Copyright (c) 2010 Fabio Proietti

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Authors and contributors
Fabio Proietti

Feedback
Please direct any comments or suggestions about this document to
fabio.proietti AT istruzione DOT it

Publication date and version

last modified 2012-03-24

Programmazione client side

Per poter continuare è necessario saper rispondere alle seguenti domande:
Che cosa è un programma?
Chi scrive il programma?
Chi esegue il programma?
Chi utilizza il programma?
Come funziona il modello client/server?

Stili di programmazione

- Programmazione *imperativa strutturata*: i linguaggi di programmazione orientati a questo modello sono costituiti da istruzioni che sono dei comandi imperativi per l'elaboratore (relativamente all'ambiente di esecuzione del programma). La programmazione strutturata è chiamata in questo modo perché le istruzioni che controllano del flusso di esecuzione si possono far corrispondere a tre tipi di strutture: la struttura condizionale, la struttura sequenziale e la struttura iterativa.
- Programmazione *modulare* o *procedurale*: i linguaggi di programmazione orientati a questo modello, simile al precedente, consentono anche di suddividere il lungo elenco delle istruzioni in parti più piccole, dette funzioni, che sono dei moduli indipendenti tra di loro ed eventualmente riutilizzabili. Questo sistema permette anche di suddividere il problema in sotto-problemi più semplici da risolvere.
- Programmazione *ad oggetti* (Object Oriented Pogramming): anche i linguaggi di programmazione orientati a questo modello consentono di creare moduli logicamente separati che costituiscono il programma, ma, questa volta, la suddivisione si basa anche su quali dati essi devono elaborare. Un modulo che è stato scritto per elaborare un certo dato non può essere utilizzato per altri dati. Un modulo che è stato scritto per elaborare dati in una certa area della memoria non può elaborare nessun altro dato al di fuori di essa. La manutenzione di questo tipo di programmi risulta più semplice perché se un dato in uscita non è corretto ci può essere solo un modulo che può aver causato quell'errore. D'altro canto, la realizzazione di questi programmi richiede una profonda analisi preliminare del problema che deve essere risolto.

Linguaggi compilati e interpretati

In generale, i linguaggi di programmazione possono essere classificati in due tipi: linguaggi compilati e linguaggi interpretati. In entrambi i casi il programmatore scrive le istruzioni, che costituiscono il codice sorgente, all'interno di un file di testo non formattato, come quelli con estensione txt.

1. Se il linguaggio utilizzato è di tipo "compilato", per poter essere eseguito dal computer deve essere prima "tradotto" e trasformato il un file eseguibile (che nei sistemi operativi Microsoft hanno estensione .exe). Nel file sorgente le istruzioni sono visibili al programmatore sotto forma di testo, mentre nel file eseguibile le istruzioni sono in forma numerica.

2. Se il linguaggio utilizzato è di tipo "interpretato", il codice sorgente non viene tradotto in un file di istruzioni numeriche. Il computer deve quindi, prima di eseguire qualsiasi istruzione testuale, interpretarne il significato. Questi linguaggi consentono all'utente di leggere anche il testo del codice sorgente, ma sono meno efficienti di quelli compilati.

Riassumendo
1. codice sorgente (txt) > compilazione > file eseguibile (.exe)
2. codice sorgente (txt) tradotto ad ogni esecuzione

Javascript

Il nome esatto di questo standard è ECMA 262
Il nome comune e diffuso è Javascript, ma crea confusione con quello di un altro linguaggio: il Java, che è ben diverso dal Javascript.
Javascript è un linguaggio di programmazione orientato agli oggetti. La programmazione ad oggetti permette di realizzare più facilmente grandi programmi e di commettere meno errori di distrazione. Questo perché il programmatore che utilizza la programmazione ad oggetti è costretto ad astrarre e scomporre il problema e a seguire precise regole di programmazione.

Dove usare Javascript

Un programma scritto in linguaggio Javascript può essere interpretato da un browser e può aggiungere (in una pagina web) funzioni di interazione con l'utente non presenti nel linguaggio HTML o CSS, perché questi **non** sono linguaggi di programmazione. Non tutti i browser possono eseguire programmi in Javascript e quindi non bisognerebbe abusarne. Inoltre, sebbene il Javascript sia uno standard internazionale, ogni browser si comporta in modo leggermente diverso dall'altro e il programmatore deve tenerne conto.

Se il codice del programma si trova nel file di testo chiamato "programma.js", questo si inserisce nella pagina web utilizzando il seguente codice HTML

```
<head>
      <script type="text/javascript" src="programma.js"></script>
</head>
```

Quando viene eseguito?

Le istruzioni o le funzioni in linguaggio Javascript vengono di solito associate a particolari eventi (events). Per il browser un evento può essere la pressione di un pulsante in una pagina, oppure il caricamento di un'immagine o dell'intera pagina.

Per eseguire una funzione Javascript al caricamento di una pagina si può usare questo codice HTML:

```
<body onload="nomeFunzione(variabile1,variabile2)">
```

Questo sistema ha diversi inconvenienti, tra cui quello che bisogna attendere il caricamento di tutte le immagini nella pagina prima che le funzioni indicate siano eseguite. Al termine dello corso sul javascript saranno analizzate strategie che permettono di evitare questi ritardi nell'esecuzione.

Il seguente esempio apre una finestra di dialogo e chiede di inserire il proprio nome. Notare che il codice dentro a <body> è completamente vuoto: la pagina mostrerà qualcosa solo grazie al javascript.

```
<html>
    <head>
       <script type="text/javascript">
            function miaFunzione()
            {
                var name=prompt("Scrivi il tuo nome","Mario");
                if (name!=null && name!="")
                {
                     document.write("<h1>Ciao " + name + "! </h1>");
                }
             }
       </script>
    </head>

    <body onload="miaFunzione()">
            <p>
                  Prova della funzione write(),
                  che sostituisce tutto il contenuto della pagina
            </p>
    </body>

</html>
```

Introduzione all'uso delle funzioni

Per capire cosa sia una funzione in un programma basta pensare ad una funzione matematica, dove c'è la variabile indipendente *x* e quella dipendente *y*. La variabile dipendente di solito è il risultato che si trova elaborando la prima variabile, come $y=f(x)$. Ad esempio, la funzione $y=x^2$ calcola il quadrato dei valori forniti tramite *x*.
Ad ogni funzione viene affidata l'esecuzione di un compito specifico e ogni volta che si deve svolgere quel compito si chiama in esecuzione quella funzione. Come accade nelle funzioni matematiche, anche le funzioni di un programma possono contenere tra parentesi gli argomenti da utilizzare. Alcune funzioni possono essere prive di argomenti, ma le parentesi si devono scrivere sempre.

Si possono riassumere alcune regole per l'uso delle funzioni:

- le funzioni che si vogliono utilizzare devono essere dichiarate e definite: bisogna spiegare come è fatta e cosa fa una funzione prima di poterla usare; dov'è la **definizione** della funzione del precedente esempio?
- una funzione può essere eseguita o perché è chiamata dal verificarsi di un "evento" su un certo tag html (vedi prossimo paragrafo) oppure perché è stata chiamata da un'altra funzione; come è stata **chiamata** la funzione del precedente esempio?
- quando è chiamata, una funzione può ricevere uno o più valori come argomenti tra parentesi, come f(x) oppure f(a,b,c); quanti **argomenti** ci sono nella funzione del precedente esempio?
- quando termina il suo lavoro, una funzione può restituire un valore a chi l'ha chiamata usando il comando "return". Nel precedente esempio, cosa restituisce miaFunzione() e cosa restituisce la funzione prompt()?

Rispondere alle 4 precedenti domande anche per tutti i prossimi esempi...

Introduzione all'uso degli eventi

Il seguente codice esegue la funzione "nomeFunzione()" quando si verifica un particolare evento che corrisponde al caricamentodel <body> (on load) della pagina html:

```
<body onload="nomeFunzione(variabile1,variabile2)">
```

Il prossimo esegue la funzione "aprire()" quando si fa doppio click (on double click) su un'immagine:

```
<img ondblclick="aprire()" src="./percorso/foto.jpg"> alt="foto" />
```

Esempi

Provare il prossimo esempio, dove il tag <img> chiama una funzione quando si verifica l'evento doppio click.

```
<html>
    <head>
       <script type="text/javascript">
            function aprire()
            {
                    miaFinestra = window.open("","","width=200,height=300");
                    miaFinestra.document.write("<h1>wow!!!</h1>");
            }
       </script>
    </head>
    <body>

      <img src="foto.jpg" alt="la mia foto" ondblclick="aprire()" />

    </body>

</html>
```

questo programma modifica lo stile di alcuni elementi di una pagina

```
    <html>
      <head>

      <script type="text/javascript">
      function aggiungiColore() {
      document.getElementById('mio').style.backgroundColor='lime';
      }
      </script>
      </head>
      <body>
      <div style="background-color:pink; text-align:center;">
      <h1>getElementById</h1>
      </div>

      <div id="mio" style="background-color:yellow; height:100px; margin:20px;">
        il javascript può essere usato per modificare, in seguito ad eventi,
        il codice HTML e il codice CSS
      </div>

      <input type="button" value="Colorami" onClick="aggiungiColore();" />

      </body>
    </html>
```

Questo verifica la presenza o l'assenza di un dato in un form:

```
<html>
    <head>
       <script type="text/javascript">
                  function validareForm()
                  {
                        var x=document.forms["mio"]["var"].value;
                        if (x==null || x==="")
                        {
                              alert("manca un dato da inviare");
                              return false;
                        }
                              else
                        {
                              document.forms["mio"].submit();
                        }
                  }
      </script>
    </head>
    <body>

      <form id="mio" action="pagina.php" method="get" >
            testo
            <input name="var" type="text" />
            <input type="button" value="invia" onclick="validareForm()" />
      </form>

    </body>
</html>
```

in alternativa alla precedente funzione `validareForm()` provare anche questa:

```
      function validareForm()
      {
            var x=document.forms["mio"]["var"].value;
            if (x==null || x==="")
            {
                  document.getElementById('inputvar').style.backgroundColor='red';
                  document.getElementById('inputvar').value='campo obbligatorio';
            }
            else
            {
                  document.forms["mio"].submit();
            }
      }
```

Questo programma controlla il mouse

```
<html>
    <head>
       <script type="text/javascript">
                  function bloccareMouse(event)
                  {
                        if (event.button==2)
                        {
                              alert("Operazione vietata!");
                        }
                  }
</script>
    </head>

    <body onclick="bloccareMouse(event)">

      <div style="background-color:yellow; height:300px;">

      </div>

    </body>
</html>
```

La programmazione ad oggetti si basa su oggetti (come una finestra) che hanno determinate proprietà (come le sue dimensioni) e metodi (come quelli che chiudono o aprono una finestra).

La sintassi delle istruzioni ha una delle due seguenti forme:

- `oggetto.proprietà`
- `oggetto.metodo()`

Nel primo caso, ad esempio, l'oggetto può essere il documento html e la proprietà specificare un suo tag, oppure l'oggetto può essere l'evento e la sua proprietà specificare quale pulsante ho premuto.

```
event.button
document.forms["mio"]["var"]
```

Nel secondo caso, l'oggetto può essere il la finestra del browser e il metodo può aprire una nuova finestra, oppure l'oggetto può essere il documento html e il metodo ottenere l'accesso ad uno specifico tag

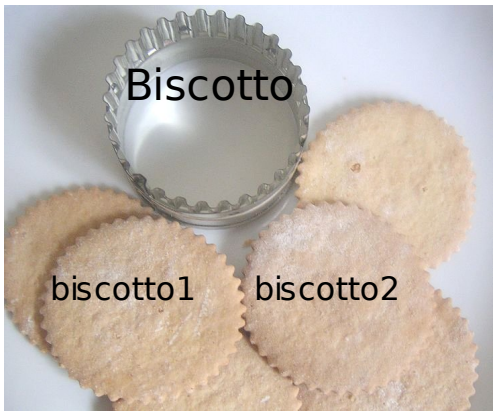
```
window.open("","","width=200,height=300")
document.getElementById('inputvar')
```

Introduzione alla OOP.

Per comprendere le basi fondamentali della Object Oriented Programmation bisogna prima di tutto aver scritto almeno una volta un programma (di qualsiasi tipo). Poi è necessario introdurre quattro nuovi concetti: le Classi, gli Oggetti, le Proprietà e i Metodi. Questi concetti astratti sono collegati tra di loro e vengono usati come un modello.

Classi

La Class (Classe) è un modello utilizzato dal programmatore per creare tanti oggetti uguali.
La Classe è come lo "stampo" che la nonna usa per creare tanti biscotti uguali.
Una Classe si riconosce perché il suo nome ha l'iniziale maiuscola (nel seguente esempio la classe si chiama Biscotto), mentre l'oggetto che si crea a partire dalla Classe si riconosce dall'iniziale minuscola (biscotto1).



http://commons.wikimedia.org/wiki/File:Motlle_crespellines.jpg

Una classe può essere rappresentata graficamente in questo modo:

Biscotto
colore: testo forma: testo temperatuta: numerico
raffreddare(numerico)

Questo schema permette di comprendere rapidamente quale sia la struttura di una classe, e come creare ed utilizzare gli oggetti di quel tipo.
Il titolo in maiuscolo è il nome della classe
La seconda parte contiene le proprietà della classe di cui si specifica il tipo di dato
La terza parte contiene i metodi della classe, cioè le funzioni che possono svolgere, di cui si specifica il tipo di dato delle variabili e il tipo di dato restituito.

Oggetti (istanze della classe)

La nonna può fare biscotti usando degli stampini esistenti oppure inventare nuove forme per i biscotti. Anche il programmatore può creare gli oggetti di cui ha bisogno usando Classi esistenti, fatte da qualcuno, oppure inventando nuove Classi.
Ad esempio, in Javascript, si può usare la classe Biscotto per creare un oggetto e chiamarlo "biscotto1".

```
biscotto1 = new Biscotto;
```

In Javascript esiste la classe `String` che può essere usata per creare un oggetto e chiamarlo "frase". Questo è il codice:

```
var frase = new String("ciao a tutti");
```

Queste istruzioni creano un nuovo oggetto del tipo della classe String.
Il nuovo oggetto si chiamerà "frase" e dentro ci saranno memorizzate le parole "Ciao a tutti".

Proprietà

Biscotto
colore: text forma: text temperatura:numeric
raffreddare()

Come già detto, ogni oggetto, appartenente ad una certa classe, possiede delle proprietà che lo caratterizzano. Se l'oggetto fosse un biscotto, una sua proprietà potrebbe essere il colore o il peso. Ogni biscotto può avere un colore e un peso diverso da quello di un altro biscotto. Per accedere al valore di una proprietà di un oggetto si usa la "dot notation", cioè si deve usare una particolare regola sintattica, del tipo:

```
oggetto.proprietà
```

```
biscotto1.colore
```

In Javascript, nel caso degli oggetti della classe String, come l'oggetto `frase`, una sua proprietà è chiamata `length`, e contiene il numero dei caratteri della frase. Questo numero potrebbe essere copiato in un'altra variabile con l'istruzione:

```
var lunghezza = frase.length;
```

Questo era solo un esempio: tutte le proprietà di questi oggetti verranno viste in dettaglio in seguito.
A volte può capitare che non sia necessario creare gli oggetti, semplicemente perché questi già esistono all'avvio del programma, perché sono degli oggetti molto importanti.
Ad esempio:

- l'oggetto `window` è la finestra della pagina web dove viene eseguito il programma Javascript. Per questo motivo questo oggetto non deve essere creato dal programmatore;
- l'oggetto `document` è il contenitore del codice HMTL visualizzato nella finestra. Questo oggetto è una proprietà dell'oggetto `window` e viene creato automaticamente quando si crea la finestra. Per questo motivo questo è un altro oggetto che non deve essere creato.

Nota: per procedere ulteriormente nello studio e comprendere i metodi è necessario essere a conoscenza di almeno un linguaggio di programmazione procedurale e conoscere il significato del termine "funzione" all'interno di un programma.

Metodi della classe

Biscotto
colore: text forma: text temperatura: numeric
raffreddare()

Ogni oggetto, dopo essere stato creato, può eseguire tutti i metodi che appartengono alla sua classe. Un metodo è una funzione o un'azione.
Ad esempio, se l'oggetto fosse il Biscotto biscotto1, un suo metodo potrebbe essere la funzione `raffreddare()` che consente all'oggetto di eseguire una certa azione, come abbassare la propria temperatura a 15 gradi Celsius.

```
biscotto1.raffreddare(15);
```

In Javascript, l'oggetto `window` possiede dei metodi per aprire altre finestre.
Ad esempio, per visualizzare un avviso all'utente si può usare il metodo alert(), come nel seguente codice:

```
window.alert("fai un click su ok");
```

In quest'altro esempio, usando il metodo open(), si può vedere come un metodo può essere usato per creare un nuovo oggetto di tipo window e quindi per aprire una nuova finestra del browser:

```
finestra2 = window.open("","","width=200,height=300");
```

Questo codice apre una nuova finestra di dimensioni 200x300 pixel.
La nuova finestra si chiama `finestra2` ed è un oggetto dello stesso tipo dell'oggetto `window`.
Tutti i metodi usati nei precedenti esempi verranno visti nel dettaglio in seguito, ma si può già notare che `open()` è un metodo che "restituisce" un nuovo oggetto.
Questo tipo di oggetti sono un caso particolare perché, per la loro creazione, non si segue lo schema visto per creare di solito un nuovo oggetto (vedere la creazione di `biscotto1).`

Oggetti window

Di solito, ogni oggetto, prima di essere usato, deve essere creato dal programmatore, partendo dal modello della sua classe, ma non l'oggetto window perché esso corrisponde alla finestra del browser, e che è già stata aperta automaticamente. Questo tipo di oggetti (come `window` o `finestra2`) hanno delle proprietà e dei metodi. Provare, continuando l'esempio precedente, a modificare la seguente proprietà, per modificare le dimensioni della finestra del browser:

```
finestra2.outerHeight = finestra2.outerHeight - 90;
```

Ogni oggetto window possiede le sue proprietà: quelle di `finestra2` sono diverse da quelle di `window.` Provare ad esempio a modificare la seguente proprietà:

```
window.outerHeight = window.outerHeight + 90;
```

Una proprietà associata ad oggetti di questo tipo (window) può essere a sua volta un oggetto, come l'oggetto document che verrà studiato di seguito.

window
outerHeight:numeric outerWidth: numeric document: object navigator: object
open(): window alert() confirm() prompt()

La seguente tabella riassume le principali PROPRIETÀ dell'oggetto WINDOW

Propietà	**Descrizione**
closed	Restituisce un valore Boolean che indica se la finestra è chiusa
document	vedi oggetto document
location	vedi oggetto location
navigator	vedi oggetto navigator
history	vedi oggetto history
screen	vedi oggetto screen
outerHeight	Restituisce o imposta (set) l'altezza della finestra
outerWidth	Restituisce o imposta (set) la larghezza della finestra
screenX	Restituisce la coordinata x della finestra sullo schermo
screenY	Restituisce la coordinata y della finestra sullo schermo

La seguente tabella riassume i principali METODI dell'oggetto WINDOW

Metodo	Descrizione
open()	Apre (e restituisce) una nuova finestra del browser
close()	Chiude la finestra
alert(str)	Crea un messaggio di avviso (pulsante OK)
confirm(str)	Crea un messaggio di avviso (pulsanti OK e Cancel)
prompt(str)	Crea una finestra di dialogo dove si può inserire un testo di input
focus()	Imposta il focus sulla finestra
blur()	Rimuove il focus dalla finestra
moveTo(x,y)	Sposta una finestra alle coordinate x,y indicate
print()	Stampa il contenuto della finestra
resizeTo(a,b)	Ridimensiona la finestra specificando larghezza e altezza
setTimeout(m)	Esegue un'istruzione dopo un numero di millisecondi

Oggetti document

`document` è un oggetto creato automaticamente come proprietà di un oggetto window. Poichè non è necessario creare l'oggetto window, non è necessario creare nemmeno quello `document` perché è una sua proprietà. `document` contiene il codice HTML della pagina e, a sua volta, come ogni altro oggetto, contiene metodi e proprietà.
Ogni oggetto window possiede un proprio oggetto `document`: `window` ha il proprio e `finestra2` ha il proprio. Per indicare i due diversi oggetti si deve scrivere: `window.document` oppure `finestra2.document`
Provare ad esempio ad utilizzare il seguente metodo, che visualizza un testo in due finestre diverse:

```
window.document.write("<h1>Scrivo in window</h1>");
finestra2.document.write("<p>Scrivo anche in finestra2</p>");
```

L'oggetto `document` ha delle proprietà e può eseguire azioni

document
title: string URL: string
open(): object write(str) close() getElementById(str) getElementByName(str) getElementByTagName(str)

La seguente tabella riassume le principali PROPRIETÀ dell'oggetto DOCUMENT

Proprietà	**Descrizione**
title	Restituisce il codice racchiuso nel tag <title> del document
URL	Restituisce l'URL completo del document
referrer	Restituisce l'URL da cui è stato aperto il document
cookie	Restituisce gli elementi dei cookies del document (nome e valore)

La seguente tabella riassume i principali METODI dell'oggetto DOCUMENT

Metodo	**Descrizione**
open(*URL,name,specs,replace*)	Crea e restituisce un nuovo oggetto document che sostituisce quello attuale nella finestra
write(str)	Invia il testo specificato (codice HTML o javascript) al document per visualizzarlo nella finestra
close()	Si usa solo dopo open(). Conclude tutte le operazioni di write e le visualizza all'interno del document
getElementById(str)	Restituisce il solo elemento HTML con "id" specificato
getElementsByName(str)	Restituisce tutti gli elementi HTML con tale "name"
getElementsByTagName(str)	Restituisce tutti gli elementi HTML di tipo specificato

Oggetti navigator

`navigator` è un oggetto creato automaticamente come proprietà di un oggetto `window`. Non è necessario creare l'oggetto `navigator` perché è una proprietà di window, esattamente come `document`.
`navigator` è un oggetto che contiene informazioni sul browser utilizzato dall'utente per visualizzare la pagina HTML e, a sua volta, come ogni altro oggetto, contiene metodi e proprietà.

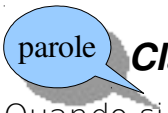
La seguente tabella riassume le principali PROPRIETÀ dell'oggetto NAVIGATOR

Proprietà	**Descrizione**
cookieEnabled	Restituisce un valore boolean relativo al browser usato
userAgent	Restituisce informazioni sul browser e sul sistema op. usato

La seguente tabella riassume i principali METODI dell'oggetto NAVIGATOR

Metodi	**Descrizione**

javaEnabled()	Specifies whether or not the browser has Java enabled



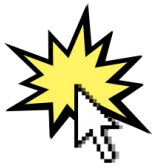
Classe String

Quando si vuole lavorare sulle parole è necessario utilizzare oggetti della classe `String`. A differenza delle classi viste fin'ora, questo tipo di oggetto deve essere normalmente creato per poi essere utilizzato (per memorizzare una frase ed eseguire azioni sulla stessa).

```
var frase = new String("<h1>ciao a tutti</h1>");
window.document.write(frase);
```

Gli oggetti di tipo `String` possiedono, come tutti gli oggetti, proprietà e metodi. Nei due seguenti esempi verrà prima visualizzata la proprietà che indica il numero dei caratteri contenuti in un oggetto di tipo String e poi verrà eseguito l'oggetto restituito da un metodo che trasforma il testo in maiuscolo:

```
window.document.write(frase.length);
window.document.write(frase.toUpperCase());
```



Gli eventi

Sono anche essi degli oggetti appartenenti ad un certo tipo classe (come window) e possono essere usati come un attributo inline di un tag, ad esempio:

```
<body onload="miaFunzione()">
```

Il valore dell'attributo è il nome della funzione da chiamare in occasione di tale evento.

Nella seguente tabella sono elencati i principali tipi di eventi
(tratto da http://www.w3schools.com/jsref/dom_obj_event.asp)

evento	tag	descrizione
onLoad	*	indicare il caricamento della pagina o dell'immagine
onUnload	*	indicare il cambiamento della pagina
onClick	*	si fa click su un elemento
onDblClick	*	si fa un doppio click su un elemento
onMouseDown	*	si tiene premuto un tasto del mouse su un elemento
onMouseUp	*	si rilascia un tasto del mouse su un elemento
onMouseOver	*	si va sopra all'elemento con il puntatore
onMouseOut	*	si esce da sopra l'elemento con il puntatore
onMouseMove	*	si muove il puntatore sopra un elemento
onKeyDown	input	il tasto viene tenuto premuto
onKeyUp	input	il tasto viene rilasciato
onKeyPress	input	il tasto viene premuto e rilasciato
onChange	input	viene immesso un qualsiasi input
onSubmit	input	quando si fa click sul pulsante di tipo submit
onFocus	input	un elemento ottiene il focus
onBlur	input	un elemento perde il focus

Per usare un evento è possibile utilizzare diverse strategie:

1. evento inline
2. evento traditional
3. evento secondo lo standard w3c

```
1. <img onclick="miaFunzione()" />
2. img.OnClick=miaFunzione();
3. img.addEventListner('click',miaFunzione(),false);
```

Gli oggetti di tipo evento hanno proprietà e metodi come gli altri oggetti

metodi
e....
proprietà

event.target contiene l'elemento della pagina che ha causato l'evento
e.type contiene il nome dell'evento che è stato rilevato
e.clientX posizione mouse
e.clientY posizione mouse
e.button pulsante del mouse

Press a mouse button over this text!

var txt="<html><body>Learning about the HTML DOM is fun!</body></html>";
doc.write(txt);
doc.close();

```
var el = document.createElement("iframe");
el.setAttribute('id', 'ifrm');
document.body.appendChild(el);
el.setAttribute('src', 'http://www.google.com');
```