JAVASCRIP

Oggetti. Programmazione condizionale: il costrutto if...else

OGGETTI

FRULLATORE

- Caratteristiche
  - marca: SMEG
  - nome: Frullatore Rosso
  - volume: 1,5
  - potenza: 800
- Azioni:
  - start
  - stop
  - riempi
  - regola

OGGETTI

- Nella vita reale, un frullatore è un **oggetto** .
- Un frullatore ha **proprietà** (caratteristiche) come capacità e colore,
- e **metodi** (azioni) come stop e start
- Tutti i frullatori hanno le stesse caratteristiche , ma i valori delle proprietà differiscono da un'auto all'altra.
- Tutti i frullatori hanno gli stessi metodi , ma i metodi vengono eseguiti in tempi diversi.

OGGETTI JAVASCRIPT

Questo codice assegna un **valore semplice** (Frullatore rosso) a una **variabile** chiamata frullatore:

```
var frullatore = "Frullatore rosso";
```

Quando devo memorizzare ed elaborare un qualcosa di complesso che non può essere ridotto ad un valore semplice utilizzo un oggetto (Object).

Questo codice assegna più valori ("Frullatore rosso", "SMEG", 800, 1.5) alla variabile frullatore:

```
var frullatore = { nome:"Frullatore rosso", marca:"SMEG", volume: 1.5, potenza: 800};
```

I valori vengono scritti come un elenco separato da virgole di coppie **nome: valore** inserito tra due parentesi graffe.

Gli oggetti JavaScript sono un insieme di valori che vengono identificati attraverso un nome.

PROPRIETÀ

- Le coppie nome:valore (negli oggetti JavaScript) sono chiamate **proprietà**.

```
var persona = {
   nome:"Mario",
   cognome:"Rossi",
   anni:50,
   capelli:"biondi"
};
```

METODI

- I metodi sono **azioni** che possono essere eseguite sugli oggetti.
- I metodi sono memorizzati in proprietà come **definizioni di funzioni**.

```
var persona = {
    nome:"Mario",
    cognome:"Rossi",
    anni:50,
    capelli:"biondi",
    nomeCompleto: function() {
            return this.nome + " " + this.cognome
    }
};
```

DEFINIZIONE

- Si definisce (si crea) un oggetto JavaScript assegnando ad una variabile un oggetto letterale.
- Spazi e interruzioni di riga non sono importanti. La definizione di un oggetto può estendersi su più righe.

ACCESSO ALLE PROPRIETÀ

È possibile accedere alle proprietà degli oggetti in due modi:

- Usando l'operatore punto (.)

```
persona.nome;
```

- oppure utilizzando l'operatore []

```
persona["nome"];
```

ACCESSO AI METODI

I metodi non sono altro che proprietà che contengono la definizione di una funzione:

- Per eseguire un metodo quindi devo utilizzare l'operatore di esecuzione **()**

```
var nome = persona.nomeCompleto();
```

- Se si accede al metodo fullName , senza **()**, verrà restituita la **definizione della funzione**

OGGETTI E TIPI PREDEFINITI

DEFINIRE UNA VARIABILE

parola chiave (direttiva)

separatore

```
var pippo;
```

Identificatore (nome della variabile)

ASSEGNARE UN VALORE

identificatore (nome della variabile)

costante stringa

separatore

```
pippo = "Ciao gente!";
```

operatore (assegnazione)

ASSEGNARE UN VALORE

identificatore (nome della variabile)

classe (prototipo dell'oggetto)

parentesi

```
pippo = new Date();
```

operatore (assegnazione)

operatore (creazione di un oggetto)

OGGETTO e CLASSE

- L'oggetto è una grandezza informatica in grado di rappresentare i dati.
- Nella programmazione **OOP** tutte i dati si rappresentano e si elaborano tramite oggetti.
- Quando scrivo:

```
pippo = new Date();
```

- o:

```
pippo = "Ciao gente!";
```

- non assegno semplicemente un valore a una variabile. Creo un dato complesso detto oggetto che oltre al valore contiene tutti gli strumenti necessari ad elaborarlo.

OGGETTO e CLASSE

- Nella programmazione **OOP** i tipi di oggetto (il **tipo** è, in generale, determinato dal tipo di dati che l'oggetto è destinato ad elaborare) si chiamano **classi**.
- La classe, cioè, è l'insieme di regole che determinano come funziona un oggetto di una determinato tipo (o, i termini OOP, un *istanza di una determinata classe*).
- In Javascript non si fa menzione espressa del temine class (che per altro è una parola riservata che non può essere usata dall'utente). Per noi quindi **oggetto** (inteso come tipo di oggetto) e **classe** saranno sinonimi.
- Esistono oggetti (classi) predefinite dal linguaggio, oggetti (classi) create dalla comunità dei programmatori e che posso caricare e utilizzare nelle mie pagine Web e, infine, noi stessi possiamo creare le nostre classi per risolvere i nostri problemi.

TITPI DI BASE E TIPI COMPLESSI

- Tipi di base (o tipi semplici):
  - **Number**: numeri interi e numeri con parte decimale
  - **String**: Stringhe di caratteri (testo)
  - **Boolean**: Vero e falso
- Tipi complessi:
  - **Array**: Lista indicizzata di valori.
  - **Date**: Date.
  - **RegExp**: *Regular Expression.*
- E infine l'oggetto **Object** che rappresenta un OGGETTO generico. Con Object posso rappresentare una struttura dati qualsiasi e costruire dei tipi di dati utente, le mie classi.

OGGETTI STATICI

- Oggetti predefiniti che hanno solo metodi e proprietà statiche.
  - **Math**: libreria di funzioni matematica
  - **JSON**: Trasformazione di oggetti nella loro rappresentazione stringa e viceversa
  - **localStorage**: Metodi per gestire la memorizzazione di informazioni sul dispositivo dell'utente.

PROPRIETÀ E METODI

- Ogni tipo di dato (classe) è caratterizzato da:
- Proprietà che ci consentono di leggere o modificare determinate caratteristiche di un individuo della classe
- Metodi che ci mettono a disposizione determinate **azioni** che gli oggetti possono compiere o subire

PROPRIETÀ

- Le proprietà contengono un valore (come le variabili) che rappresenta una caratteristica dell'oggetto a cui la proprietà è riferita.
- Inizializzando una varibile le assegno un tipo (o classe)

```
pippo = "Ciao gente!";
```

- E posso accedere alle PROPRIETÀ E AI METODI DEFINITI DA QUEL TIPO.

ACCESSO ALLE PROPRIETÀ

- Usando l'operatore di appartenenza "."

```
var len = pippo.length;
```

- Usando le parentesi quadre che racchiudano il **nome della proprietà come stringa di caratteri**.

```
var len = pippo["length"];
```

- In entrambi i casi `len` conterrà il numero di caratteri di cui è composta la variabile stringa `pippo`

ACCESSO ALLE PROPRIETÀ

- Alcune proprietà sono a sola lettura

```
pippo.length = 3;
```

è un errore perché la proprietà **`length`** è a sola lettura.

Mentre invece:

```
elemento.innerHTML = "Ciao gente!";
```

modifica il contenuto HTML dell'elemento della pagina **elemento** (che un elemento HTML definito nella pagina).

METODI

- I metodi contengono codice eseguibile (come le funzioni), e ci consentono di eseguire azioni sull'oggetto.
- Una volta che una variabile contiene un oggetto

```
pippo = new Date();
```

- Posso accedere ai suoi metodi.
- Come le funzioni i metodi possono ricevere parametri e ritornare valori. Normalmente quando è previsto il passaggio di parametri i metodi modificano l'oggetto quando restituiscono valore restituiscono informazioni sull'oggetto.

ACCESSO A UN METODO

- Uso l'operatore di appartenenza "**.**" e faccio seguire l'identificatore dalle parentesi tonde

```
var giorno = pippo.getDate();
```

il metodo **getDate** restituisce il giorno del mese della data contenuta in **pippo** che viene memorizzato nella variabile **giorno**.

- Quando un metodo prevede dei parametri normalmente serve a modificare l'oggetto:

```
pippo.setDate(3);
```

Modificherò la data contenuta nella variabile **pippo:** il giorno del mese sarà 3.

PROPRIETÀ E METODI STATICI

- Esistono metodi e proprietà particolari che non si applicano ai singoli oggetti (istanze de una classe) ma hanno un uso globale.
- Per usarli non li applico all'oggetto ma alla classe

```
var data = Date.parse("March 21, 2012");
```

- La variabile **`data`** contiene l'oggetto Date ottenuto convertendo in data la stringa passata come parametro al metodo statico **`Date.parse`**

L'OGGETTO **document**

- Quando il browser carica la pagina web crea automaticamente l'oggetto **document**. **document** è una variabile riservata che contiene l'insieme di tutti gli elementi HTML definiti nella pagina.
- La maggior parte dell'attività di programmazione in javascript consiste nell'interagire con l'oggetto **document**: aggiungere elementi, estrarre elementi e modificarli, rispondere con azioni agli eventi provocati dall'utente e *subiti* dagli elementi interattivi.
- Come per gli altri oggetti l'accesso alle proprietà e ai metodi avviene attraverso l'operatore di appartenenza (.)
- Se, per esempio, voglio recuperate l'elemento con l'attributo id = mio_id:

```
var elemento = document.getElementById('mio_id');
```

COSTRUCTOR

- È la particolare metodo che consente di creare un oggetto di un particolare tipo (un'istanza di una classe):
  - Implicito:

```
var str = "Ciao!";
```

  - Esplicito:

```
var adesso = new Date();
```

  - Altra funzione o metodo:

```
var elemento = document.createElement("p");
```

NUMBER

CONSTRUCTOR

```
var n = 5;
var n = 10.6;
```

Proprietà statiche

Proprietà	Descrizione
`Number.MAX_VALUE;`	Restituisce il massimo numero consentito in JavaScript
`Number.MIN_VALUE;`	Restituisce il minimo numero consentito in JavaScript
`Number.NEGATIVE_INFINITY;`	Rappresenta l'infinito negativo.
`Number.POSITIVE_INFINITY;`	Rappresenta l'infinito positivo.

METODI (CONVERSIONI IN STRING)

Metodo	Descrizione
`toExponential(x)`	Reastituisce una stringa, che rappresenta il numero come notazione esponenziale dove x (opzionale) indica il numero dei decimali da usare
`toFixed(x)`	Converte il numero in una stringa, con x numero di decimali. Se x non viene specificato nessun decimale.
`toPrecision(x)`	Converte il numero in una stringa di lunghezza x. Se necessario vengono aggiunti punto decimale e 0.
`toString(base)`	Converte il numero in una stringa secondo la base specificata da base. La base di default è 10 (numero decimale). Se base vale 2 si ottiene la rappresentazione binaria del numero, se 16 quella esadecimale, ecc.

```
/*=============================================================
                    USO DEI METODI DI NUMBER
  ============================================================= */
var num = 1289.2;                                   // Numero decimale
var numIntero = 65000;                              // Numero intero
var str;
str = num.toExponential();                          // "1.2892e+3"
str = num.toExponential(5);                         // "1.28920e+3"
str = num.toFixed();                                // "1289"
str = num.toFixed(3);                               // "1289.200"
str = num.toPrecision()                             // "1289.2"
str = num.toPrecision(8)                            // "1289.2000"
str = num.toString()                                // "1289.2"
str = numIntero.toString(16)                        // "fde8"
str = numIntero.toString(2)                         // "1111110111101000"
str = num.toString(16)                              // "509.33333333334"
```

STRING

CONSTRUCTOR

```
var str = "Ciao!";
```

PROPRIETÀ

- Gli oggetti della classe String hanno una sola proprietà, la proprietà **`length`** che restituisce la lunghezza della stringa, cioè il numero di caratteri di cui è composta.

MANIPOLAZIONE

Metodo	Descrizione
`charAt(pos)`	Restituisce il carattere alla posizione pos
`charCodeAt(pos)`	Restituisce il codice Unicode corrispondente al carattere alla posizione pos
`concat(s1, s2)`	Concatena due stringhe (stesso risultato di s1 + s2)
`fromCharCode(code)`	Restituisce il carattere corrispondente al valore unicode code
`indexOf(searchstring, start)`	Restituisce la posizione della prima occorrenza della stringa searchstring in una stringa (-1 se non lo trova). Opzionalmente la ricerca può partire dalla posizione start.
`lastIndexOf(searchstring, start)`	Restituisce la posizione dell'ultima occorrenza della stringa searchstring in una stringa (-1 se non lo trova). Opzionalmente la ricerca può partire dalla posizione start in vece che dall'ultimo carattere.
`match(regexp)`	Il metodo match cerca le corrispondenza e tra l'espressione regolare regexp e la stringa, e restituisce un array di corrispondenze. Se non viengono trovate corrispondenze viene restituito null.
`replace(regexp/substr, newstring)`	Replace() cerca una corrispondenza tra una stringa (o un'espressione regolare) e una stringa, e sostituisce la corrispondenze trovate con newstring

MANIPOLAZIONE

Metodo	Descrizione
`search(regexp)`	Il metodo search cerca le corrispondenza e tra l'espressione regolare regexp e la stringa, e restituisce la posizione in cui è stata trovata oppure -1 se non viengono trovate corrispondenze.
`slice(inizio, fine)`	Estrae la parte di una stringa compresa tra inizio e fine e restituisce la parte estratta in una nuova stringa. In caso non sia passato un valore, fine sarà l'ultimo carattere della stringa.
`split(char)`	Converte la stringa in un array usando char come carattere di separazione.
`substr(start, length)`	Estrae lentgth caratteri dalla stringa, a partire da start e li restituisce in una nuova stringa. Se length non è specificati vengono restituiti i caratteri da start fino alla fine della stringa.
`substring(from, to)`	Estrae i caratteri delle stringa tra from e to non compreso. Se to è omesso fino alla fine della stringa.
`toLowerCase()`	Converte in minuscolo
`toUpperCase()`	Converte in maiuscolo

replace()

- Il metodo replace () cerca in una stringa i gruppi di caratteri che corrispondono a una substringa o a una *regular expression*, e restituisce una nuova stringa in cui le occorrenze trovate vengono sostituite da un valore specificato,
- Nota: Se si usa una substring e non una regular expression per la ricerca, verrà sostituita solo la prima occorrenza trovata. Per sostituire tutte le occorrenze di una ricerca è necessario usare una regular expression con il modificatore g,
- Questo metodo non cambia la stringa originale.

split()

- Il metodo split () viene utilizzato per dividere una stringa in un array di stringhe utilizzanda uno o più caratteri come separatore, e restituisce il nuovo array.
- Se si utilizza una stringa vuota ("") come separatore, la stringa è diviso nei singoli caratteri che la compongono.
- Nota: Il metodo split () non modifica la stringa originale.

```
/*=============================================================
                    USO DEI METODI DI STRING
  ============================================================= */
var origine = "Ciao gente!";
var origine2 ="Domenica,Lunedì,Martedì,Mercoledì,Giovedì,Venerdì,Sabato";
var str, pos, len, giorni;
len = origine.length;                    // 11
str = origine.charAt(3);                 // "o"
str = origine.indexOf("e");              // 6
str = origine.indexOf("!!");             // -1 (non trovato)
str = origine.lastIndexOf("e");          // 9
str = origine.replace("!", "?");         // "Ciao gente?"
giorni = origine2.split(",");            // ["Domenica", "Lunedì", "Martedì",
                                         // "Mercoledì", "Giovedì",
                                         // "Venerdì", "Sabato"]
str = origine.substr(5)                  // "gente!"
str = origine.substr(5,2)                // "ge"
```

ARRAY

CONSTRUCTOR

```
var a = [1,6,78,23];
var a = new Array(1,6,78,23);
```

PROPRIETÀ

- Gli oggetti della classe Array hanno una sola proprietà, la proprietà **`length`** che restituisce la lunghezza dell'array, cioè il numero di elementi di cui è composto.

METODI

Metodo	Descrizione
`concat( array2,array3, arrayX)`	Unisce uno o più array all'array a cui il metodo è applicato, e restituisce una copia degli array così uniti.
`indexOf( elemento, start)`	Cerca elemento in un array partendo da start (o dall'inizio se start è omesso) e ne restituisce la posizione. Se start è negativo indica la posizione reativa alla fine dell'array.
`join(separatore)`	Unisce gli elementi di un array in una stringa, e restituisce la stringa. Gli elementi sono separati da separatore. Il separatore di default è la virgola .
`lastIndexOf(elemento, start)`	Cerca l'ultima ricorrenza di elemento in un array partendo da start (o dall'iniziose start è omesso) e ne restituisce la posizione o -1 se elemento non viene trovato.
`pop()`	Rimuove l'ultimo elemento di un array, e restituisce l'elemento rimosso.
`push(elemento)`	Aggiunge elemento alla fine dell'array e restituische la nuova lunghezza.

METODI

Method	Description
`reverse()`	Inverte l'ordine degli elementi dell''array.
`shift()`	Rimuove il primo elemento di un array, e restituisce l'elemento rimosso.
`slice( inizio, fine)`	Estrae gli elementi a partire da inizio, fino a fine, non incluso e li restituisce in un nuovo array. L'array originale non viene modificato.
`sort( sortfunct)`	Ordina gli elementi di un array (alfabetico ascendente) o usa sortfunct per stabilire l'ordine
`splice(indice, quanti, item1, itemX)`	Rimuove quanti elementi dall'array a partire dalla posizine indice e inserisce gli elementi item1, ..., itemX (se forniti) a partire dalla posizine indice. Restituisce gli elementi rimossi.
`toString()`	Restituisce l'array convertito in stringa.
`unshift(elemento)`	Aggiunge elemento all'inizio dell'array e restituische la nuova lunghezza

```
/*
      rubrlca è un array di oggetti che hanno due proprietà: nome e cognome.
*/
var rubrica = [
      {nome:"Mario", cognome:"Rossi" },
      {nome:"Luigi", cognome:"Neri" },
      {nome:"Piero", cognome:"Verdi" },
      {nome:"Mario", cognome:"Bianchi" }
];
/*
      Per ordinare l'array definisco una funzione che stabilisce il criterio
      con cui vengono confrontati gli elementi dell'array e ritorna un valore
      positivo se è maggiore il primo elemento, 0 se gli elementi sono uguali
      e negativo se è maggiore il secondo elemento.
      La funzione che segue ordina gli elementi in rubrica per cognome in
      ordine crescente.
*/

var sortCognome = function (a,b){
                                                if (a.cognome > b.cognome){
                                                      return 1;
                                                } else if (a.cognome == b.cognome){
                                                      return 0;
                                                } else {
                                                      return 1;
                                                }
                                    };
//Ordino l'array secondo il criterio definito dalla funzione sortCognome
rubrica.sort(sortCognome);
```

DATE

CONSTRUCTOR

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
var d = new Date(year, month, day, hours,
                 minutes, seconds,
                 milliseconds);
```

Metodo statico

Date.parse(str)	Analizza una data in formato stringa e restituisce il numero di millisecondi dalla mezzanotte del 1 Gennaio 1970.

Metodo	Descrizione
`getDate()`	Restituisce il giorno del mese (1-31)
`getDay()`	Restituisce il giorno della settimana (0-6, 0 = domenica)
`getFullYear()`	Restituisce l'anno (quattro cifre)
`getHours()`	Restituisce l'ora (da 0-23)
`getMilliseconds()`	Restituisce i millisecondi (0-999)
`getMinutes()`	Restituisce i minuti (0-59)
`getMonth()`	Restituisce il mese (0-11)
`getSeconds()`	Restituisce i secondi (0-59)
`getTime()`	Restituisce il numero di millisecondi trascorsi dalla mezzanotte del 1 gennaio 1970
`getTimezoneOffset()`	Restituisce la differenza di tempo tra il GMT e l'ora locale
`getUTCDate()`	Restituisce il giorno del mese, in base all'ora universale (da 1-31)
`getUTCDay()`	Restituisce il giorno della settimana, in base all'ora universale (da0-6)
`getUTCFullYear()`	Restituisce l'anno, in base all'ora universale (quattro cifre)
`getUTCHours()`	Restituisce l'ora, in base all'ora universale (da 0-23)
`getUTCMilliseconds()`	Restituisce i millisecondi, in base all'ora universale (0-999)
`getUTCMinutes()`	Restituisce i minuti, in base all'ora universale (da 0-59)
`getUTCMonth()`	Restituisce il mese, in base all'ora universale (da 0-11)
`getUTCSeconds()`	Restituisce i secondi, in base all'ora universale (da 0-59)

Metodi	Descrizione
`setDate()`	Imposta il giorno del mese di un oggetto data
`setFullYear()`	Imposta l'anno (quattro cifre) di un oggetto data
`setHours()`	Imposta l'ora di un oggetto data
`setMilliseconds()`	Imposta i millisecondi di un oggetto data
`setMinutes()`	Impostare i minuti di un oggetto data
`setMonth()`	Imposta il mese di un oggetto data
`setSeconds()`	Imposta i secondi di un oggetto data
`setTime()`	Consente di impostare una data e un'ora aggiungendo o sottraendo un determinato numero di millisecondi a partire dalla mezzanotte del primo gennaio 1970
`setUTCDate()`	Imposta il giorno del mese di un oggetto data, in base all'ora universale
`setUTCFullYear()`	Imposta l'anno di un oggetto data, in base all'ora universale (quattro cifre)
`setUTCHours()`	Imposta l'ora di un oggetto data, in base all'ora universale
`setUTCMilliseconds()`	Imposta i millisecondi di un oggetto data, in base all'ora universale
`setUTCMinutes()`	Impostare i minuti di un oggetto data, in base all'ora universale
`setUTCMonth()`	Imposta il mese di un oggetto data, in base all'ora universale
`setUTCSeconds()`	Impostare i secondi di un oggetto data, in base all'ora universale
`setDate()`	Imposta il giorno del mese di un oggetto data
`setFullYear()`	Imposta l'anno (quattro cifre) di un oggetto data
`setHours()`	Imposta l'ora di un oggetto data

Metodi	Descrizione
`toDateString()`	Converte la parte relativa alla data di un oggetto Date in una stringa leggibile
`toISOString()`	Restituisce la data come una stringa, utilizzando lo standard ISO
`toJSON()`	Restituisce la data come una stringa, formattato come una dataJSON
`toLocaleDateString()`	Restituisce la parte relativa alla data di un oggetto Date come una stringa, utilizzando le convenzioni di localizzazione
`toLocaleTimeString()`	Restituisce la parte di ora di un oggetto Date come una stringa, utilizzando le convenzioni di localizzazione
`toLocaleString()`	Converte un oggetto Date in una stringa, utilizzando le convenzioni di localizzazione
`toString()`	Converte un oggetto Date in una stringa
`toTimeString()`	Converte la parte ora di un oggetto Date in una stringa
`toUTCString()`	Converte un oggetto Date in una stringa, in base all'ora universale
`UTC()`	Restituisce il numero di millisecondi in una stringa data a partire dalla mezzanotte del 1 gennaio 1970, in base all'ora universale

MATH

PROPRIETÀ

Proprietà	Descrizione
`Math.E`	Restituisce il numero di Eulero (circa 2,718)
`Math.LN2`	Restituisce il logaritmo naturale di 2 (circa 0,693)
`Math.LN10`	Restituisce il logaritmo naturale di 10 (circa 2,302)
`Math.LOG2E`	Restituisce il logaritmo in base 2 di E (circa 1,442)
`Math.LOG10E`	Restituisce il logaritmo in base 10 di E (circa 0,434)
`Math.PI`	Restituisce PI (3.1416....)
`Math.SQRT1_2`	Restituisce la radice quadrata di 1/2 (circa 0,707)
`Math.SQRT2`	Restituisce la radice quadrata di 2 (circa 1,414)

METODI

Method	Description
`Math.abs(x)`	Restituisce il valore assoluto di x
`Math.acos(x)`	Restituisce l'arcocoseno di x, in radianti
`Math.asin(x)`	Restituisce l'arcoseno di x, in radianti
`Math.atan(x)`	Restituisce l'arcotangente di x come un valore numerico compreso tra-PI / 2 e PI 2 radianti
`Math.atan2(y,x)`	Restituisce l'arcotangente del quoziente dei suoi argomenti
`Math.ceil(x)`	Restituisce X arrotondato per eccesso al numero intero più vicino
`Math.cos(x)`	Restituisce il coseno di x (x è in radianti)
`Math.exp(x)`	Restituisce il valore di E elevato alla x
`Math.floor(x)`	Restituisce X arrotondato per difetto al numero intero più vicino

METODI

Method	Description
`Math.log(x)`	Restituisce il logaritmo naturale (base e) di x
`Math.max(x,y,z,...,n)`	Restituisce il numero con il valore più alto
`Math.min(x,y,z,...,n)`	Restituisce il numero con il valore più basso
`Math.pow(x,y)`	Restituisce il valore di x alla potenza y
`Math.random()`	Restituisce un numero casuale compreso tra 0 e 1
`Math.round(x)`	Arrotonda x al numero intero più vicino
`Math.sin(x)`	Restituisce il seno di x (x è in radianti)
`Math.sqrt(x)`	Restituisce la radice quadrata di x
`Math.tan(x)`	Restituisce la tangente di un angolo

PRENDERE DECISIONI

LE STRUTTURE DI CONTROLLO

- Le strutture di programmazione che mi consentono di prendere decisioni sono essenzialmente due:
  - **condizionale**: faccio una determinata cosa se una condizione risulta vera altrimenti ne faccio un'altra
  - **iterativa** (o loop): ripeto una determinata operazione finche una condizione risulta vera

L'ISTRUZIONE IF

SINTASSI DELL'ISTRUZIONE IF

- L'istruzione if può avere due forme:
  - if (espressione) blocco di istruzioni
  - if (espressione) blocco di istruzioni else blocco di istruzioni
- L'espressione che compare dopo la parola chiave if deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'else (senza condizioni) in posizione finale.

BLOCCO IF

```
if (condizione)
{
    //comandi se condizione è vera
}
// il programma continua qui
```

BLOCCO IF ELSE

```
if (condizione)
{
    //comandi se condizione è vera
}
else
{
    //comandi se condizione è falsa
}
// il programma continua qui
```

SERIE DI CONDIZIONI

```
if (condizione1)
{
      //comandi se condizione1 è vera
}
else if (condizione2)
{
      //comandi se condizione2 è vera
}
else if (condizione3)
{
      //comandi se condizione3 è vera
}
else
{
      //comandi se tutte le condizioni sono false
}
// il programma continua qui
```

GLI OPERATORI LOGICI

operazione	javascript	precedenza
uguaglianza	== oppure ===	1
disuguaglianza	!= oppure !==	1
maggiore	>	1
maggiore o uguale	>=	1
minore	<	1
minore o uguale	<=	1
and	&&	2
or	||	2
not	!	2

COMBINARE LE CONDIZIONI

- Prendiamo questi enunciati:
  - esco se il tempo è bello ed è caldo
  - esco se il tempo è bello o è caldo
  - non esco se il tempo non è bello e non è caldo
  - non esco se il tempo non è bello o non è caldo

LE TABELLE DI VERITÀ
esco se il tempo è bello **ed** è caldo

enunciato 1	congiunzione	enunciato 2	risultato
tempo bello	ed	temperatura caldo	esco?
`true`	`&&`	`true`	`true`
`false`	`&&`	`true`	`false`
`true`	`&&`	`false`	`false`
`false`	`&&`	`false`	`false`

LE TABELLE DI VERITÀ

esco se il tempo è bello **o** è caldo

enunciato 1	congiunzione	enunciato 2	risultato
tempo bello	o	temperatura caldo	Esco ?
`true`	`||`	`true`	`true`
`false`	`||`	`true`	`true`
`true`	`||`	`false`	`true`
`false`	`||`	`fasle`	`false`

LE TABELLE DI VERITÀ

non esco se il tempo **non** è bello **e non** è caldo

	enunciato 1	congiunzione	enunciato 2	risultato
non	tempo è bello	e	temperatura è caldo	Esco ?
!	`true`	&&	`true`	`true`
!	`true`	&&	`false`	`true`
!	`false`	&&	`true`	`true`
!	`false`	&&	`false`	`false`

LE TABELLE DI VERITÀ

non esco se il tempo **non** è bello **o non** è caldo

	enunciato 1	congiunzione	enunciato 2	risultato
non	tempo è bello	o	temperatura è caldo	Esco ?
!	`true`	`||`	`true`	`true`
!	`true`	`||`	`false`	`false`
!	`false`	`||`	`true`	`false`
!	`false`	`||`	`false`	`false`

FORM

FORM

Il tag <form> viene utilizzato per creare un modulo HTML per l'input dell'utente.
L'elemento <form> può contenere uno o più dei seguenti elementi del modulo:

```
<input>
<textarea>
<button>
<select>
<option>
<optgroup>
<fieldset>
<label>
```

```
<form action="demo_form.asp" method="get">
    Nome: <input type="text" name="nome"><br>
    Cognome: <input type="text" name="cognome"><br>
    <input type="submit" value="Invia">
</form>
```

INPUT

- Il tag `<input>` specifica un campo di input in cui l'utente può inserire i dati.
- Gli elementi `<input>` vengono utilizzati all'interno di un elemento `<form>`
- La funzione di imput cambia a secondo del valore definito dall'attributo **`type`**:
  - Pulsante **`(button, submit, reset),`** casella selezionabile **`(checkbox, radio),`** casella di testo **`(text),`** caricamento file **`(file),`** valore nascosto **`(hidden)`**
- L'attributo `placeholder` inserisce nel campo un suggerimento che scompare automaticamente quando l'utente inserisce un contenuto
- L'attributo `value` definisce il *valore* del controllo e ha significati diversi a seconda del del contenuto dell'attributo `type`:
  - Rappresenta il contnuto iniziale del campo quando input è una casello di testo
  - Rappresenta il valore del campo quando è selezionato per i tipi **`checkbox`** e **`radio`**
  - Rappresenta l'etichetta del pulsante quando input è un pulsante

```
<form action="demo_form.asp" method="get">
      Nome: <input type="text" name="nome" placeholder="Inserisci il tuo nome"><br>
      Cognome: <input type="text" name="cognome" placeholder="Inserisci il tuo cognome"><br>
      <input type="submit" value="Invia">
</form>
```

TEXTAREA

- Il tag `<textarea>` definisce un controllo di input di testo multilinea.
- Un'area di testo può contenere un numero illimitato di caratteri.
- Per default il testo viene reso in un font a larghezza fissa (di solito Courier).
- L'attributo placeholder inserisce nel campo un suggerimento che scompare automaticamente quando l'utente inserisce un contenuto

```
<textarea name="message" rows="4" cols="50"
          placeholder="Inserisci il tuo messaggio">
          Testo che appare nel controllo
</textarea>
```

BUTTON

- Il tag `<button>` definisce un pulsante cliccabile.
- Contrariamente che per l'elemento input all'interno di un elemento `<button>` Posso inserire qualsiasi tipo di contenuto.
- Browser diversi (e sistemi operativi diversi) utilizzano diversi stili di default per l'elemento `<button>`.

```
<button type="button">Cliccami!</button>
```

SELECT

- L'elemento `<select>` viene utilizzato per creare un elenco a discesa.
- Gli elementi `<option>` all'interno dell'elemento `<select>` definiscono le opzioni disponibili nella lista.
- La proprietà **`value`** dell'elemento `<select>` indica il valore (attributo **`value`**) della `<option>` selezionata dall'utente

```
<select>
   <option value="volvo">Volvo</option>
   <option value="saab">Saab</option>
   <option value="mercedes">Mercedes</option>
   <option value="audi">Audi</option>
</select>
```