JavaScript (autore: Vittorio Albertoni)

Premessa

Il 30 aprile 1993 il CERN decide di rendere pubblica la tecnologia alla base del World Wide Web.

Nello stesso anno nasce il primo browser, software che, installato su un computer (oggi anche un tablet o uno smartphone), ci consente di entrare nel World Wide Web per vedere che cosa contiene: azione che viene subito battezzata con il nome di «navigazione»: si chiama Mosaic. Ma il browser che prende il sopravvento, subito l'anno dopo nel 1994, è Netscape Navigator, un perfezionamento di Mosaic.

Ciò che il browser trova nel World Wide Web viene visualizzato su una pagina e si tratta, nella tecnologia di base, di documenti arricchiti da collegamenti ipertestuali con altri documenti.

La Società che sviluppava Netscape Navigator era in parte proprietà della Sun Microsystem, quella che lanciò il linguaggio di programmazione Java che, proprio in quegli stessi anni, dopo un'incubazione iniziata nel 1991, si affermava in pieno. Da subito, pertanto, vi fu un connubio tra il browser e il linguaggio Java: le così dette applet. Si trattava di programmi che, inclusi nella pagina del browser, potevano essere eseguiti utilizzando la macchina virtuale Java presente sul computer. Un modo per vivacizzare la pagina Web e instaurare processi di interazione e non solo di lettura di contenuti: su una pagina Web che promoziona l'acquisto di un appartamento inserisco un programma attraverso il quale puoi calcolare quanto ti costa il mutuo per acquistarlo.

L'idea era talmente bella che Brendan Eich, nel 1995, sviluppò un nuovo linguaggio di programmazione, più facile da apprendere e da utilizzare rispetto a Java, ma, soprattutto, eseguibile attraverso un interprete incorporato nel browser, senza bisogno d'altro: per un brevissimo periodo si chiamò LiveScript ma venne subito battezzato JavaScript.

L'origine di questo nome resta misteriosa. Sul piano della natura, dello stile e della filosofia del linguaggio JavaScript - se si prescinde dal fatto che si tratta di un linguaggio orientato alla programmazione per oggetti e dalla somiglianza di alcune figure sintattiche insieme ereditate da precedenti linguaggi come il C - non ha nulla a che fare con Java. Probabilmente tutto deriva dalla volontà di contrapporre qualche cosa, sul terreno della vivacizzazione delle pagine Web, a JavaApplet: JavaScript.

Negli anni immediatamente successivi vi furono tentativi da parte di Microsoft di affermare un proprio linguaggio per fare le stesse cose su Internet Explorer, browser tardivamente partorito nel 1995, con impostazione proprietaria (ricordo VBScript e JScript, un clone di JavaScript per Internet Explorer).

Ma Netscape e Sun Microsystem decisero di standardizzare JavaScript e si affidarono all'European Computer Manufacturers Association (ECMA), fino a che, nel 2003, con la benedizione e l'aiuto finanziario della Sun Microsystem, sulle ceneri di Netscape Communication nasce la Mozilla Foundation e JavaScript si consolida come standard confermando la propria natura di software libero.

Dalla sua iniziale natura di linguaggio interpretato dal browser, cioè operante sul lato client in sede di navigazione su Internet, l'evoluzione attorno a JavaScript è stata tale che oggi possiamo utilizzare il linguaggio anche sul lato server, cioè per compiere elaborazioni sul server e inserire i risultati sulla pagina web per il client, fino a poter utilizzare un interprete installato sul computer per elaborazioni al di fuori dal Web.

Questa varietà di contesti nei quali viene utilizzato JavaScript origina una serie di complicazioni che fanno apparire questo semplicissimo linguaggio di programmazione più difficile di quanto non sia.

In questo manualetto mi propongo di attuare un'esposizione che, partendo dalle cose facili, faccia capire dove e di quale natura siano quelle più difficili.

Indice

1 Installazione

Come possiamo derivare dal suo nome, JavaScript è un linguaggio di scripting, cioè i suoi programmi non vengono preventivamente compilati su un file in codice macchina per essere eseguiti ma vengono interpretati per come sono scritti ed eseguiti di conseguenza.

Per scrivere il programma - più propriamente lo script - basta un editor di testo di quelli che si trovano su tutti i sistemi operativi (editor di testo e non word processor).

Se ci orientiamo prevalentemente all'utilizzo di JavaScript lato client dovremo scrivere o richiamare lo script in pagine web, pure da costruire: allora è bene che ci procuriamo un editor come Bluefish.

Per l'utilizzo di JavaScript lato client non ci serve altro che un browser abilitato a JavaScript[1].

Se vogliamo utilizzare JavaScript anche lato server o anche interpretandolo al di fuori del browser dobbiamo installare l'interprete che troviamo a questo indirizzo
https://nodejs.org/it/download/,
dove troviamo ciò che serve per Linux, Windows e Mac OS X.

In ogni caso il browser abilitato a JavaScript è uno strumento prezioso per l'apprendimento del linguaggio e per la sperimentazione dei nostri costrutti in quanto, nella zona degli strumenti di sviluppo, ci offre una console JavaScript attraverso la quale possiamo interagire con il linguaggio avendo a disposizione preziosi strumenti di code completion.

Per raggiungere la console JavaScript dobbiamo agire nel menu del browser alla ricerca degli strumenti di sviluppo.
Esistono comunque i seguenti comandi abbreviati allo scopo:
. per i browser Firefox, Google Chrome e Chromium la pressione contemporanea dei tasti `Ctrl+Shift+I`;
. per il browser Internet Explorer la pressione del tasto `F12`;
. per il browser Safari occorre preventivamente fare in modo che nella barra dei menu compaia la scelta degli strumenti di sviluppo (click sul pulsante delle IMPOSTAZIONI GENERALI in alto a destra ▷ PREFERENZE ▷ AVANZATE e scelta dell'opzione MOSTRA MENU SVILUPPO NELLA BARRA DEI MENU e poi, dalla barra dei menu, SVILUPPO ▷ AVVIA DEBUG JAVASCRIPT e scelta della CONSOLE).

2 Basi del linguaggio

In questo capitolo vediamo la parte facile di JavaScript, quella che si riferisce al nucleo centrale del linguaggio, indipendente dal contesto in cui applichiamo il linguaggio stesso.

Ciò che si trova in questo capitolo vale sia se lavoriamo lato client, sia se lavoriamo lato server o lavoriamo al di fuori del Web.

2.1 Struttura lessicale

Prima cosa da ricordare, in JavaScript maiuscole e minuscole hanno importanza: var, Var e VAR sono tre parole diverse.

Eventuali righe vuote tra una riga e l'altra di codice sono ignorate e possono invece essere utili per facilitare la comprensione del codice da parte di un lettore umano.

Il punto e virgola (`;`) alla fine di ciascuna istruzione è facoltativo (ma consigliato) se le istruzioni sono su righe diverse. E' obbligatorio per separare istruzioni scritte sulla stessa riga.

I commenti al codice, se posti sulla stessa riga dell'istruzione da commentare, sono preceduti da `//`. Se sono scritti su più righe, queste sono delimitate dai caratteri `/*` e `*/` e ciascuna riga inizia con il carattere `*`.

[1] Tutti i browser installati insieme al sistema operativo dovrebbero essere per default abilitati ad interpretare JavaScript. Per verificarlo e per avere indicazioni su come effettuare l'abilitazione eventualmente mancante basta collegarsi a questo indirizzo *https://enablejavascript.co/it/*.

Quando si scelgono simboli o nomi per identificare variabili o funzioni occorre evitare di utilizzare parole che hanno un loro significato nel linguaggio (le così dette parole chiave): visto che tutte queste parole sono in lingua inglese basta che per le nostre esigenze usiamo parole italiane ed evitiamo così il rischio di sovrapposizione.

2.2 Tipi di dati

JavaScript lavora su tre tipi primitivi: numeri, stringhe e valori booleani.

Numeri

Non c'è differenza tra numeri interi e numeri in virgola mobile: tutti i numeri sono rappresentati come valori in virgola mobile, con la precisione che in altri linguaggi (C e Java) è associata al tipo `double` (17 cifre, virgola compresa).

Attraverso gli operatori aritmetici + (somma), - (sottrazione), * (prodotto) e / (divisione) si lavora con i numeri in maniera diretta: 3 + 2 ritorna 5, 7 / 2 ritorna 3.5.

Stringhe

La stringa è una sequenza di caratteri racchiusi tra apici singoli (') o doppi (").

Attraverso l'operatore + si concatenano più stringhe: 'Vittorio ' + 'Albertoni' ritorna 'Vittorio Albertoni'

Valori booleani

Servono per indicare un valore verità e possono assumere due soli valori: `true` e `false`, per vero o falso.

2.3 Variabili e Costanti

La variabile è un nome associato a un dato e identifica una zona di memoria che permette di archiviare e manipolare i dati all'interno di un programma.

JavaScript è un linguaggio a tipizzazione molto debole: quando si crea una variabile non c'è bisogno di dichiarare il tipo di dato che deve contenere ed alla stessa variabile cui avevamo assegnato un numero possiamo, in un secondo tempo, assegnare una stringa.

Grazie alla mancata tipizzazione, il linguaggio converte automaticamente i valori da un tipo all'altro ove necessario.
Possiamo, per esempio, scrivere tranquillamente "10" / 5 ed ottenere il risultato 2, cosa che non mi risulta essere permessa in nessun altro linguaggio di programmazione (divisione tra una stringa contenente un numero e un numero).

Le variabili che intendiamo utilizzare vanno preventivamente dichiarate con la parola chiave `var` con la sintassi
`var` <identificatore> [= <valore>]
Per esempio
`var a` definisce una variabile di nome `a` con valore indefinito;
`var a = 5` definisce una variabile di nome `a` e le assegna il valore `5`, inizializzandola.

Una volta che la variabile è dichiarata possiamo assegnarle o modificarne il valore con l'istruzione
`<identificatore> = <valore>`.
`a = 15` assegna alla variabile a precedentemente definita il valore `15`.

Una variabile può essere globale o locale, a seconda se deve essere accessibile in qualsiasi punto del programma o solo all'interno di una parte del programma (funzione).
Se scriviamo l'istruzione di assegnamento senza avere prima dichiarato la variabile, questa viene creata con il nome indicato e come variabile globale, anche se l'istruzione di assegnamento è inserita in una parte del programma e dovrebbe essere creata come variabile locale.

Pertanto la pratica di eludere l'uso della parola chiave `var`, approfittando della possibilità di creare variabili globali con la semplice istruzione di assegnamento, è sconsigliabile.

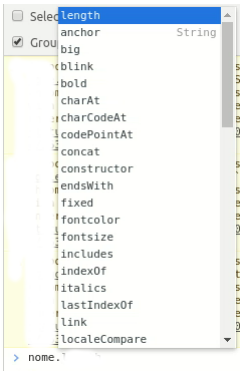
Ogni dato primitivo possiede una propria classe di oggetti, rispettivamente `Number`, `String` e `Boolean`, che ci mette a disposizione proprietà e metodi inerenti ciascun tipo.

Quando inizializziamo una variabile assegnando ad essa un valore di tipo primitivo, JavaScript associa ad essa un oggetto involucro dotato di proprietà e metodi adatti al tipo di dato assegnato.

Per esempio, con l'istruzione `var x = 100000` definiamo ed inizializziamo la variabile `x` al valore di tipo numerico `100000`. Successivamente, con l'istruzione `x.toString()`, che richiama il metodo toString dell'involucro della variabile numerica `x`, possiamo convertire il contenuto di tipo numerico di `x` in contenuto di tipo stringa. Con l'istruzione `x.toExponential()` possiamo ottenere che il valore `100000` sia rappresentato in notazione scientifica come `1e+5`.

Così, con l'istruzione `var nome = 'Vittorio'` definiamo ed inizializziamo la variabile `nome` al valore di tipo stringa `Vittorio`. Successivamente, con l'istruzione `nome.length`, che richiama la proprietà length dell'involucro della variabile stringa `nome`, possiamo trovare il valore corrispondente al numero dei caratteri che compongono la stringa (`8`).

Per conoscere le proprietà e i metodi associati ad una variabile basta scrivere il nome della variabile stessa nella console JavaScript del nostro browser seguito da un punto.
Se facciamo questo per la variabile `nome` di cui abbiamo appena parlato otteniamo quanto vediamo qui



e possiamo scegliere dal menu la proprietà o il metodo che ci interessa.

* * *

La variabile è lo strumento per definire nel programma un valore modificabile. Se vogliamo definire un valore che non sia modificabile dobbiamo definire una Costante. Ciò avviene con l'istruzione
`const` <identificatore> = <valore>.

2.4 Operatori

Gli operatori sono simboli che, inseriti tra valori o variabili contenenti valori, producono un risultato.

Operatori aritmetici

\+ per la somma,
\- per la sottrazione,
* per il prodotto,
/ per la divisione,
% per il modulo (resto della divisione),
** per l'elevamento a potenza.

La precedenza è quella delle espressioni algebriche (prima si eseguono moltiplicazioni e divisioni e poi addizioni e sottrazioni). Per ottenere precedenze diverse occorre usare le parentesi.
`3+2*5` dà `13`,
`(3+2)*5` dà `25`.

L'operatore + tra stringhe ne effettua la concatenazione.

Operatori di confronto

< minore di,
<= minore o uguale a,
> maggiore di,
>= maggiore o uguale a,
== uguale a,
=== identico a,
!= diverso da.

La differenza tra l'operatore di uguaglianza (==) e l'operatore di identità (===) sta nel fatto che il primo, se necessario, effettua automatiche conversioni di tipo per rendere possibile il confronto mentre il secondo no.
Se abbiamo la variabile numerica `a = 3` e la variabile stringa `b = '3'`,
`a == b` dà `true`,
`a === b` dà `false`.

Operatori logici

`&&` per l'AND logico,
`||` per l'OR logico,
`!` per il NOT logico.

Contenitore per costanti e funzioni matematiche

E' un oggetto che si chiama `Math` e possiamo vedere l'elenco di ciò che contiene digitando `Math.` in una console JavaScript.

Il contenuto si richiama con
`Math.`<costante>
`Math.`<funzione(parametro o parametri)>

Tra le costanti ricordo la costante e, richiamabile con l'istruzione `Math.E` e la costante π, richiamabile con l'istruzione `Math.PI`.

Tra le funzioni abbiamo
tutte le funzioni trigonometriche
`Math.abs()` per calcolare il valore assoluto,
`Math.ceil()` per arrotondare un numero all'intero superiore,
`Math.exp()` per calcolare l'esponenziale di e,
`Math.floor()` per arrotondare un numero all'intero inferiore,
`Math.log()` per calcolare un logaritmo naturale,
`Math.pow()` per calcolare una potenza (parametri: base ed esponente),
`Math.random()` per calcolare un numero casuale,
`Math.round()` per arrotondare all'intero più vicino,
`Math.sqrt()` per calcolare una radice quadrata.

Esempi:
`Math.sin(Math.PI/2)` dà `1`,
`Math.pow(3,5)` dà `243` (3^5),
`Math.sqrt(16)` dà `4`,
`Math.pow(27,1/3)` dà `3` ($27^{1/3}$, cioè $\sqrt[3]{27}$)

2.5 Istruzioni

Il programma (nel nostro caso meglio chiamarlo script) è un insieme di istruzioni.

2.5.1 Istruzioni semplici

Sono istruzioni molto brevi, che generalmente occupano una sola riga.

Espressioni matematiche

Sono quelle che, utilizzando valori espressi letteralmente o richiamando le variabili o le costanti che li contengono, attraverso gli operatori aritmetici o le funzioni del contenitore Math determinano un nuovo valore.

Per esempio, data l'esistenza di una variabile `x` e di una costante `y`,
`32 + y - Math.cos(x) + 13` è un'espressione matematica.

Istruzioni di assegnazione

Sono quelle che assegnano a una variabile o, una sola volta, a una costante un valore espresso letteralmente o attraverso un'espressione matematica.

L'operatore di assegnazione è il segno `=`.

Per esempio, data l'esistenza di una variabile `x`,
`x = 22 - Math.sqrt(9)` assegna a `x` il valore `19`.

2.5.2 Istruzioni condizionali

Normalmente le istruzioni contenute nello script vengono eseguite linearmente, una dopo l'altra, nell'ordine in cui si trovano.

Attraverso l'istruzione condizionale possiamo assoggettare l'esecuzione di una o più istruzioni al verificarsi di determinate condizioni.

if

La sua sintassi prevede tre possibilità

```
if <condizione> <istruzione>
if <condizione> <istruzione> else <istruzione>
if <condizione> <istruzione> else if <condizione> <istruzione> ...else <istruzione>
```

<condizione> va scritta entro parentesi tonde;
<istruzione> va scritta entro parentesi graffe.
Se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.

Nel primo caso se la condizione è vera viene eseguita l'istruzione, altrimenti non succede nulla e il programma continua con le altre istruzioni, se ce ne sono, non dipendenti dalla condizione.

```
if (x > y) {<istruzione_1>}
<altre_istruzioni>
```

se il valore di `x` è superiore al valore di `y` viene eseguita l'istruzione_1 e poi le altre istruzioni, in caso contrario vengono eseguite solo le altre istruzioni.

Nel secondo caso, se la condizione è vera viene eseguita la prima istruzione, altrimenti viene eseguita la seconda, quella dopo `else`.

```
if (x > y) {<istruzione_1>} else {<istruzione_2>}
<altre_istruzioni>
```

se il valore di `x` è superiore al valore di `y` viene eseguita l'istruzione_1 e poi le altre istruzioni, in caso contrario viene eseguita l'istruzione_2 e poi le altre istruzioni.

Nel terzo caso possiamo prevedere più casi, per esempio

```
if (<condizione>) {<istruzione_1>}
else if (<condizione>) {<istruzione_2>}
else if (<condizione>) {<istruzione_3>}
else {<istruzione_4>}
```

switch

Consente, in certi casi, di semplificare il codice quando le alternative `else if` viste prima sono tante.

La sintassi è

```
switch (<espressione>) {
    case <ricorrenza>:  <istruzioni>; break
    case <ricorrenza>:  <istruzioni>; break
    ...
}
```

dove
`<espressione>` genera valori,
`<ricorrenza>` è uno dei possibili valori generati da `<espressione>`,
`<istruzioni>` è ciò che si deve fare in caso di corrispondenza tra `<ricorrenza>` e `<espressione>`,
l'istruzione `break` dopo ciascun `case` va inserita per interrompere l'analisi al raggiungimento della ricorrenza.

Questo programmino traduce in inglese il giorno della settimana indicato in italiano in una variabile stringa chiamata `giorno` e colloca la traduzione in una variabile chiamata `traduzione`.

```
switch (giorno) {
   case 'Lunedì':  var traduzione = 'Monday'; break
   case 'Martedì':  var traduzione = 'Tuesday'; break
   case 'Mercoledì':  var traduzione = 'Wednesday';break
   case 'Giovedì':  var traduzione = 'Thursday';break
   case 'Venerdì':  var traduzione = 'Friday'; break
   case 'Sabato':  var traduzione = 'Saturday'; break
   case 'Domenica':  var traduzione = 'Sunday'; break
   default:  var traduzione = 'Non ho capito il giorno'; break
}
```

Notare la parola chiave `default`, per il caso in cui non si trovi alcuna corrispondenza tra `<ricorrenza>` e `<espressione>`.

2.5.3 Istruzioni di ciclo

Servono per ripetere più volte una istruzione o un blocco di istruzioni.

for

E' il classico comando per eseguire una istruzione per un numero prefissato di volte.

La sintassi è

```
for (<partenza>; <test>; <prossimo>) <istruzione>
```

dove
`<partenza>` imposta il valore iniziale di un contatore a valore zero,
`<test>` imposta una condizione,
`<prossimo>` conta le ripetizioni aumentando di una unità il contatore ad ogni giro,
`<istruzione>` è l'azione che si deve ripetere e va scritta entro parentesi graffe.
Se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.

Si usa identificare la variabile contatore con la lettera `i`.

```
for (var i = 0; i < 5; i = i + 1) {<istruzione>}
```

esegue cinque volte l'istruzione.

while

Esegue una o più istruzioni fino a quando è vera una condizione, verificando la condizione prima di ogni iterazione.

La sintassi è:

```
while (<condizione>) <istruzione>
```

dove

`<condizione>` è la condizione da verificare ad ogni ciclo,

`<istruzione>` è ciò che si deve fare fino a quando la condizione è vera e si scrive tra parentesi graffe.

Se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.

Se nella condizione utilizziamo un contatore, tra le istruzioni non dobbiamo dimenticare quella di aumentare di una unità il contatore stesso ad ogni giro.

```
var i = 0
while (i < 5) {<istruzione>; i = i + 1}
```

esegue cinque volte l'istruzione.

In questo otteniamo in altro modo ciò che avevamo fatto con il comando `for`.

Ma con `while` possiamo fare altre cose, potendo esprimere condizioni non necessariamente legate a un contatore.

do...while

Esegue una o più istruzioni fino a quando è vera una condizione, verificando la condizione alla fine di ogni iterazione.

La sintassi è:

```
do <istruzione> while (condizione)
```

dove

`<istruzione>` è ciò che si deve fare fino a quando la condizione è vera e si scrive tra parentesi graffe.

Se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.

`<condizione>` è la condizione da verificare ad ogni ciclo.

Se nella condizione utilizziamo un contatore, tra le istruzioni non dobbiamo dimenticare quella di aumentare di una unità il contatore stesso ad ogni giro.

Mentre nel ciclo `while` se la condizione non è vera già al primo ciclo non viene eseguita alcuna istruzione, nel ciclo `do...while` l'istruzione viene eseguita almeno una volta, indipendentemente dalla verifica della condizione.

2.6 Funzioni definite dall'utente

JavaScript ci offre tante funzioni predefinite. Si tratta delle funzioni membro dei molti oggetti che ha predisposto per noi chi ha costruito il linguaggio e che servono per fare tante cose. Ne abbiamo viste alcune, le funzioni membro dell'oggetto Math, nel precedente paragrafo 2.4.

Può accadere che risulti comoda una funzione che non troviamo tra quelle predefinite e abbiamo modo di costruircela noi utilizzando la parola chiave `function` con la seguente sintassi:

```
function <nome_funzione> (<parametro>, <parametro>,...)
{
   <istruzioni>
}
```

Per utilizzare la funzione ne scriviamo il nome seguito, tra parentesi tonde, dai parametri richiesti, con la sintassi
`<nome_funzione> (<parametro>, <parametro>,...).`

I parametri possono essere espressioni letterali, espressioni matematiche o contenuti di variabili globali create altrove.
Se si tratta di contenuti di variabili di tipo primitivo, cioè di dati del tipo visto nel precedente paragrafo 2.2, essi vengono passati alla funzione per valore e, qualunque cosa ne faccia la funzione, essi restano immutati nella variabile richiamata.

Quello che ho indicato è uno dei modi con cui possiamo costruirci una funzione in JavaScript, quello con approccio dichiarativo/statico. Dal momento che è il più semplice, da buoni dilettanti, ci limitiamo ad esso.

A titolo di esempio vediamo come possiamo costruirci una funzione per calcolare il fattoriale di un numero, esempio utile anche per dimostrare come il linguaggio consenta la ricorsività (funzioni che richiamano sé stesse).

```
function fattoriale(n)
{
   if (n <= 1) return 1
   return n * fattoriale (n - 1)
}
```

Per inserire in una variabile `f` il fattoriale di 15, una volta creata la funzione, basterà scrivere
`var f = fattoriale(15)`

2.7 Oggetti

JavaScript, più che essere, come altri linguaggi, orientato agli oggetti, è basato sugli oggetti: praticamente, in JavaScript tutto è oggetto.

Abbiamo visto che le stesse variabili, grazie al loro contenitore, sono degli oggetti dotati di attributi e metodi.

Abbiamo visto che esiste un oggetto Math che raccoglie funzioni per la matematica.

Troveremo altri oggetti precostituiti nel linguaggio per fare tutta una serie di altre cose.

Ovviamente possiamo creare noi stessi degli oggetti.

Il meccanismo costruttore parte da una funzione richiamabile con la parola chiave `new`.

La sintassi per la funzione è

```
function <nome_oggetto> (<parametro_1>, <parametro_2>,...)
{
   this.<nome_attributo_1> = <parametro_1>
   this.<nome_attributo_2> = <parametro_2>
   .....
   this.<nome_funzione_1> = <istruzioni>
   this.<nome_funzione_2> = <istruzioni>
   .....
}
```

La sintassi per utilizzare la funzione per costruire l'oggetto vero e proprio, mioOggetto, è

```
var mioOggetto = new <nome_oggetto> (<parametro_1>, <parametro_2>,...)
```

Attraverso le istruzioni
`mioOggetto.<nome_attributo>` e `mioOggetto.<nome_funzione>` possiamo poi recuperare gli attributi e i risultati dell'applicazione delle funzioni ai parametri indicati per costruire concretamente l'oggetto.

Per esempio, proponiamoci di radunare in un oggetto cerchio i metodi per determinarne area e circonferenza.
Definiamo in questo modo il costruttore

```
function cerchio (r)
{
    this.raggio = r
    this.area = r * r * Math.PI
    this.circonferenza = r * 2 * Math.PI
}
```
Costruiamo un cerchio concreto di raggio 3 con l'istruzione
```
var mioCerchio = new cerchio (3)
```
A questo punto, con le istruzioni
```
var a = mioCerchio.raggio
var b = mioCerchio.area
var c = mio Cerchio.circonferenza
```
collochiamo, rispettivamente nelle variabili a, b e c, il raggio, l'area e la circonferenza di un cerchio di raggio 3.

3 Interattività con l'utente

JavaScript può essere utilizzato in diversi contesti.

Dall'origine, nel lontano 1995, JavaScript è interpretato da un browser web abilitato a farlo e i risultati delle elaborazioni, effettuate sul computer dell'utente attraverso il browser, vengono esposti nel browser stesso: si tratta del contesto lato client ed è quello che ancora oggi va per la maggiore.

Da qualche anno, esattamente dal 2009, l'interprete JavaScript della Google, quello incorporato nei browser Chromium e Google Chrome, è stato utilizzato per una nuova tecnologia, chiamata Node, grazie alla quale è diventato possibile eseguire codice JavaScript anche in altri contesti.
Uno di questi è il contesto lato server: JavaScript è interpretato non sul computer dell'utente ma sul computer server e i risultati delle elaborazioni vengono esposti su una pagina web inviata dal server al computer dell'utente.
L'altro contesto è il terminale del computer, senza nessun collegamento con la tecnologia web, come avviene quando utilizziamo linguaggi come C, Pascal, Python, ecc.

JavaScript, in quanto tale, non è attrezzato per lavorare nei vari contesti, ma sono i contesti che si attrezzano per lavorare con JavaScript. Da qui sorgono le complicazioni: non basta conoscere il linguaggio JavaScript, che, come abbiamo visto nel precedente capitolo, è abbastanza semplice, ma occorre conoscere anche tutti gli strumenti che nei vari contesti sono stati predisposti per utilizzare JavaScript.

Nei paragrafi che seguono vedremo di capirci qualche cosa, segnatamente per quanto riguarda i diversi modi che abbiamo, in ciascun contesto, di creare interattività con l'utente: chiedergli elementi per fare delle elaborazioni (input) e comunicargli dati e risultati delle elaborazioni (output).

3.1 JavaScript lato client

Con questa espressione, nata quando le alternative erano soltanto lato client e lato server in ambiente web, ci si riferisce ancora oggi al contesto in cui JavaScript è interpretato da un browser web[2].

Affinché lo script sia interpretato da un browser web dobbiamo incorporarlo in una pagina HTML.
Il modo più diretto per farlo è inserire il codice dello script tra i tag del linguaggio HTML `<script>` e `</script>`. Un altro modo sarebbe quello di inserire lo script in un URL e richiamarlo nella pagina HTML, ma lasciamo queste complicazioni ai professionisti.

[2]Oggi abbiamo anche l'alternativa del terminale del computer che andrebbe classificata come lato client.

Il tag `<script>` possiede l'attributo opzionale `language` attraverso il quale, se vogliamo, possiamo specificare il linguaggio utilizzato per lo script, ad esempio `<script language = "Javascript 1.5"`. In questo esempio lo script verrà interpretato solo dai browser con JavaScript 1.5.

Per quanto riguarda l'interattività con l'utente, l'ambiente di programmazione fornito da un browser web contiene un oggetto globale di esecuzione, l'oggetto `window`, che fornisce i seguenti tre elementi utili per interfacciarsi con l'utente:
. `document`, oggetto che rappresenta il documento HTML visualizzato nella finestra del browser; il suo principale metodo è `write()`, che scrive qualche cosa sul documento HTML;
. `alert()`, metodo che visualizza una finestra a comparsa in cui si comunica qualche cosa;
. `prompt()`, metodo che visualizza una finestra a comparsa in cui si chiede qualche cosa.

In HTML, inoltre, si possono creare dei moduli (form), che sono documenti organizzati, vere e proprie interfacce grafiche per l'utente.

Tutto questo armamentario possiede anche strumenti di gestione degli eventi.

Facciamo alcuni esempi di cosa si possa fare in campo HTML senza bisogno di ricorrere a JavaScript.

Questo codice

```
<html>
   <input type = button
   value = 'SALUTA'
   onClick = "document.write('CIAO')">
</html>
```

scrive la parola CIAO su una pagina HTML alla pressione di un pulsante SALUTA.

Questo codice

```
<html>
   <input type = button
   value = 'SALUTA'
   onClick = "alert('CIAO')">
</html>
```

scrive la parola CIAO su una finestra a comparsa alla pressione di un pulsante SALUTA.

Questo codice

```
<html>
    <form>
        <input type = text name = nome>
        <input type = text name = saluto>
        <input type = button
        value = "SALUTA"
        onClick = "saluto.value = 'Ciao ' + nome.value">
    </form>
</html>
```

alla pressione di un pulsante SALUTA scrive un saluto personalizzato con il nome inserito in una prima finestrella (nome) in una seconda finestrella (saluto) di un semplice modulo (form) HTML.

Questo codice

```
<html>
    <form>
        <input type = text
        name = nome
        onChange = "document.write('Ciao ' + nome.value)">
    </form>
</html>
```

scrive un saluto personalizzato con il nome inserito in una finestrella (nome) in una pagina HTML non appena si dà INVIO al nome inserito.

Come si vede da questi esempi, la reazione agli eventi "pressione del pulsante" e "dato inserito" con la realizzazione di semplici risposte che non richiedano particolari elaborazioni si può fare senza ricorrere a JavaScript.
E qui mi sono limitato ad esemplificare solo alcuni degli eventi gestibili con il linguaggio HTML.

Ciò che manca al linguaggio HTML è la capacità di compiere elaborazioni sui dati inseriti dall'utente ed è a colmare questa mancanza che interviene JavaScript.

Quando JavaScript interviene combina le proprie le istruzioni con le istruzioni HTML e in questo modo sembra che JavaScript sia complicato, ma, in realtà, la complicazione deriva dal fatto che, oltre a JavaScript, occorre conoscere a fondo anche il linguaggio HTML.

Così, per scrivere la parola CIAO su una pagina HTML alla pressione del pulsante SALUTA, possiamo ricorrere a JavaScript in questo modo

```
<html>
   <input type = button value = 'SALUTA' onClick = saluta()>
   <script>
       function saluta()
       {
           document.write("CIAO")
       }
   </script>
</html>
```

Per avere un saluto personalizzato su una finestra a comparsa dopo aver inserito il proprio nome in un'altra finestra a comparsa possiamo ricorrere a JavaScript in questo modo

```
<html>
   <script>
       var nome = prompt('Come ti chiami?')
       alert('Ciao, ' + nome)
   </script>
</html>
```

Con il metodo `prompt()` possiamo ovviamente richiedere anche dati per eseguire calcoli, come in questo esempio:

```
<html>
   <script>
       var r = prompt('Inserisci il raggio di un cerchio')
       var a = r * r * Math.PI
       var c = 2 * r * Math.PI
       document.write('Per un cerchio di raggio ' + r)
       document.write('<br>')
       document.write("l'area è " + a)
       document.write('<br>')
       document.write('la circonferenza è ' + c)
   </script>
</html>
```

L'esempio mostra come, per andare a capo scrivendo nella pagina HTML, dobbiamo inserire un tag `<br>`. Peraltro, l'oggetto `document` possiede anche un metodo `writeln()`, che inserisce un `new line` alla fine della stringa, ma tutto ciò non viene riconosciuto in ambiente HTML.

Ovviamente il connubio tra HTML e JavaScript consente di realizzare anche cose più complesse di quelle esemplificate qui.
Quest'altro esempio mostra come realizzare un modulo HTML per calcolare la rata annua necessaria per estinguere un mutuo, utilizzando le direttive di formattazione del modulo HTML e la capacità di calcolo di JavaScript.
Notare come tutte le istruzioni per la creazione del form siano fuori dallo script, e non potrebbero essere nello script.

Notare, inoltre, come lo script richiami dati dalle finestrelle del form e inserisca dati nelle finestrelle del form.

```
<html>
    <form name="mutuo">
        <table>
            <tr><td colspan=3><big><b>Inserire i dati sul mutuo:</b></big></td></tr>
            <tr>
            <td>Importo del mutuo:</td>
            <td><input type=text name=importo size=12/></td>
            </tr>
            <tr>
            <td>Durata in anni:</td>
            <td><input type=text name=anni size=12/></td>
            </tr>
            <tr>
            <td>Tasso percentuale annuo:</td>
            <td><input type=text name=tasso size=12/></td>
            </tr>
            <tr><td colspan=3>
            <big><b>
            <input type=button value="Calcola" onclick="calcola()">
            </b></big>
            </td></tr>
            <tr>
            <td colspan=3><big><b>Informazioni sul rimborso:</b></big></td></tr>
            <tr>
            <td>La rata annuale sarà:</td>
            <td><input type=text name=rata size=12/></td>
            </tr>
            <tr>
            <td>Il rimborso totale sarà:</td>
            <td><input type=text name=rimborso size=12/></td>
            </tr>
            <tr>
            <td>Il totale degli interessi sarà:</td>
            <td><input type=text name=interessi size=12/></td>
            </tr>
        </table>
    </form>
    <script>
        function calcola()
        {
            var m = document.mutuo.importo.value
            var i = document.mutuo.tasso.value / 100
            var n = document.mutuo.anni.value
            var r = m * (i * (Math.pow(1+i,n)))/((Math.pow(1+i,n))-1)
            document.mutuo.rata.value = Math.round(r)
            document.mutuo.rimborso.value = Math.round(r * n)
            document.mutuo.interessi.value = Math.round(r * n - m)
        }
    </script>
</html>
```

Nota Bene:
I valori letti con le funzioni `prompt()` e `document.....value` sono di tipo stringa. Negli esempi non me ne sono preoccupato in quanto tali valori erano destinati ad entrare in calcoli insieme a valori numerici che ne avrebbero, come hanno, forzato la conversione (vedere in proposito a pagina 4). Per ottenerne subito la conversione in tipo numerico basterebbe comunque moltiplicarli, al momento della lettura, per 1.

* * *

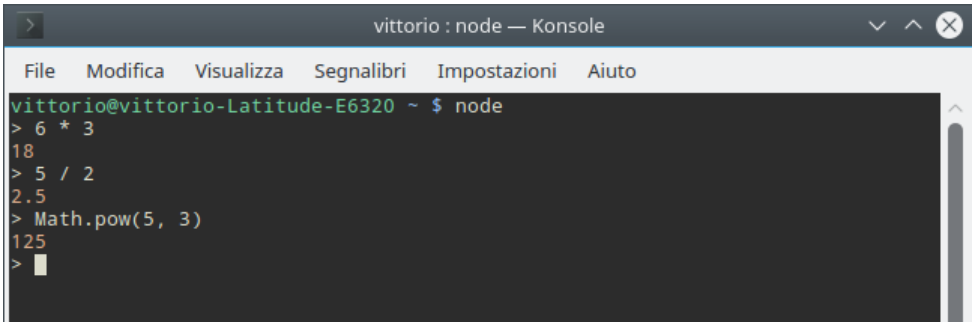
Mi rendo conto che la lettura di questo paragrafo non può aver fornito che una semplice infarinatura su tutta la materia. Ma il mio obiettivo era soprattutto quello di far vedere come, nel contesto JavaScript lato client, probabilmente la difficoltà maggiore stia nel padroneggiare il linguaggio HTML che, per certi versi, è più complicato del linguaggio JavaScript.

3.2 JavaScript sul terminale

In Linux e Mac OS X si chiama terminale, in Windows si chiama prompt dei comandi. Si tratta di una finestra nella quale, scrivendo con la tastiera, possiamo interagire con il computer.

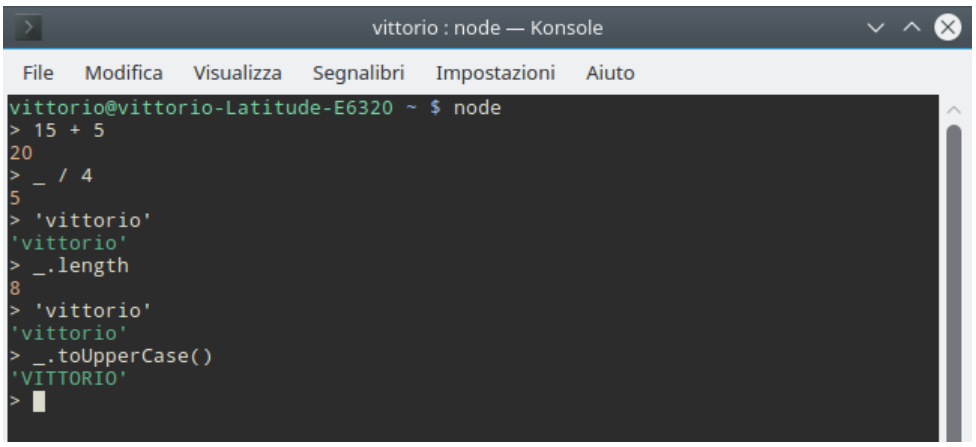
L'interprete che ci consente di utilizzare JavaScript da terminale è Node.

Se l'abbiamo installato secondo quanto indicato nel Capitolo 1, scrivendo sul terminale il comando `node` il terminale stesso si trasforma nella REPL (Read Eval Print Loop), che è la Konsole di Node, caratterizzata dal prompt > ad indicare la riga su cui scrivere comandi JavaScript per vederli eseguiti.



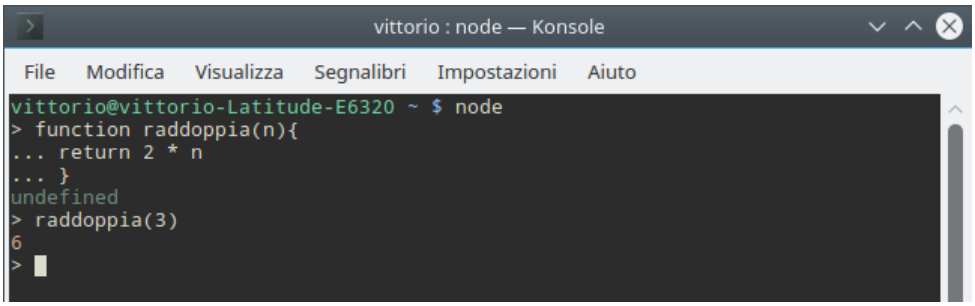
A differenza di altre shell per comandi di varia natura, la console di Node ha le seguenti particolarità:

. mette a disposizione una speciale variabile identificata dal carattere underscore (_) che può essere utilizzata per recuperare il risultato dell'ultima espressione eseguita, oppure per invocare un metodo o accedere alle proprietà di un oggetto:



. si presta alla code completion: una volta inserito il nome di un oggetto (o averlo richiamato con l'underscore) e il punto, premendo il tasto Tab vediamo l'elenco delle proprietà e dei metodi dell'oggetto stesso;

. si presta all'inserimento di istruzioni multilinea: dopo che abbiamo scritto qualche cosa sulla riga indicata dal prompt > e premuto Invio, la REPL capisce se ciò che abbiamo scritto è un'istruzione completa e, se così è, la esegue, altrimenti porta il cursore sulla riga successiva sulla quale evidenzia una serie di puntini, come invito a continuare:



La REPL è uno strumento utilissimo per acquisire dimestichezza con il linguaggio JavaScript, consentendoci di provare istantaneamente l'effetto di determinate istruzioni.

Volendo, possiamo utilizzare la REPL per compiere vere e proprie elaborazioni. In questo caso può essere utile salvare la nostra sessione di lavoro per poi poterla ricaricare per proseguire il lavoro stesso.
Il salvataggio si fa con l'istruzione
`.save` seguita dal percorso al nome del file con estensione .js in cui salvare la sessione di lavoro, e la ricarica si fa con l'istruzione
`.load` seguita dal percorso al nome del file in cui si era salvata la sessione di lavoro.

Per uscire da REPL basta scrivere il comando `.exit`

Se nel terminale facciamo seguire al comando `node` il nome di un file con estensione .js in cui abbiamo salvato uno script in linguaggio JavaScript otteniamo l'esecuzione dello script stesso nel terminale.

* * *

Ora veniamo alle modalità con cui Node consente interattività con l'utente.

La cosa più semplice è l'output, che si ottiene con il comando JavaScript
`console.log()`
nelle cui parentesi indichiamo ciò che va scritto (numero, stringa, contenuto di una variabile, risultato di una espressione matematica).

Per l'input le cose si complicano e dobbiamo ricorrere ad un modulo aggiuntivo di Node, il modulo readline con cui creare un'interfaccia di input e output. Le istruzioni necessarie sono le seguenti

```
var readline = require('readline');
var rl = readline.createInterface({
    input:  process.stdin,
    output:  process.stdout
});
```

A questo punto dobbiamo inserire nell'interfaccia tutto lo script, comprese le eventuali elaborazioni.

Un semplice script che richiede il nome all'utente per rivolgergli un saluto personalizzato diventa

```
var readline = require('readline');
var rl = readline.createInterface({
   input:  process.stdin,
   output:  process.stdout
```

```
});
rl.question("Come ti chiami?  ", function(leggi) {
   console.log("Ciao, ", leggi)
   rl.close()
})
```

Utilizzando l'operatore => quest'ultimo script può essere così semplificato

```
var readline = require('readline');
var rl = readline.createInterface({
   input:  process.stdin,
   output:  process.stdout
});
rl.question("Come ti chiami?  ", (leggi) => {
   console.log("Ciao, ", leggi)
   rl.close()
})
```

Purtroppo, se le richieste di input necessarie per le successive elaborazioni sono più di una, occorre nidificare le `question` con un lavoro da certosini, come ho dovuto fare in questo script per calcolare la rata del mutuo (è la versione per terminale di quello che abbiamo visto nel paragrafo precedente per l'ambiente HTML):

```
var readline = require('readline');
   var rl = readline.createInterface({
   input:  process.stdin,
   output:  process.stdout
});
rl.question('Importo:  ', (r1) => {
   rl.question('Tasso percentuale:  ', (r2) => {
       rl.question('Anni:  ', (r3) => {
       var m = (r1) * 1
       var i = (r2/100) * 1
       var n = (r3) * 1
       var r = m * (i*(Math.pow(1+i,n)))/((Math.pow(1+i,n))-1)
       console.log("Per l'estinzione:")
       console.log('rata:  ', Math.round(r))
       console.log('rimborso:  ', Math.round(r * n))
       console.log('interessi:  ', Math.round(r * n - m))
       rl.close();
       })
   });
});
```

Anche in questo caso i valori letti sono di tipo stringa e, se si vogliono inserire in variabili di tipo numerico, occorre moltiplicarli per 1, come è stato fatto (anche se, alla luce dei successivi calcoli, non sarebbe stato necessario).

Questi script, composti con un qualsiasi editor di testo, vanno salvati in un file con estensione `.js` ed eseguiti a terminale con il comando

```
node <nome_file.js>
```

Chi lavora in Linux o in Mac OS X può inserire nella prima riga dello script l'istruzione

```
#!/usr/bin/env node
```

indi rendere eseguibile il file per poterlo eseguire con il comando

```
./<nome_file.js>
```

In questo caso sarebbe inutile l'estensione `.js`.

3.3 JavaScript lato server

Con questa espressione ci si riferisce al contesto in cui JavaScript è interpretato da un server web che trasmette i risultati delle elaborazioni su una pagina HTML messa a disposizione del browser client.

Tutto ciò si realizza con la tecnologia Node, la stessa che abbiamo visto nel precedente paragrafo per l'esecuzione dello script su terminale.

Innanzi tutto, per provare questa tecnologia sul nostro computer, dobbiamo costruirci un server e lo facciamo con Node utilizzando il modulo `html`.

Le istruzioni sono le seguenti

```
var http = require('http')
var server = http.createServer(function (richiesta, risposta) {
risposta.writeHead(200)
var n = 5
var p = Math.pow(n, 3)
risposta.write('Cubo di ' + n + ':  ' + p)
risposta.end()
})
server.listen(8080)
```

Se salviamo questo script in un file chiamato, per esempio, `server.js` e lo lanciamo a terminale con il comando `node server.js`, apparentemente non notiamo nulla e il terminale rimane senza vita.
In realtà è stato avviato il server e se inseriamo nel nostro browser web preferito l'indirizzo `localhost:8080` apriamo una pagina web che riporta la scritta `Cubo di 5:  125`.

Le istruzioni partono dal richiamo della libreria `http` che contiene i metodi necessari per creare il nostro web server. La variabile `http` rappresenta l'oggetto JavaScript che ci permetterà di lanciare il server web e lo facciamo chiamando il metodo `createSever()` contenuto nell'oggetto `http` e salvando il risultato nella variabile `server`. Al metodo createServer passiamo come parametro la funzione che ci consente di gestire l'input (richiesta) e l'output (risposta).
La prima risposta serve per dire al browser "Tutto ok, la pagina è questa" (codice 200).
Seguono le elaborazioni per determinare il cubo di 5.
Quindi con il metodo `risposta.write()` viene scritta la risposta e si chiude con `risposta.end()`.
Infine indichiamo la porta per l'ascolto del server (8080).

Per uscire dal terminale dopo che abbiamo chiuso la pagina di risposta del server basta premere insieme i tasti CTRL e C.

Come si vede, le cose si complicano sempre più e, per questa dilettantesca rassegna del mondo JavaScript direi che può bastare.

A chi voglia approfondire la tecnologia Node suggerisco, per cominciare, di visitare il sito *https://www.nodeacademy.it/*, dove si trovano vari tutorial basici in lingua italiana.