

# Introduzione a JavaScript

## Perché JavaScript?



HTML e CSS sono (per il momento, perlomeno) privi di capacità di programmazione. Niente variabili, niente strutture di controllo di flusso, niente strutture iterative. Il *cervello* della elaborazione di pagine sempre più ricche, sempre più complesse (o apparentemente semplici) è scritto in JavaScript.

Anche per questa parte la traccia è Matthew Mc Donald, "[HTML5 - the missing manual](#)", all'Appendice B.

- Per eseguire una porzione di codice Javascript basta includerlo nel markup HTML al punto desiderato:  

```
<script>  
  // Codice da eseguire:  alert("Interrompiamo le trasmissioni per un  
annuncio speciale.");  
</script>
```
- I commenti ammettono entrambe le sintassi del C++ (`//` e `/* */`).
- L'uso del punto e virgola al termine di ogni riga *non* è obbligatorio, ma fortemente raccomandato per maggiore leggibilità e per evitare errori durante l'editing.
- È possibile anche includere script contenuti in file appositi che contengono soltanto codice JavaScript:  

```
<script src="MagicoScript.js"></script>
```

## Eventi e funzioni

L'esecuzione di codice Javascript può venire associata a vari *eventi* associati all'uso interattivo di vari elementi HTML. Per associare codice ad un'evento basta definire l'attributo HTML con lo stesso nome nell'elemento corrispondente. Ad esempio:

```

```

Qui 'showMessage' è, come in ogni linguaggio di programmazione, una *funzione*, blocco di codice ben definito che riceve una lista di argomenti e restituisce eventualmente un risultato, e che può venire definita globalmente attraverso un elemento `script` oppure in file sorgente separato:

```
function showMessage(additional_text)  
{  
  alert("This is an announcement from showMessage: " + additional_text);  
}
```

Questo è *equivalente* a dichiarare una variabile globale il cui contenuto è una funzione (parleremo delle

variabili di Javascript fra poco):

```
var showMessage = function (additional_text)
{
    alert("This is an announcement from showMessage: " + additional_text);
};
```

Ecco una breve lista di riferimento degli eventi utilizzati più di frequente:

Nome dell'evento/attributo	Quando viene generato?	Elementi a cui si applica
<a href="#">onClick</a>	Quando si preme un tasto del mouse su un elemento.	Praticamente tutti
<a href="#">onMouseOver</a>	Quando il puntatore viene spostato con il mouse su un elemento.	Praticamente tutti
<a href="#">onMouseOut</a>	Quando si fa uscire con il mouse il puntatore da un elemento.	Praticamente tutti
<a href="#">onKeyDown</a>	Quando si preme un tasto della tastiera.	<select>, <input>, <textarea>, <a>, <button>
<a href="#">onKeyUp</a>	Quando si rilascia un tasto della tastiera che era stato premuto.	<select>, <input>, <textarea>, <a>, <button>
<a href="#">onFocus</a>	Quando un elemento di controllo riceve il <i>focus</i> , cioè quando si posiziona il cursore nell'elemento in modo da poter scrivere qualcosa. Sono elementi di controllo le finestre di testo, i checkbox, etc.	<select>, <input>, <textarea>, <a>, <button>
<a href="#">onBlur</a>	Quando un elemento cessa di essere in <i>focus</i> .	<select>, <input>, <textarea>, <a>, <button>
<a href="#">onChange</a>	Quando il valore di un elemento di input viene cambiato. Per una casella di testo, questo evento viene generato solo quando ci si sposta su un altro elemento di controllo.	<select>, <input type="text">, <textarea>
<a href="#">onSelect</a>	Quando viene evidenziata una porzione di testo in un elemento di input.	<input type="text">, <textarea>
<a href="#">onError</a>	Quando il browser non riesce a scaricare un'immagine (tipicamente per un errore nella URL)	<img>
<a href="#">onLoad</a>	Quando il browser termina di scaricare una nuova pagina o un oggetto come un'immagine.	<img>, <body>
<a href="#">onUnload</a>	Quando il browser disattiva una pagina, ad esempio quando si imposta una nuova URL o quando si seleziona un nuovo link. L'evento viene generato appena <i>prima</i> che il browser inizi a scaricare la nuova pagina.	<body>

## Variabili, tipi di dati e scope

In Javascript ogni variabile, indipendentemente dal tipo di dati che conterrà (e che può essere testo, un numero intero, un numero a virgola mobile, un array o un 'oggetto' - array associativo), viene dichiarata sempre con la stessa keyword `var`:

```
var variabileGlobale = "Tutti vanno matti per le variabili.";
```

`typeof` `variabileGlobale` restituisce il tipo della variabile ( $\rightarrow$  `'string'`) - cambiando il valore cambia il tipo:

```
variabileGlobale = 123; //typeof(variabileGlobale == 'number')
variabileGlobale = true; //typeof(variabileGlobale == 'boolean')
variabileGlobale = undefined; //typeof(variabileGlobale == 'undefined')
```

`undefined` indica che una variabile non esiste o non è stata inizializzata.

Le variabili dichiarate all'interno di una funzione sono visibili solo all'interno della funzione (e delle funzioni *inline* lì definite - *lexical scope*):

```
function dumpVariable() { alert("variabileLocale == "+variabileLocale); }
function aFunction() {
    variabileLocale = "Wunderbar.";
    setTimeout( function() { dumpVariable(); }, 2000);
}
```

Il concetto di una funzione con associato il riferimento ad alcuni dati (ad esempio le variabili del contesto dove la funzione è stata definita) si chiama *closure*. [Closures are not complicated](#).

## Strutture condizionali e di controllo di flusso

Operatori e strutture condizionali (`if`, `switch/case`, `try/catch(/finally)`) e di controllo di flusso (`for`, `while`) sono molto simili a quelle dei linguaggi C, C++ o Java:

```
if (myNumber < 100) {
    // (Questo codice viene eseguito se il valore di myNumber è minore di 100)
}
else if (myNumber < 200) {
    // (Questo codice viene eseguito se il valore di myNumber è minore di 200
    // ma maggiore o uguale a 100.)
}
else {
    // Questo codice viene eseguito in tutti gli altri casi.
}
```

```
for (var i = 0; i < 5; i++) {
    // Questo codice viene eseguito per cinque volte.
    alert("Messaggio numero: " + i);
}
```

I dati vengono convertiti automaticamente dove possibile per soddisfare le varie operazioni. Dove occorra evitare che questo avvenga sono stati introdotti due operatori di uguaglianza speciali per Javascript:

`===` Verifica se due variabili sono identiche, quindi anche dello stesso tipo

`!==` Verifica se due variabili sono diverse, anche solo per il tipo

## Array

Gli array di Javascript possono contenere una lista di oggetti di qualsiasi tipo, e sono molto flessibili. Si possono inizializzare così:

```
var myColors = []; // Array vuoto.
```

riempire con il metodo `push`:

```
myColors.push("red");
myColors.push("green");
myColors.push("blue");
```

dereferenziare con le parentesi quadre:

```
var secondColor = myColor[1];
```

ed iterare utilizzando l'attributo `length` che contiene il numero di elementi:

```
for (var i = 0; i < myColors.length; i++) {
    alert("Ho trovato il colore: " + myColors[i]);
}
```

o con un iteratore automatico (come si fa anche per gli 'oggetti'), avendo cura del fatto che la variabile assume il numero d'ordine dell'elemento dell'array che viene considerato:

```
for (var i in myColors) {
    alert("Ho trovato il colore: " + myColors[i]);
}
```

## Oggetti (1)

La nozione di 'oggetto' in Javascript è basata su una generalizzazione del concetto di array nella forma di array associativo: la chiave di ricerca dell'array non è un numero, ma una stringa qualsiasi:

```
var oggetto = {}; // Vuoto, anche 'null'.
var oggetto = {elemento1: "Elemento 1"}; // Inizializzato con 1 elemento.
oggetto.elemento2 = "Elemento 2"; // oppure - equivalente:
oggetto['elemento2'] = "Elemento 2";
```

Ma, come diceva Larry Wall, Javascript è un linguaggio orientato agli oggetti. Il *costruttore* di un oggetto ha questa forma:

```
function Person() {
    this.firstName = "Mino";
    this.lastName  = "Cancelli";
    this.getName = function() { return (this.firstName + " " + this.lastName) }
}
var p = new Person;
(...)
delete p;
```

## Oggetti (2)

Le caratteristiche di un linguaggio di programmazione OO vengono realizzate così:

1. Ereditarietà - si realizza attraverso il membro `prototype`:

```
Person.prototype.speak = function () {
    alert("Parlo italiano!");
};
function Student(first, last, school) {
    Person.call();
    this.firstName = first;
    this.lastName = last;
```



```
this.school = school;
};
Student.prototype = Object.create(Person.prototype); // ECMAScript 5 a
Student.prototype.constructor = Student;
```

2. Polimorfismo - gli oggetti (e in particolare le funzioni) sono qualificati solo dal loro nome e non dal numero (e men che meno dal tipo) degli argomenti, quindi l'unica possibilità è:

```
function Student(first, last, school) {
{
    if (school === 'undefined') school = "Bicocca";
    (...)
}
```

## Oggetti (3)

3. Incapsulamento - le *uniche* variabili non pubbliche sono quelle interne a una funzione. Ma possono essere accessibili altrove attraverso le *closures*:

```
var person = (function () {
var fullName = "Mino Cancelli";
return {
    "setFullName" : function (newValue) {
        fullName = newValue;
    },
    "getFullName" : function () {
        return fullName;
    }
}; }()); // Invoca la funzione appena creata

alert(person.getFullName()); // Al nome si può accedere così
person.fullName = "Junk"; // Ma non così.
```

## Manipolazione di elementi della pagina


Esistono sistemi più efficienti e tersi per la manipolazione della struttura ad albero che contiene il documento HTML in fase di elaborazione (ad esempio il già citato [jQuery](#)), ma il metodo fondamentale nel core di Javascript per accedere a un elemento della pagina richiede di identificarlo per ID: `var mainTextElement = document.getElementById("mainText");` Il contenuto dei vari elementi si può poi manipolare facendo uso degli attributi dell'oggetto ottenuto. Questa è una lista dei principali:

Nome della proprietà	Descrizione
<code>className</code>	Permette di leggere o modificare il nome dell'attributo <code>class</code> dell'elemento selezionato. Questo permette di modificare l'aspetto dell'elemento (se esiste uno stile definito per tale classe negli stylesheet collegati alla pagina).
<code>innerHTML</code>	Permette di leggere o modificare <i>tutto</i> il codice HTML contenuto all'interno di un elemento. Questo permette <i>anche</i> di aggiungere o togliere altri elementi.

parentElement	Permette di accedere (e quindi anche di modificare) all'oggetto HTML corrispondente all'elemento che contiene (padre) l'elemento in esame.
style	Permette di leggere o modificare l' <i>oggetto style</i> corrispondente all'elemento selezionato. Per modificare lo stile occorre quindi aggiungere un <code>.style</code> all'oggetto contenuto in questa proprietà
tagName	Fornisce il nome dell'elemento HTML selezionato, senza < e >.

Ad esempio:

```
mainTextElement.innerHTML = "Il contenuto di questa pagina &grave;
modificato dinamicamente";
```



Enable style

Disable Style

