tree, fino a raggiungere l'elemento `radice`. Si verifica per la maggior parte degli eventi ma non per tutti.

```

<style>
body * {
  margin: 10px;
  border: 1px solid blue;
}
</style>

<form onclick="alert('form')">FORM
<div onclick="alert('div')">DIV
  <p onclick="alert('p')">P</p>
</div>
</form>

```

- Nell'esempio qui a sinistra, cliccando sull'elemento p, ha come effetto l'esecuzione del gestore del evento click per:
 - L'elemento più interno
 - Per l'elemento div
 - Per l'elemento form
 - Per gli ulteriori eventuali elementi in cui form è innestato fino all'oggetto document

Per capire su quale elemento si è verificato l'evento, si può usare l'elemento `event.target`.

La propagazione dell'evento può essere interrotta mediante il metodo: `event.stopPropagation()`.

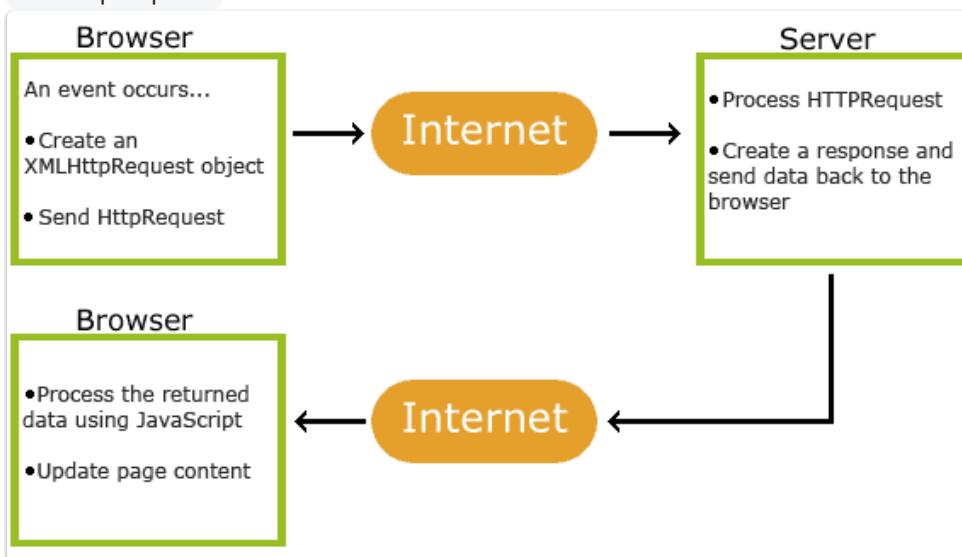
- Se un elemento ha associati più gestori per uno stesso evento e vogliamo impedire che tutti gestiscano l'evento propagato dobbiamo invocare il metodo **`event.stopImmediatePropagation()`** in uno di essi. Nota: quando su un elemento abbiamo più gestori per lo stesso evento questi vengono eseguiti nell'ordine con cui li abbiamo definiti.

Il blocco del bubbling dovrebbe essere effettuato solamente quando è realmente necessario in quanto è possibile andare in contro ad insidiosi bug difficili da identificare.

5 Ajax - Asynchronous JavaScript and XML

5.1 Come funziona Ajax

Si utilizza per il trasferimento di dati da server a browser in **background**. Per l'utilizzo si ricorre all'oggetto XMLHttpRequest .



1. Si verifica un evento JS;
2. Il gestore dell'evento crea un oggetto XMLHttpRequest ;
3. Viene inviata una richiesta al web server tramite i metodi dell'oggetto XMLHttpRequest ;
4. Il server processa la richiesta ed invia una risposta al client;
5. Il client genera un evento che può essere gestito in JS;
6. Viene aggiornato il contenuto della pagina.

Ajax sfrutta gli eventi `readystatechange` per informarci sullo stato di una richiesta.

5.2 Come usare Ajax

1. Creare un oggetto XMLHttpRequest: `let httpRequest = new XMLHttpRequest();`
2. Definire il gestore dell'evento `readystatechange` : `httpRequest.onreadystatechange = handleResponse;`
3. Specificare la tipologia di richiesta al server: `httpRequest.open("GET", "pagina.html");`
4. Definire i gestori degli eventi che si verificheranno dopo l'invio della richiesta al server (eg. `load` , `error` , `progress`);
5. Inviare la richiesta con `httpRequest.send()` ;

```
httprequest.onload = function() {
    // viene scatenato quando la risposta è stata scaricata del tutto
    alert(`Loaded: ${xhr.status} ${xhr.response}`);
};

httprequest.onerror = function() {
    // viene innescato solo se la richiesta non puo' essere eseguita
};

httprequest.onprogress = function(event) {
    // viene scatenato periodicamente
};

httprequest.onreadystatechange = function(event) {
    // viene scatenato al variare dello stato di XMLHttpRequest
};
```

5.3 GET o POST?

Quando usare POST:

- Aggiornamenti file o database;
 - Inviare grandi quantità di dati al server (POST non ha limiti sul quantitativo di dati da inviare);
 - Inviare dati inseriti dall'utente in un form;
- L'invio dei dati con POST è generalmente più **lento** dell'invio tramite GET.

Inoltre, quando si usa POST, è buona norma specificare il **formato dei dati** con il metodo

```
setRequestHeader():
```

- Quando costruiamo noi la stringa con le coppie chiave-valore:
`xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded")`
- Quando inviamo dati usando `JSON.stringify`:
`xhttp.setRequestHeader("Content-type", "application/json; charset=utf-8")`
- Quando vogliamo inviare un file, o preleviamo i dati da un form con new
`FormData(document.forms.modulo):`
`xhttp.setRequestHeader("Content-type", "multipart/form-data")`

È buona norma impostare anche il [formato della risposta](#) con la proprietà `responseType`:

Valore	Descrizione
text	La risposta è in formato testuale
document	La risposta è un documento XML o HTML
json	La risposta è un oggetto JSON risultante dal parsing di una stringa inviata dal server
arraybuffer	La risposta è un buffer di dati binary rappresentati come ArrayBuffer
blob	La risposta è un oggetto Blob, assimilabile ad un file

5.4 JSON

JSON - JavaScript Object Notation è un [formato testuale](#) per la strutturazione e lo scambio dei dati in applicazioni client-server. Sfrutta la [sintassi js](#):

```
var JSON = {  
    proprietA1: 'Valore',  
    proprietA2: 'Valore',  
    proprietAN: 'Valore'  
}'''  
Sono coppie di proprietà/valori separate da virgole... raggruppate in un **unico  
oggetto** racchiuso tra graffe. JSON ammette solamente valori semplici ed atomici come  
stringhe, numeri, array, boolean, null... Esempio:  
'''json  
{  
    "home": "Html.it",  
    "link": "http://www.html.it",  
    "argomento": "Standard del web",  
    "aree": [  
        {  
            "area": "CSS",  
            "url": "http://css.html.it"  
        },  
        {  
            "area": "Basic",  
            "url": "http://basic.html.it"  
        }  
    ]  
}
```

Per accedere agli oggetti JSON tramite js si usa la notazione tradizionale:

```
oggetto.home; // Html.it  
oggetto.link; // http://www.html.it  
var aree = oggetto.aree; // Array;  
for(var i = 0; i < aree.length; i += 1) {  
    var oggettoArea = aree[i];  
    var area = oggettoArea.area;
```

```
var url = oggettoArea.url;  
}
```

5.5 API

Sono **interfacce** che permettono alle applicazioni di **interagire con altre applicazioni**. le più diffuse sono le **REST** (Representational State Transfer) API che si basano sui metodi **HTTP standard** e sono **stateless**, ovvero ogni richiesta è indipendente dall'altra e quindi ogni richiesta deve includere tutte le informazioni necessarie per elaborarla.

L'accesso alle API generalmente è fornito mediante un **endpoint**, un **URL** e viene usato JSON o XML come formato per lo scambio dei dati.

4 esempi

- Aggiornamento pagina con dati prelevati in background

