JavaScript
executable contents in Web pages

Linguaggio di programmazione con caratteristiche object oriented, segue il modello di computazione **event driven**. L'utente genera degli eventi (tramite il mouse o la tastiera) che attivano l'esecuzione di sequenze di istruzioni (**event handler** o **gestori di eventi**).

La versione più nota è la parte **client-side** che permette di interagire con il browser e con i documenti in esso contenuti grazie all'accesso agli oggetti del **Document Object Model** (DOM).

In particolare, con il client-side JavaScript è possibile

1. controllare l'aspetto di un documento
2. controllare il browser
3. creare finestre di dialogo
4. interagire con i moduli
5. creare e modificare i cookie
6. ……..

La versione **server-side** è meno usata ma fornisce un'alternativa alla programmazione lato server. Il codice è incluso nei file HTML e viene eseguito dal server quando arrivano delle richieste da parte degli utenti. I file HTML che contengono codice server-side sono precompilati per motivi di efficienza.

Con questa versione del linguaggio è possibile

1. accedere all'oggetto File per leggere/scrivere file sul server
2. usare l'oggetto Database per interrogare basi di dati
3. usare l'oggetto Request per avere informazioni sulla richiesta HTTP (query-string, valori in input da form, …)
4. usare l'oggetto Client per salvare lo stato tra richieste HTTP multiple da parte dello stesso client

Prima di vedere la parte client-side bisogna conoscere il **core-language** che fornisce i costrutti sintattici del linguaggio (molto simili a quelli di C, C++, Java). Il modo più semplice per inserire degli script nei file HTML fa uso del tag

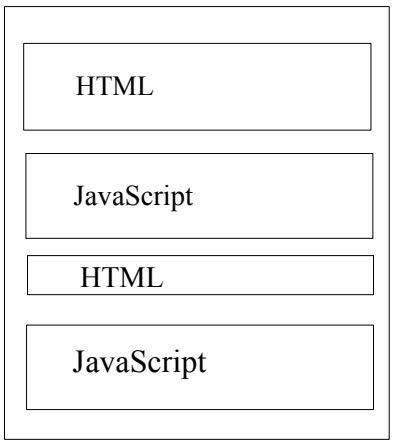
```
<script>

      inserire istruzioni qui

</script>
```

Il tag `<script>` possiede un attributo opzionale `language` che permette di specificare il linguaggio usato per gli script

```
<script language="JavaScript">      //si possono specificare le varie versioni del
                                    //linguaggio
<script language="VBScript">        // solo per Explorer
```

Se non si scrive nulla, il default è JavaScript.

La struttura di un file HTML, con l'introduzione del tag `<script>`, diventa simile a quella seguente:



Il browser legge il documento HTML (+ JavaScript) in modo sequenziale e

1. quando incontra tag HTML, **visualizza** le informazioni secondo le direttive di formattazione

2. quando incontra codice JavaScript, traduce ed **esegue le istruzioni** una alla volta (interpretazione); se ci sono degli errori l'esecuzione viene interrotta.

Interprete JavaScript

Digitando **javascript:** nella *location bar* (la barra degli indirizzi) del browser Navigator si apre una finestra (*JavaScript Console*) che permette di eseguire istruzioni JavaScript, una alla volta. Questa stessa finestra fornisce informazioni sugli errori che si verificano a tempo di esecuzione (purtroppo non sempre i messaggi che vengono visualizzati sono facili da capire).

Core language: struttura lessicale

La struttura lessicale di un linguaggio di programmazione fornisce un insieme di regole elementari che specificano come si devono scrivere i programmi; a livello lessicale si deve per esempio decidere come si possono scrivere gli identificatori, come si scrivono i commenti nel testo, come si separano le istruzioni.

1. JavaScript è *case-sensitive*, cioè distingue tra lettere maiuscole e lettere minuscole. Ad esempio, **`Nome`** e **`nome`** sono due identificatori diversi.

2. Gli identificatori non possono iniziare con un numero e non possono essere parole riservate del linguaggio. Ad esempio, l'identificatore **`cognome`** è legale, così pure **`_cognome`**, mentre non va bene l'identificatore **`3cognome`**. Esempi di parole riservate sono **`var, while, if, else, boolean, break, continue.`** Infine, gli identificatori non possono contenere dei *blank* (spazi bianchi) e quindi l'identificatore **`mio nome`** non è corretto.

3. Un programma è costituito da una **sequenza di istruzioni** che devono essere separate tra loro dal **;** (punto e virgola). Più istruzioni formano un *blocco di istruzioni* che inizia e finisce con le parentesi graffe. Possiamo scrivere

```
{                    oppure          { istr1; istr2; istr3;}
  istr1;
  istr2;
  istr3;
}
```

ed è preferibile la prima possibilità (per ragioni di leggibilità del codice). Se le istruzioni sono sulla stessa linea il separatore è obbligatorio, se sono su linee diverse può essere omesso (ma non è consigliabile).

4. Si possono inserire dei commenti nei programmi come segue:

```
// questo è un commento
/* questo è un commento che può essere spezzato su più righe */
```

Core language: variabili e tipi di dati

Una variabile identifica **una locazione di memoria** all'interno della quale si possono leggere e scrivere dei valori. Esistono variabili di tipo diverso e i **tipi di dati sono una parte fondamentale nella definizione di un linguaggio** di programmazione (**attenzione: JavaScript non è veramente tipato**).

In JavaScript abbiamo:

NUMERI (notazione *floating point* a 8 byte)
Si possono usare numeri decimali, ottali 0(0-7)*, esadecimali 0(x|X)(0-9|a-f|A-F)*, numeri con la virgola scrivendoli nella notazione scientifica, es. 3.14.
Il linguaggio fornisce le operazioni aritmetiche tradizionali e molte funzioni matematiche che vengono invocate nel contesto dell'oggetto **`Math`**.
Esistono dei valori numerici speciali (**`Infinity, -Infinity, NaN)`** e delle **costanti numeriche** definite come proprietà dell'oggetto **`Number`** (ad esempio, **`Number.MAX_VALUE`** e **`Number.MIN_VALUE`**).

STRINGHE
Permettono di usare **sequenze di caratteri** che possono essere incluse negli script tra apici semplici o apici doppi

```
"hello world"
"Via Dodecaneso, 35"
'30 marzo 2003'
```

Poichè il codice JavaScript è usato per produrre markup HTML si deve fare particolare attenzione all'uso degli apici singoli e degli apici doppi. In particolare, ci sono problemi quando in una stringa racchiusa tra apici semplici si vuole inserire una parola con l'apostrofo.

Esempio: `msg='Il tuo account è stato disabilitato dall'ultima visita';`

La stringa precedente non va bene e può essere modificata introducendo il carattere *backslash* (\) che, in combinazione con il carattere che lo segue, rappresenta un unico carattere altrimenti non utilizzabile (si parla di sequenza di *escape*).

```
msg='Il tuo account è stato disabilitato dall\'ultima visita';
```

Naturalmente si poteva anche scrivere:

```
msg="Il tuo account è stato disabilitato dall'ultima visita";
```

JavaScript fornisce operatori per operare con le stringhe. La **concatenazione** tra due stringhe si ottiene con l'operatore **+** (notate che questo operatore rappresenta la somma quando si lavora con i numeri interi, la concatenazione quando si lavora con le stringhe)

```
msg = "hello " + "world";
```

Per conoscere la lunghezza di una stringa si può usare la proprietà **`length`**

```
lunghezza = msg.length;
```

Esistono molti metodi (dell'oggetto **`String`**) che permettono di manipolare le stringhe.

BOOLEANI
Una variabile booleana può assumere solo **due valori** (`true` e `false`) che sono generalmente il risultato di un'operazione di confronto (**esempi**: `(a==b) (a!=b) (a>10)`).

ARRAY
Un array permette di definire una collezione di valori cui si può accedere mediante un indice. In genere gli elementi di un array sono tutti dello stesso tipo ma in JavaScript possono essere di tipo diverso. Gli array vengono creati con l'operatore **`new`** e il costruttotore **`Array()`** (visione ad oggetti) oppure mediante un elenco di valori separati dalla virgola.

Esempi

```
a = new Array( );        // crea un array inizialmente vuoto
a = new Array(3,5,7);    // crea un array [3, 5, 7]
a = new Array(3);        // crea un array [undefined, undefined, undefined]
                         // oppure [,,] a seconda del browser

a = [10, "prova", null];      // crea un array con dati di tipo diverso
```

Si può conoscere la lunghezza di un array usando la sua proprietà **`length`** nella notazione con il punto (si scriverà **`a.length`**). Si osservi che in questo modo l'array viene visto come un oggetto di tipo Array. Oltre alla proprietà **`length`** esistono dei metodi che permettono di manipolare gli array.
Gli array in Javascript sono **dinamici**: è sempre possibile aggiungere nuovi elementi ad un array semplicemente assegnando dei nuovi valori oltre l'ultima cella.

OGGETTI
Un oggetto è una collezione di dati più un insieme di funzioni che possono operare su questi dati (si parla di **proprietà** e **metodi**). JavaScript permette al programmatore di definire i propri oggetti mediante il costruttore **`Object()`** ma fornisce anche degli oggetti *built-in* e, soprattutto, una **gerarchia di oggetti** che descrivono i documenti contenuti nel browser: questa gerarchia è il **DOM** che vedremo nei dettagli nella parte client-side del linguaggio.

È possibile creare un oggetto vuoto con l'istruzione

```
var nome_oggetto = new Object( );
```

cui poi si assoceranno delle proprietà enumerandone i nomi e i valori

```
nome_oggetto.prop1 = valore1;
nome_oggetto.prop2 = valore2;
…
```

In alternativa si può usare una sintassi come quella seguente

```
var nome_oggetto = {

      prop1: valore1,
      prop2: valore2,
      prop3: valore3,
      ……
      ……
}
```

Per far riferimento ad una proprietà (o ad un metodo) di un oggetto si usa la sintassi

```
nome_oggetto.nome_proprietà          nome_oggetto.nome_metodo
```

Se la proprietà non esiste, JavaScript non genera errori ma restituisce il valore **undefined**.
È possibile accedere alle proprietà di un oggetto anche utilizzando la notazione degli array. In particolare, si può leggere/scrivere il valore di una proprietà indicizzandola mediante il suo nome (si parla di *array associativi*) .

```
nome_oggetto["nome_proprietà"]
```

Si noti che nella notazione standard il nome della proprietà è un identificatore, nel caso degli array associativi è una stringa.

Si possono eliminare le proprietà degli oggetti usando l'operatore **delete** oppure assegnandovi il valore **null.**

FUNZIONI
In JavaScript anche le funzioni sono considerate tipi di dati e possono essere passate come parametri ad altre funzioni (ma non le useremo in questo modo "sofisticato"). Ci basta sapere che vengono definite come segue

```
function nomefunzione(arg1, arg2, …, argn)
{
      istruzione1;
      istruzione2;
      istruzione3;
      …

      return [espressione]    // opzionale
}
```

e che sono richiamate specificando il loro nome e il valore dei parametri attuali.

Le funzioni possone restituire dei valori nell'ambiente chiamante usando la parola chiave **`return`** e una **espressione** che viene valutata per produrre il risultato della funzione stessa. L'istruzione **`return`**, inoltre, causa l'uscita dalla funzione. Scrivendo solo **`return`**, senza alcuna espressione, il risultato prodotto è **`undefined.`**

Esempio

```
function quadrato(x)
  {
      return x*x;
  }
```

Core language: dichiarazione delle variabili

Una variabile è un nome associato ad una locazione di memoria e può contenere dei dati. Come abbiamo già visto negli esempi precedenti, per scrivere un dato in una variabile si usa l'istruzione di **assegnamento**

```
variabile = valore;
```

Ad esempio:

```
i = 2;
i = i + 3;
i = j + k;
```

JavaScript è un linguaggio debolmente tipato e **quindi la stessa variabile può contenere dati di tipo diverso** (questo non vale in molti linguaggi di programmazione). Quindi si potrà scrivere

```
i = 2;
i = "ciao";
```

senza che si verifichi alcun errore.

Per essere usate le variabili "devono" essere **dichiarate,** usando la parola chiave **`var.`** Al momento della dichiarazione le variabili assumono il valore **`undefined.`**

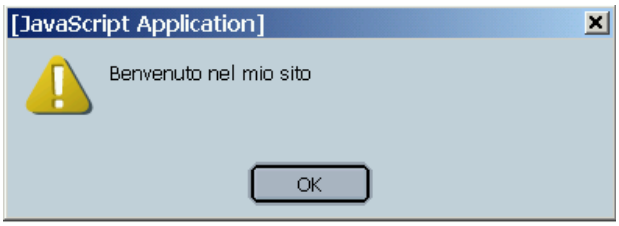
Anche in JavaScript si distingue tra variabili **globali** (visibili in qualunque punto del codice) e variabili **locali** (definite all'interno del body delle funzioni). All'interno del body di una funzione, se si ridefinisce una variabile già definita globalmente, l'effetto è quello di "nascondere" la variabile globale.

In realtà, in JavaScript **le variabili possono essere create anche senza essere dichiarate**: la prima volta che viene assegnato un valore ad una variabile, questa viene anche creata se non lo si era fatto in precedenza. Si noti che in molti linguaggi di programmazione l'assegnamento ad una variabile non dichiarata causa un errore.

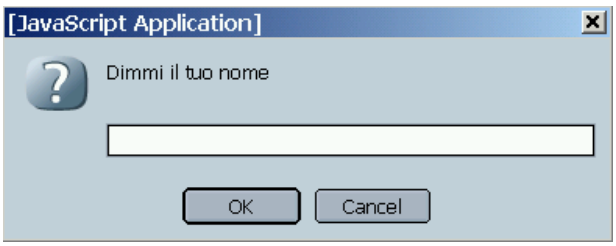
Per cominiciare a vedere qualche semplice esercizio, oltre alle **variabili** dobbiamo avere qualche meccanismo per realizzare **l'input/output**. Useremo alcuni **metodi** dell'oggetto **Window**. Vedremo questo oggetto più avanti, per ora ci limitiamo ad usare alcuni suoi metodi che ci permettono di fornire in input dei valori tramite la tastiera e di leggere -visualizzare sullo schermo- il contenuto delle variabili. I metodi sono: **`window.alert()`**, **`window.prompt()`**, **`window.confirm()`**.

NB: questi metodi possono essere invocati anche senza scrivere esplicitamente **`window`**, ma scrivendo solo **`alert()`**, **`prompt()`**, **`confirm()`**.

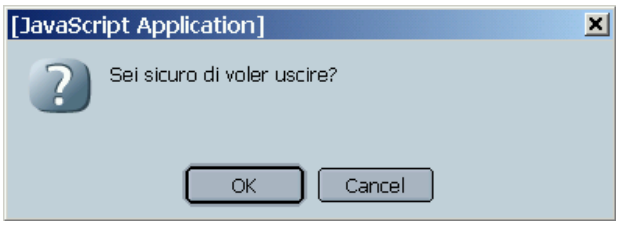
```
window.alert("Benvenuto nel mio sito")
```




```
window.prompt("Dimmi il tuo nome","");
```




```
window.confirm("Sei sicuro di voler uscire?");
```



Esempio
Uso del metodo `window.alert()` per realizzare una prima forma di output; osservate il valore delle variabili dopo la loro dichiarazione.

```
<html>
<body>

<center>
<H3> Primo esercizio in JavaScript</H3>
</center>

<h4>sto per iniziare lo script ed eseguirò le azioni in sequenza </h4>

<script>
var nome;
var cognome;
window.alert("Nome = " + nome + " Cognome= " + cognome);

nome = "Mario";
cognome = "Rossi";
window.alert("Nome = " + nome + " Cognome= " + cognome);
</script>

<h4>sono uscito dallo script e torno ad interpretare i tag HTML</h4>
</body>
</html>
```

Esempio
Uso dei metodi `window.prompt()` e `window.alert()` per realizzare una prima forma di input/output.

```
<html>

<body>
<center>
<h3> Secondo esercizio in JavaScript </h3>
</center>

<h4>1) inizio il primo script ed eseguirò le azioni in sequenza </h4>


<script>
var nome;
var cognome;
window.prompt("Dammi il tuo nome","");
window.prompt("Dammi il tuo cognome","");
window.alert("Nome = " + nome + ", Cognome= " + cognome);
</script>

<h4>2) i valori in input non sono stati memorizzati in nessuna variabile!!!</h4>

<h4>3) inizio il secondo script ed eseguirò le azioni in sequenza </h4>

<script>
var nome;
var cognome;
nome = window.prompt("Dammi il tuo nome","");
cognome = window.prompt("Dammi il tuo cognome","");
window.alert("Nome = " + nome + " Cognome= " + cognome);
</script>

<h4>4) ho finito il secondo script, torno ad interpretare i tag HTML</h4>

</body>
</html>
```

Esempio
Conversione da stringa a intero per effettuare la divisione. Poiché non c'è nessun controllo sui valori inseriti in input, provate l'input 0 e delle sequenze di caratteri e osservate il risultato della divisione.

NB: i dati letti tramite il browser (sia attraverso la finestra di dialogo generata dal metodo **`window.prompt()`** che attraverso gli elementi di un modulo) sono di tipo stringa. Per eseguire operazioni aritmetiche è necessario modificarne il formato (casting). Per fare questo, nell'esempio precedente è stato sottratto il valore 0 (zero) dall'input.

```
<html>

<body>
<center>
<h3> Terzo esercizio in JavaScript </h3>
</center>

<script>
var a;
var b;

a = window.prompt("Inserisci il primo numero","0");
a = a-0;    // converto in intero

b = window.prompt("Inserisci il secondo numero","0");
b = b-0;    // converto in intero

window.alert("Risultato della divisione: " + a/b);
</script>
</body>
</html>
```

Esempio
Visibilità delle variabili in JavaScript.
Nel primo script la funzione **`prova_scope()`** utilizza la variabile globale **`scope`**, nel secondo script la variabile **`scope`** viene ridefinita nel body della funzione.

NB: a differenza di altri linguaggi, JavaScript non ha un meccanismo di ambito di visibilità a livello di blocco. Tutte le variabili dichiarate all'interno di una funzione rimangono definite in tutta la funzione e questo può portare a comportamenti che non ci aspettiamo. Osservate con attenzione il risultato prodotto dal secondo script.

```
<script>
var scope="globale";
function prova_scope()
{
  window.alert("Valore della variabile prima dell'assegnamento: " + scope);
  scope = "locale";
  window.alert("Valore della variabile dopo l'assegnamento: " + scope);
}
prova_scope();  // richiamo della funzione
</script>
```

```
<script>
var scope="globale";
function prova_scope()
{
  window.alert("Valore della variabile prima dell'assegnamento: " + scope);
  var scope = "locale";
  window.alert("Valore della variabile dopo l'assegnamento: " + scope);
}
prova_scope();    // richiamo della funzione
</script>
```

Per scrivere dei programmi più complessi è necessario introdurre i costrutti del linguaggio. Iniziamo con il **costrutto condizionale** che permette di eseguire determinate sequenze di istruzioni a seconda che una certa condizione sia verificata o meno.

Istruzione condizionale if

La sintassi generale è

```
if (condizione)
 {
     istruzione11;
     istruzione12;
     …
     istruzione1N;
 }
else
 {
     istruzione21;
     istruzione22;
     …
     istruzione2M;
 }
```

Se la condizione tra parentesi tonde è verificata (cioè se vale **`true`**) si eseguono le istruzioni che fanno parte del ramo **`if`** del costrutto, cioè `istruzione11, istruzione12, …, istruzione1N`.

Se la condizione non è verificata (cioè se vale **`false`**) si eseguono le istruzioni che fanno parte del ramo **`else`**. Se in un ramo c'è una sola istruzione si possono togliere le parentesi.

Il ramo **`else`** è opzionale e può essere omesso.

Esempio

```
<html>

<body>
<center>
<h3> Quinto esercizio in JavaScript </h3>
</center>

<h4>1) inizio lo script ed eseguirò le azioni in sequenza </h4>

<script>
var nome;
var cognome;
var eta;
nome = window.prompt("Inserisci il tuo nome","");
cognome = window.prompt("Inserisci il tuo cognome","");
eta = window.prompt("Bene, " + nome + " " + cognome + ", ora dimmi anche la tua
età","");
eta = eta - 0;  // converto in numero intero

if (eta>17 && eta<26)
  window.alert("OK, visita il sito ... per le informazioni sul concorso");
else
  window.alert("Mi dispiace il concorso è riservato ai giovani tra i 18 e i 25 anni");

</script>

<h4>2) ho finito lo script e torno ad interpretare i tag HTML</h4>
</body>
</html>
```

Istruzione ripetitiva while

La sintassi generale è

```
while (condizione)
{
   istruzione1;
   istruzione2;
   …
   istruzioneN;
}
```

Le istruzioni `istruzione1, istruzione2`, etc. vengono ripetute fino a quando la condizione tra parentesi tonde è verificata (**`true`**). Quando la condizione diventa falsa, si passa ad eseguire la prima istruzione che segue il costrutto **`while`**.

Istruzione ripetitiva do while

È simile alla precedente ma le istruzioni vengono eseguite *almeno una volta*. La sintassi è

```
do
{
   istruzione1;
   istruzione2;
   …
   istruzioneN;
}
while (condizione)
```

Le istruzioni `istruzione1, istruzione2`, etc. vengono ripetute fino a quando la condizione tra parentesi tonde è verificata (**`true`**). Quando la condizione diventa falsa, si passa ad eseguire la prima istruzione che segue il costrutto **`do while`**. Le istruzioni vengono eseguite almeno una volta e le parentesi graffe non sono obbligatorie perché le parole chiave **`do`** e **`while`** delimitano il costrutto stesso.

NB: nel `while` (`do/while`) si deve fare molta attenzione a definire delle condizioni che prima o poi saranno verificate per evitare di entrare in cicli infiniti. **Se si entra in un loop si deve terminare (kill, Ctrl+Alt+Del) il processo associato al browser.**

Istruzione ripetitiva for

La sintassi generale è

```
for (inizializzazione; test; modifica)
 {
     istruzione1;
     istruzione2;
     …
     istruzioneN;
 }
```

Nella parte di inizializzazione si deve inizializzare una variabile (contatore) con un valore intero. Nella parte di test si deve verificare il valore del contatore che viene cambiato nella parte di modifica.
Si ripetono le istruzioni fino a quando la condizione specificata nella parte di test è verificata.

NB: l'uso del costrutto **`for`** insieme agli **`array`** (che in JavaScript sono dinamici) può causare un comportamento anomalo.

Supponiamo di avere un array `a` e di volerlo inizializzare con valori crescenti

L'istruzione

```
for (i=0; i < a.length; i++)
     a[i]=i;
```

non causerà nessun problema e l'array verrà inizializzato correttamente.

Se al posto di `<` nella condizione mettiamo `<=` non si uscirà più dal ciclo for. Perché ?

Istruzione ripetitiva for/in

La sintassi generale è

```
for ( variabile in oggetto )
 {
     istruzione1;
     istruzione2;
     …
     istruzioneN;
 }
```

L'istruzione **`for/in`** offre un modo per scorrere ciclicamente attraverso le proprietà di un oggetto: il corpo del ciclo viene infatti eseguito una volta per ciascuna proprietà dell'oggetto.

Con questo costrutto si possono enumerare le proprietà di un oggetto e i rispettivi valori anche senza conoscerne il nome.

Istruzione switch

L'istruzione condizionale **`if`** causa una differenziazione (*branch*) nel flusso di esecuzione di un programma a seconda del verificarsi o meno di una condizione. Si possono avere più "percorsi" usando più costrutti **`if`** in cascata ma quando questi "percorsi" dipendono dal valore di una stessa variabile allora si può usare il costrutto **`switch`** (si evita così di valutare più volte lo stesso valore). La sintassi è

```
switch (espressione)
{
     case a :
        istruzioni per a;
        break;
     case b :
        istruzioni per b;
        break;
     case c :
        istruzioni per c;
        break;
        …….
        …….
     default :
     istruzioni per il default;
}
```

I valori a, b, c, … nella parte **`case`** possono essere numeri interi, stringhe, valori booleani e non necessariamente in un singolo costrutto **`switch`** devono assumere tutti lo stesso tipo (perché il linguaggio non supporta i tipi di dati).

La prima volta che la condizione su un ramo **`case`** è soddisfatta, si eseguono le istruzioni corrispondenti e poi si esce dal costrutto grazie all'istruzione **`break`**. Senza questa istruzione, invece, l'interprete JavaScript passa ad eseguire i blocchi di istruzioni seguenti (in genere questo non è il comportamento che vogliamo).

Istruzione break

Causa l'uscita anticipata da un ciclo o da uno **`switch.`**

Istruzione continue

Si può usare all'interno dei cicli **`while`** e **`for`** e fa saltare l'iterazione corrente senza uscire dal ciclo (si passa all'iterazione successiva).

JavaScript offre al programmatore molti oggetti built-in (DOM) ma gli permette anche di definire i propri tipi di oggetti, o **classi**, che possono essere definiti usando la parola chiave **`function`** e poi istanziati usando l'operatore **`new`**. Senza entrare nei dettagli, segue il codice che permette di creare una classe **`Rect`** caratterizzata dalle proprietà **`base`** e **`altezza`** e dai metodi **`calc_area()`** e **`calc_perim().`**

```
<html>
<head>
<script>

// definizione del metodo per il calcolo dell'area
function calc_area()
 {
      return this.base * this.altezza;
 }

// definizione del metodo per il calcolo del perimetro
function calc_perim()
 {
      return (this.base + this.altezza) * 2;
 }

// definizione della classe Rect con 2 proprietà e 2 metodi

function Rect(x,y)
 {
      this.base = x;
      this.altezza = y;
      this.area=calc_area;
      this.perimetro=calc_perim;
 }

</script>
</head>

<body>
<center><h3> Definizione di oggetto rettangolo</h3></center>

<script>
var pippo = new Rect(4,10);  // creo un'istanza di Rect di nome pippo

// leggo la proprietà base del rettangolo pippo
window.alert("Base pippo: " + pippo.base);

// leggo la proprietà altezza del rettangolo pippo
window.alert("Altezza pippo: "  + pippo.altezza);

// calcolo l'area invocando il metodo area() sull'oggetto pippo
window.alert("Area pippo: " + pippo.area());

// calcolo il perimetro invocando il metodo perimetro() sull'oggetto pippo
window.alert("Perimetro pippo: " + pippo.perimetro());

</script>
...
...
```