JAVASCRIPT

Parte seconda

COSA PUÒ FARE JAVASCRIPT

MODIFICARE IL CONTENUTO HTML

- Il metodo **`document.getElementById()`** è un metodo JavaScript che serve ad identificare un elemento HTML con un determinato `id`.
- In questo esempio viene usato per *trovare* l'elemento HTML con `id="demo"` e cambiare il contenuto dell'elemento (**`innerHTML`**) inserendo la data corrente.
- Il codice JavaScript viene eseguito quando viene premuto un pulsante:

```
<p id="demo">
    Quello che leggi è il contenuto 'innerHTML' di questo paragrafo
</p>
<button type="button" onclick="document.getElementById('demo').innerHTML=Date()">
    Cliccami e vedrai la data di oggi!
</button>
```

CAMBIARE GLI ATTRIBUTI HTML

- In questo esempio il metodo `document.getElementById()` viene utilizzato per modificare il valore dell'attributo `src` dell'elemento `<img>` con `id="myimage"`.

```
<div>
  <p><img id="myimage" src="https://unsplash.it/400/300/?image=110" /></p>
  <p>
    <button type="button"
onclick="document.getElementById('myimage').src='https://unsplash.it/400/300/?image=110'">
    Immagine 110</button>
    <button type="button"
onclick="document.getElementById('myimage').src='https://unsplash.it/400/300/?image=111'">
    Immagine 111</button>
    <button type="button"
onclick="document.getElementById('myimage').src='https://unsplash.it/400/300/?image=112'">
    Immagine 112</button>
  </p>
</div>
```

CAMBIARE GLI STILI HTML

- In questo esempio il metodo `document.getElementById()` viene utilizzato per modificare il corpo del carattere dell'elemento `<p>` con `id="demo"`.

```
<p id="demo">
   Questo è il contenuto dell'elemento p con id="demo".
</p>
<button onclick="document.getElementById('demo').style.fontSize='40px'">
   Cliccami!
</button>
```

MOSTRARE E NASCONDERE

- In questo esempio il metodo `document.getElementById()` viene utilizzato per mostrare e nascondere l'elemento `<img>` con `id="myimage"`.

```
<div>
  <p>
    <img id="myimage" src="https://unsplash.it/400/300/?image=110" />
  </p>
  <p>
    <button onclick="document.getElementById('myimage').style.display='none'">Nascondi</button>
    <button onclick="document.getElementById('myimage').style.display='inline'">Mostra</button>
  </p>
</div>
```

DOVE INSERIRE JAVASCRIPT

IL TAG SCRIPT

- Il codice JavaScript va inserito tra l'apertura e la chiusura del tag `<script>` così:

```
<script>
  alert("ciao");
</script>
```

- Possiamo inserire il codice JavaScript in qualsiasi parte del documento (nella head oppure nel body) a seconda delle nostre esigenze.

FILE ESTERNO

- Quando si scrive codice di una certa lunghezza e/o che potrebbe essere ripetuto su più pagine
- Quando si utilizza una libreria Javascript esistente
- Per caricare Javascript da un file esterno si una l'attributo `src` applicandolo all'elemento `<script>`

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

GESTIONE DIRETTA EVENTO

- Javascript è fatto principalmente per rispondere a degli eventi, come quello di un utente che clicca un elemento della pagina
- Si può associare direttamente del codice javascript all'evento di un elemento usando appositi attributi come onclick, onload, ecc:

```
<button onclick="alert('Ciao!')">Cliccami !</button>
```

<noscript>

All'interno del tag noscript può essere utilizzata la sintassi HTML. Il contenuto viene visualizzato solo se il browser non supporta Javascript o Javascript è disabilitato per scelta dell'utente:

```
<noscript>
  <div>
      <h3>Per visualizzare correttamente il
          contenuto della pagina occorre avere
          JavaScript abilitato.</h3>
  </div>
</noscript>
```

SINTASSI

PROGRAMMI JAVASCIRPT

Un programma per computer è un elenco di "istruzioni" che può essere "eseguito" da parte del computer.

In un linguaggio di programmazione, le istruzioni del programma sono chiamati **statement**.

Le istruzioni JavaScript (statement) sono separate da un punto e virgola.

```
var x = 5;
var y = 6;
var z = x + y;
```

ISTRUZIONI JAVASCRPT

Le istruzioni JavaScript (statements) sono composte da:

- Valori
- Operatori
- Espressioni
- Parole chiave
- Commenti.

VALORI

- La sintassi JavaScript definisce due tipi di valori: i valori fissi e valori variabili.
- Valori fissi sono chiamati **letterali** o **costanti** . I valori variabili sono chiamati **variabili**

VALORI LETTERALI (O COSTANTI)

- Le *costanti* (o letterali) sono quantità note a priori il cui valore non dipende dai dati inseriti dall'utente e non cambia durante l' esecuzione del programma.
- Ogni tipo di dato ha il suo particolare modo di essere rappresentato in una costante. Perché il browser interpreti correttamente il valore bisogna seguire esattamente le regole.
- Esempi di letterali (vari tipi di dati):
  - A dare un valore iniziale ad una variabile
  - A confrontare una variabile con un valore di riferimento

ESEMPI DI VALORI LETTERALI

```
/* Number. Inizia con un numero */
10;                                                    // Numero intero
3.14159;                                               // Numero con parte decimale. Il simbolo è il punto
0xff0000;                                              // Un numero esadecimale è identificato dal prefisso 0x

/* String. Tra apici semplici o doppi */
'Ciao da Javascript';                                  // Stringa tra apici semplici
"Ciao da Javascript";                                  // Stringa tra apici doppi
'Il mio nome è "Pietro"';                              // Stringa tra apici semplici (che contiene apici doppi)
"Il mio nome è 'Pietro'";                              // Stringa tra apici semplici (che contiene apici semplici

/* Boolean: vero e falso */
true;                                                  // vero;
false;                                                 // falso;

/* Array */
["Lunedì","Martedì","Mercoledì","Giovedì","Venerdì","Sabato","Domenica"];

/* Object */
{ nome: "Mario", cognome:"Rossi", telefono: "3338887777" }

/* Altri valori letterali */
null;                                                  // Variabile senza un valore
undefined;                                             // Variabile non definita
NaN;                                                   // Not a Number
```

VALORI VARIABILI (VARIABILI)

In un linguaggio di programmazione, le variabili vengono utilizzate per memorizzare i dati.

JavaScript utilizza la parola `var` per dichiarare variabili.

Il segno `=` (uguale) viene utilizzato per assegnare un valore a una variabile.

In questo esempio, x è definito come una variabile. Quindi ad x viene assegnato il valore 6 (dato) :

```
var x;
x = 6;
```

OPERATORI

Javascript ha a disposizione numerosi operatori per compiere calcoli ed altre elaborazioni sui dati.

- Ad esempio gli operatori aritmetici (+ - * /) sono utilizzati per eseguire calcoli
- Mentre l'operatore = (uguale) viene utilizzato per assegnare un valore a una variabile.

```
var r, a;
r = 6 / 2;
a = r * r * 3.1416;
```

ESPRESSIONI

Un'espressione è una combinazione di valori, variabili e operatori, che può essere risolta calcolandone il valore.

Questa operazione di sviluppo e calcolo è chiamata una *evaluation*.

Ad esempio se scrivo:

```
var x;
x = 5 * 10;
```

x avrà il valore 50 perché l'espressione `5 * 10` viene risolta e ad x viene assegnato il risultato

Le espressioni possono anche contenere variabili come in:

```
x = y * 10;
```

Le espressioni posso contenere tipi di dato non numerici, se ad esempio scrivo:

```
var nome;
nome = 'Mario' + ' ' + 'Rossi';
```

nome conterrà 'Mario Rossi'

PAROLE CHIAVE

In JavaScript parole chiave sono utilizzate per identificare le azioni da eseguire.

Ad esempio la parola chiave **var** indica al browser di creare variabili.

```
var r, a;
r = 6 / 2;
a = r * r * 3.1416;
```

COMMENTI

Non tutto il codice JavaScript viene *eseguito*.

I frammenti di script compresi tra il segno `//` e la fine della riga e quelli che è compresi tra `/*` e `*/` sono considerati commenti e non vengono eseguiti.

```
/* Creo la variabile nome */
var nome;   // Dichiaro la variabile
nome = 'Mario' + ' ' + 'Rossi';  //Assegno un valore
```

IDENTIFICATORI

- Per identificatori (*identifiers*) si intendono i nomi che in JavaScript, si danno alle variabili, alle parole chiave e alle funzioni.
- Gli identificatori seguono regole grammaticali ben precise per altro comuni alla maggior parte dei linguaggi di programmazione.
- In JavaScript, il primo carattere di un identificatore deve essere una lettera, un carattere di sottolineatura (_), o il un simbolo del dollaro ($).
- I caratteri successivi possono essere lettere, cifre, sottolineature, o segni di dollaro.
- **Importante!** I numeri non sono consentiti come primo carattere.
- **Importante!** lo spazio è un separatore. Se introduco uno spazio in un identificatore creo due identificatori diversi.

JAVASCRIPT È CASE SENSITIVE

- Tutti gli identificatori JavaScript sono **case sensitive**.
- Questo significa che per JavaScript le variabili `Nome` e `nome` **sono due varibili diverse**.
- E che ~~`VAR`~~ o ~~`Var`~~ non sono interpretati come la parola riservata JavaScript `var`.

LEGGIBILITÀ DEGLI IDENTIFICATORI

Spesso per rendere significativo il nome di una variabile (o di una funzione) si uniscono più parole. Per favorire la leggibilità si usano degli accorgimenti che mantengano visivamente la divisione tra le parole senza violare le norme che regolano la grammatica degli identificatori. Molto spesso in Javascript si procede così:

- Quasi sempre i nomi delle variabili e delle funzioni iniziano (come per altro succede per le parole chiave) con la lettera minuscola perché si preferisce riservare l'iniziale maiuscola ai soli nomi delle classi.
- Quando un nome è composto da più parole posso separare le parole con un carattere underscore (_)
- In alternativa posso usare le iniziali maiuscole (*Camel Case*) per evidenziare le varie parole. Molto spesso in Javascript si usa lo stile Lower Camel Case che comunque mantiene minuscola le prima lettera del nome.

```
/* Definisco la varabile che contiene il Codice Fiscale */
var codice_fiscale;      // Underscore
var codiceFiscale;       // Lower Camel Case
```

ISTRUZIONI JAVASCRIPT

Uno script JavaScript è costituito da una seria di istruzioni che vengono eseguite dal browser web.

PUNTO E VIRGOLA

- La maggior parte del codice JavaScript contiene numerose istruzioni.
- Le istruzioni vengono eseguite, una per una, nello stesso ordine in cui sono state scritte.
- Il Punto e virgola separa le istruzioni JavaScript l'una dell'altra.

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML = z;
```

- Volendo posso collocare più istruzioni sulla stessa riga.

```
var x, y, z; x = 5; y = 6; z = x + y;
```

SPAZI E INTERRUZIONI DI RIGA

- Lo spazio è un elemento separatore fondamentale nel codice JavaScript. Ma gli spazi multipli o inutili sono ignorati. Posso quindi aggiungere quanti spazi voglio per rendere lo script più leggibile.
- Le seguenti linee sono equivalenti:

```
z = x + y;
```

```
z=x+y;
```

- Per una migliore leggibilità, di solito si evitano righe di codice troppo lunghe.
- Di solito il punto migliore in cui interrompere una riga è dopo un operatore.

```
document.getElementById("demo").innerHTML =
        "Ciao da Javascript";
```

BLOCCHI DI ISTRUZIONI

- Un **blocco di istruzioni** racchiuso tra parentesi graffe e viene considerato nella sintassi come un'unica istruzione.
- Viene utilizzato ogni qualvolta un gruppo di istruzioni vanne eseguite insieme come, ad esempio, quando si definisce una funzione.
- Spesso, per migliorare la leggibilità del codice, il blocco di istruzioni viene indentato rispetto al codice che lo precede.

```
function unaFunzione() {
    document.getElementById("demo1").innerHTML = "Ciao da Javascript.";
    document.getElementById("demo2").innerHTML = Date();
}
```

PAROLE CHIAVE JAVASCRIPT

Parola chiave	Descrizione
`break`	Esce da un blocco switch o da un ciclo
`continue`	Interrompe l'iterazione di un ciclo se una determinate condizione si verifica e contuna con quella successiva
`debugger`	Interrompe l'esecuzione e lancia il debugger, se disponibile.
`do ... while`	Esegue un blocco di comandi fino a che una determinata condizione è vera.
`for`	Esegue un blocco di comandi fino a che una determinata condizione è vera.
`for ... in`	Esegue un blocco di comandi per ogni elemento presente in un insieme (Array o Object)
`function`	Dichiara una funzione
`if ... else ... else if`	Esegue un blocco di comandi quando una condizione è vera
`return`	Interrompe l'esecuzione di una funzione e ritorna un valore
`switch`	Organizza una serie di blocchi di istruzioni dioendenti da condizioni alternative.
`throw`	Genera un errore.
`try ... catch ... finally`	Gestione degli errori
`var`	Dichiara una variabile.
`while`	Esegue un blocco di comandi fino a che una determinata condizione è vera.

VARIABILI

Le variabili JavaScript sono contenitori per memorizzare valori.

COME IN ALGEBRA

- In questo esempio, prezzo1, prezzo2, e totale, sono variabili:

```
var prezzo1 = 10;
var prezzo2 = 12;
var totale = prezzo1 + prezzo2;
```

- Nella programmazione, proprio come in algebra, usiamo le variabili (come **`prezzo1`**) per contenere i valori.
- Nella programmazione, proprio come in algebra, usiamo le variabili nelle espressioni (**`totale = prezzo1 + prezzo2`**).

NOMI DELLE VARIABILI

- Tutti le variabili devono essere identificate con nomi univoci .
- Questi nomi unici sono chiamati **identificatori** .
- Gli identificatori possono essere nomi brevi (come x e y) o nomi più descrittivi (somma, codiceFiscale).
- Le regole generali per la costruzione di nomi per le variabili (identificatori unici) sono:
  - I nomi possono contenere lettere, cifre, sottolineature, e segni di dollaro.
  - I nomi devono iniziare con una lettera, con $ o _
  - Nomi sono *case sensitive* (y e Y sono variabili diverse)
  - Le parole riservate (come le parole chiave JavaScript) non possono essere utilizzati come nomi

ASSEGNAZIONE DI UN VALORE

- In JavaScript, il segno di uguale (=) è un operatore *assegnazione*, non significa *uguale a*.
- La seguente espressione non ha senso in algebra:

  x = x + 5

- In JavaScript, invece assegna il valore dell'espressione **x + 5** a **x** (Viene calcolato il valore di **x + 5** e il risultato viene posto in **x**, in altri termini il valore di x viene incrementato di 5).

TIPI DI DATI

- Le variabili JavaScript possono contenere numeri (come `5`) e testi come `"Ciao da Javascript"`.
- In programmazione, i valori di testo sono chiamati `string`.
- Le variabili JavaScript possono contenere qualsiasi tipo di dato. Per ora parleremo di numeri (`Number`) e stringhe (`String`).
- Le stringhe di testo sono scritte tra virgolette doppie o singole. I numeri sono scritti senza virgolette.
- Se si mette un numero tra virgolette, si sarà trattato come una stringa di testo.

DICHIARAZIONE

La creazione di una variabile in JavaScript si chiama "dichiarazione". Si dichiara una variabile utilizzando il comando var:

```
var carName;
```

Dopo la dichiarazione, la variabile non ha alcun valore. (per essere precisi il valore speciale `undefined`). Per **assegnare** un valore alla variabile, si utilizza il segno `=`.

```
carName = "Fiat 500";
```

Posso anche assegnare un valore alla variabile all'atto della dichiarazione.

```
var carName = "Fiat 500";
```

Nell'esempio qui sotto, creiamo una variabile denominata carName, assegniamo alla variabile il valore "Fiat 500". Infine mettiamo il valore di carName all'interno di un paragrafo HTML con id = "demo":

```
<p id="demo"></p>
<script>
        var carName = "Fiat 500";
        document.getElementById("demo").innerHTML = carName;
</script>
```

Posso anche dichiarare più variabili con un unico comando var. In questo caso separo le variabili con una virgola.

```
var x = 5, y = 6, z;
```

OPERATORI

OPERATORI

- Gli operatori sono elementi del linguaggio composti di uno o più caratteri speciali o da una parola che applicati a uno o più operandi (che possono essere letterali o variabili) producono un risultato.
- Alcuni operatori usati in JavaScript:

```
++  !   !=  !==  %   %=   &   &&   &=  ()  -   *   *=    .   ?  :  /
/=   ^   ^=  |   ||   |=   ~   +   +=   <   <<   <<=   <=   <>   =
-=   ==   ===  >   >=   >>   >>=   >>>   >>>=  typeof  new
```

OPERATORI ARITMETICI

Operatore	Funzione	Espressione	Valore di y	Valore di x
		Valore iniziale: y=5		
+	Addizione	`x = y + 2`	`y = 5`	`x = 7`
-	Sottrazione	`x = y - 2`	`y = 5`	`x = 3`
*	Moltiplicazione	`x = y * 2`	`y = 5`	`x = 10`
/	Divisione	`x = y / 2`	`y = 5`	`x = 2.5`
%	Resto intero (Modulo)	`x = y % 2`	`y = 5`	`x = 1`
++	Incremento	`x = ++y`	`y = 6`	`x = 6`
		`x = y++`	`y = 6`	`x = 5`
--	Decremento	`x = --y`	`y = 4`	`x = 4`
		`x = y--`	`y = 4`	`x = 5`

PRECEDENZA DEGLI OPERATORI

Una operazione aritmetica si può operare su letterali:

```
var x = 100 + 50;
```

variabili:

```
var x = a + b;
```

espressioni:

```
var x = (100 + 50) * a;
```

Come nella matematica appresa a scuola la moltiplicazione (*) e la divisione (/) hanno una maggiore **priorità** di addizione (+) e sottrazione (-).

La precedenza può essere modificato utilizzando parentesi:

```
var x = (100 + 50) * 3;
```

PRECEDENZE

Precedenza	Operatore	Descrizione	Esempio
19	`()`	Operatore di raggruppamento	`(3 + 4)`
18	`.`, `[]`	Operatori di appartenenza	`persona.nome`, `persona["nome"]`
17	`()`, `new`	Richiamo di una funzione, creazione di un oggetto	`myFunction()`, `new Date()`
16	`++`, `--`	Incremento e decremento postfissi	`i++`, `i--`
15	`++`, `--`	Incremento e decremento prefissi	`++i`, `--i`
15	`!`	`not` logico	`!(x == y)`
15	`typeof`	Individuazione del tipo	`typeof x`
14	`*`, `/`, `%`	Moltiplicazione, divisione, modulo	`10 * 5`, `10 / 5`, `10 % 5`
13	`+`, `-`	Addizione, sottrazione	`10 + 5`, `10 - 5`

PRECEDENZE

Precedenza	Operatore	Descrizione	Esempio
12	<<, >>, >>>	Shift binario	x << 2, x >> 2, x >>> 2
11	<, <=, >, >=	Confronto (maggiore e minore)	x < y, x <= y, x > y, x >= y
10	==, ===, !=, !==	Confronto (uguale e non uguale)	x == y, x === y, x != y, x !== y
6	&&	**and** logico	x && y
5	||	**or** logico	x || y
3	=, +=, -=, *=, /=, %=, ecc.	Operatori di assegnazione	x = y, x += y, x -= y, x += y, x /= y, x %= y

OPERATORI DI ASSEGNAZIONE

Operatore	Espressione	Espressione equivalente	Valore assegnato a x
	Valori di partenza: `x = 10; y = 5`		
`=`	`x = y`	`x = y`	5
`+=`	`x += y`	`x = x + y`	15
`-=`	`x -= y`	`x = x - y`	5
`*=`	`x *= y`	`x = x * y`	50
`/=`	`x /= y`	`x = x / y`	2
`%=`	`x %= y`	`x = x % y`	0

OPERATORI DI ASSEGNAZIONE

- L'operatore = assegna un valore a una variabile.

```
var x = 10;
```

- L'operatore **+=** aggiunge un valore di una variabile.

```
var x = 10;
x += 5;
```

- L'operatore **-=** sottrae un valore da una variabile.

```
var x = 10;
x -= 5;
```

- L'operatore ***=** moltiplica il valore di una variabile per un numero.

```
var x = 10;
x *= 5;
```

- L'operatore **/=** divide il valore di una variabile per un numero.

```
var x = 10;
x /= 5;
```

TIPI DI DATO

TIPI DI DATI

Nella programmazione, il tipo di dato è un concetto centrale.

A secondo del tipo di dato contenuto il browser tratterà diversamente i valori delle variabili.

```
var x = 10 + 5;      // x vale 15
var x = "10" + "5"; // x vale "105"
```

Contrariamente ad altri linguaggi il tipo dei dati delle variabili si adatta automaticamente ai dati contenuti:

```
var x;                                  // il tipo di dati di x è undefined
x = 16;                                 // il tipo di dati di x è Number
x = "Ciao gente!";                      // il tipo di dati di x è String
x = {nome:"John", cognome:"Doe"}; // il tipo di dati di x è Object
```

TIPI DI DATI

L'operatore + consente sia di sommare numeri che di concatenare stringhe di caratteri. Quando mescolo i due tipi in una espressione i numeri saranno convertiti in stringa:

Operazione	Espressione	Conversione	Risultato
Somma numero con numero	`x = 5 + 5;`	Nessuna conversione	10
Somma numero in una stringa con numero	`x = '5' + 5;`	`x = '5' + '5'`	'55'
Somma una stringa con un numero	`x = 'Carlo'+ 5;`	`x = 'Carlo'+ '5';`	'Carlo5'

TIPI DI DATI

Bisogna comunque tener conto dell'ordine con cui vengono risolte le operazioni che compongono l'espressione. La conversione avverrà solo quando necessario:

Operazione	Espressione	Conversioni	Risultato
La somma tra numeri precede una somma con una stringa	`x = 5 + 10 + "Pippo";`	`x = 15 + "Pippo";` `x = "15" + "Pippo";`	"15Pippo"
La somma tra numeri segue una somma con una stringa	`x = "Pippo" + 5 + 10;`	`x = "Pippo" + '5' + '10';`	'Pippo510'
Uso delle parentesi	`x = "Pippo" + (5 + 10);`	`x = "Pippo" + 15;` `x = "Pippo" + "15";`	'Pippo15'

STRING

- Una stringa è una sequenza di caratteri che permette di rappresentare testi. Le stringhe sono racchiuse tra apici singoli o apici doppi.

```
/* String. Tra apici semplici o doppi */
x = 'Ciao da Javascript'; // Stringa tra apici semplici
x = "Ciao da Javascript"; // Stringa tra apici doppi
x = 'Il nome è "Pietro"'; // Stringa tra apici semplici (che contiene apici doppi)
x = "Il nome è 'Pietro'"; // Stringa tra apici semplici (che contiene apici semplici
```

- Per inserire ritorni a capo, tabulazioni, particolari caratteri o informazioni di formattazione si utilizzano speciali sequenze di caratteri dette *sequenze di escape*. Una sequenza di escape è formata da un carattere preceduto dal simbolo "\" (*backslash*). La sequenza di escape inserisce un carattere che non sarebbe altrimenti rappresentabile in una stringa.

PRINCIPALI SEQUENZE DI ESCAPE

\n nuova riga;
\r ritorno a capo;
\t tabulazione orizzontale;
\' apostrofo (o apice singolo);
\" doppio apice;
\\ backslash (essendo un carattere speciale deve essere inserito con una sequenza di escape).

NUMBERS

Contrariamente d altri linguaggi JavaScript ha un solo tipo di numeri.

I numeri possono essere scritti con, o senza decimali. Il separatore tra la parte intera e la parte decimale è il punto.

```
var x1 = 34.00;      // Con decimali
var x2 = 34;         // Senza decimali
```

Numeri molto grandi o molto piccoli possono essere scritti in notazione scientifica (esponenziale):

```
var y = 123e5;       // 1230000
var z = 123e-5;      // 0.00123
```

BOOLEAN

I dati di tipo boolean, poiché rappresentano valori logici, possono avere solo due valori: vero (rappresentato da **true**) e falso (rappresentato da **false**).

```
var x = true;
var y = false;
```

ARRAY

- Un *Array* è costituito da una serie di elementi ordinati identificabili dal loro indice. Si scrivono come elementi separati da virgole tra due parentesi quadre.

```
/* Giorni della settimana */
var giorni = ["Lunedì", "Martedì", "Mercoledì",
  "Giovedì", "Venerdì", "Sabato", "Domenica"];
```

- Gli indici sono a base zero, il che significa che il primo elemento è [0], secondo è [1], e così via.

OBJECT

- Un dato di tipo *Object* è compreso tra parentesi graffe ed è costituito da una serie di coppie "nome:valore" separate da virgole:

```
var persona = { nome: "Mario", cognome:"Rossi",
  telefono: "3338887777" }
```

- L'oggetto nell'esempio sopra ha 3 proprietà: nome, cognome e telefono.

typeof

- L'operatore **typeof** restituisce una stringa che rappresenta il tipo di una variabile o un'espressione. I risultati possibili sono:
  - `"string"`
  - `"number"`
  - `"boolean"`
  - `"function"`
  - `"undefined"`

```
typeof "John"                // "string"
typeof 3.14                  // "number"
typeof true                  // "boolean"
```

FUNZIONI

COSA È UNA FUNZIONE

- Una funzione (o metodo) è una costrutto presente in tutti i linguaggi di programmazione progettato per eseguire un compito particolare.
- Il codice contenuto in una funzione JavaScript viene eseguita quando "qualcosa" invoca la funzione.

SINTASSI

Una funzione JavaScript viene dichiarata (creata) con la parola chiave **`function`** seguita da una coppia di parentesi **`()`** e da un blocco di codice (serie di comandi separati da **`;`**) racchiuso tra parentesi graffe **`{}`**.

```
/* Funzione che calcola la media tra due numeri */
function (a, b) {
    return (a + b) / 2;
}
```

Le parentesi possono includere nomi di parametro separati da virgole: **`(parametro1, parametro2, ... )`**

FUNZIONE CON NOME

Posso creare una funzione con nome facendo seguire un identificativo alla parola chiave **function**.

```
/* Funzione che calcola la media tra due numeri */
function media(a, b) {
    return (a + b) / 2;
}
```

FUNZIONE ANONIMA

Posso assegnare una funzione a una variabile come un qualsiasi altro valore.

```
/* Funzione che calcola la media tra due numeri */
var media = function (a, b) {
    return (a + b) / 2;
}
```

FUNZIONE

In entrambi i casi media conterrà la funzione e, dal punto di vista funzionale il risultato sarà in molti casi identico.

Con **parametri** della funzione si indicano i **nomi** elencati tra parentesi nella definizione.

Con **argomenti** della funzione si indicano i **valori** ricevuti dalla funzione, tramite i parametri, quando viene richiamata (eseguita).

All'interno del corpo della funzione, i parametri si comportano come variabili locali.

INVOCAZIONE (ESECUZIONE)

Mentre il codice contenuto nelle altre parti di un programma JavaScript viene eseguito non appena il browser lo incontra, il codice all'interno della funzione verrà eseguito solo quando "qualcosa" esegue la funzione. Una funzione può essere *invocata* in tre modi:

- La funzione viene espressamente richiamata dal **codice**, come qualsiasi altro comando JavaScript
- La funzione viene associata ad un **evento** (ad esempio il click su un pulsante) e viene richiamata automaticamente ogni volta che l'evento si verifica
- Infine una funzione può essere **self invoked** (cioè richiamarsi da sola)

Nel primo e nel terzo caso (quando, cioè, la funzione non è eseguita automaticamente) ciò che *scatena* l'esecuzione è l'operatore di esecuzione, una coppia di parentesi tonde (), che seguono la funzione e che possono contenere gli argomenti (i valori) che la funzione elabora.

INVOCAZIONE (ESECUZIONE)

Il comando **return** nel corpo della funzione arresta l'esecuzione di una funzione.

Se la funzione viene richiamata da uno script, una volta eseguita, l'esecuzione del codice principale continua dal punto in cui la funzione è stata richiamata.

Le funzioni possono restituire un valore che normalmente è un calcolo o una elaborazione svolta sui valori passati ala funzione attraverso i parametri. Il valore di ritorno viene "restituito" al "chiamante" con il comando **return**.

```
/* Funzione che calcola la media tra due numeri */
  function media (a, b) {
      return (a + b) / 2;
  }
  // Calcolo la media tra 34 e 78
  var m = media(34, 78);
  // E comunico il risultato cambiando il contenut html dell'elemento con id="demo"
  document.getElementById('demo').innerHTML = "La media è " + m;
```

UTILITÀ DELLE FUNZIONI

- L'uso di funzioni ha due vantaggi:
  - È possibile riutilizzare il codice: Definire il codice di una volta, e utilizzarlo più volte.
  - È possibile utilizzare lo stesso codice con argomenti diversi, per produrre risultati diversi.

L'OPERATORE ()

- Per richiamare (eseguire) la funzione devo fare seguire il nome della funzione dall'operatore ( ) che può contenere l'elenco degli argomenti che passo alla funzione, se previsti.
- La funzione non seguita dall'operatore di esecuzione restituisce il contenuto della funzione stessa (la definizione) anziché il risultato