Umberto Emanuele

JavaScript 1
Giorno 2

Novembre 2023

Operatori di comparazione

Typeof

Abbiamo già messo in evidenza che JavaScript gestisce la tipizzazione dinamica.

Può essere un operatore specifico per determinare il tipo di dato gestito in una variabile.

Typeof ci permette l'output del tipo di dato.

typeof "Mario" => string / dato stringa
typeof 3 => number / dato numerico
typeof true => boolean / dato booleano
typeof Mario => undefined
typeof "" => string /dato stringa

Operatori di uguaglianza e di identità

In programmazione identità e uguaglianza vengono identificati dai simboli === e ==

```
var numero = 1;
var numero2 = 3;
var numero3 = 1;
var nome = “Mario”;
```

Uguaglianza => indica che due valori comparati hanno lo stesso valore

```
numero == numero2; => false
numero == numero3; true
```

Identità (strict equality) => due valori non solo devono avere lo stesso valore ma anche il medesimo tipo

```
document.write(1 === “1”); => false
document.write(3 === 3); => true
document.write(“Mario” === “Mario”);
```

Operatori negativi di uguaglianza e di identità

In programmazione la mancanza di identità e di uguaglianza vengono identificati dai simboli

!== non identico

!= non uguale

```
var numero = 1;
var numero2 = 3;
var numero3 = 1;
```

```
numero2 != 3; => false
numero2 != 5; => true

numero2 !== 3; => false
numero2 !== “3”; => true
```

Operatori di maggiore e minore

Sono operatori fondamentali nella comparazione di due operandi e in associazione a risposte booleane

Maggiore >
Minore <
Maggiore uguale >=
Minore uguale <=

```
var numero = 3;
var numero1 = 5;
```

Esempi di output

```
10 < 8; => false
10 <= 10; => true
5 > 5; => false
5 >= 5; => true

numero < numero1; => true
numero1 >= numero; => true;
```

Operatore ternario

Attraverso l'operatore ternario possiamo attribuire ad una variabile un valore basato su una certa condizione.

```
var ingresso = (anni < 18) ? "ingresso vietato " : "ingresso permesso";
```

```
anni = 15;
output => ingresso vietato

anni = 20;
output => ingresso permesso

anni = 18;
output => ingresso vietato
```

Le funzioni: basi

La funzione è una porzione di codice, strutturalmente delimitata, che esegue le istruzioni definite al suo interno.

```
function miaFunzione()
{
// istruzioni da eseguire
}
```

Il nome della funzione è preceduto dalla keyword dedicata **function** e seguito dalle parentesi tonde.

Nelle parentesi tonde vengono specificati i **parametri** della funzione.

Anche quando non sono presenti parametri, le parentesi tonde sono necessarie.

Il corpo della funzione è delimitato dalle parentesi graffe che circoscrivono tutto il codice che dovrà essere eseguito.

Definire e invocare una funzione

Definire una funzione significa nominarla e indicare le istruzioni da eseguire.

```
function miaFunzione()
{
return 3 + 2;
}
```

Invocare una funzione significa richiamarla attraverso il suo nome, in un punto del programma perché possa essere eseguita.

Se non invocata, la funzione non ha effetto sul codice.

Se la funzione possiede dei parametri, a questi viene assegnato un valore (**argomento**).

La funzione può essere invocata molteplici volte all'interno di una programma.

```
<button onclick="miaFunzione();">Clicca qui</button>
```

Il concetto di hoisting

Lo **hoisting** (sollevamento), indica la possibilità per JavaScript di eseguire una funzione prima che sia dichiarata.

Si parla di sollevamento perché è come se la definizione della funzione fosse portata al top del programma corrente.

L'hoisting è applicabile anche alle variabili.

Uno dei motivi più importanti per cui usare l'hoisting, è la maggiore leggibilità del codice.

A causa del hoisting è perciò possibile questa sintassi:

```
miaFunzione(10);

function miaFunzione(numero)
{
return numero + numero;
}
```

La keyword this

Questa keyword indica uno stretto riferimento all’ambito della funzione a cui appartiene.

Viene perciò utilizzato per definire un collegamento veloce con il valore indicato in un elemento della funzione.

Il suo uso avrà pieno senso nell’ambito del concetto di oggetto in JavaScript.

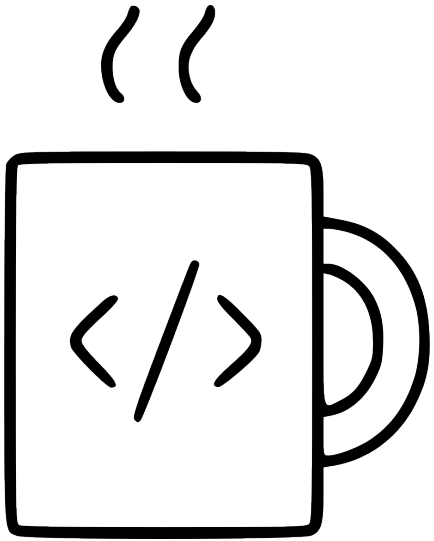
L’istruzione return

Fondamentalmente **return** restituisce il valore relativo all’esecuzione di una serie di istruzioni in una funzione.

Questa istruzione interrompe l’esecuzione ulteriore della funzione.

Per esempio:

```
function miaFunzione()
{
        return “Hello world”;
}
```



PAUSA

Ci vediamo alle ore 11.17

Le funzioni: parametri e closures

Funzioni con parametri

I **parametri** attribuiscono alla funzione la possibilità di manipolare dei valori e di utilizzarli per l'esecuzione di istruzioni.

Per esempio,

Dovendo ottenere la somma di due numeri, possiamo definire due parametri in una funzione:

```
function somma (numero1, numero2) {
   //istruzioni da eseguire
}
```

Definiamo la funzione:

```
function somma(numero1, numero2){
   return numero1 + numero2;
}
```

Invochiamo la funzione:

```
somma(4, 5); => 9
```

Ricapitolando:

- numero1 e numero2 sono i **parametri**
- 4 e 5 sono gli argomenti assegnati ai parametri

Regole dei parametri

- Il tipo di dato del parametro non deve essere specificato.
- JavaScript non ha il controllo sulla corrispondenza del tipo di dato tra parametro e argomento.
- JavaScript non ha il controllo sulla corrispondenza del numero di parametri e di argomenti passati alla funzione.

Le funzioni con parametro possono anche non avere argomenti.

In tal caso il valore o i valori mancanti sono di tipo undefined.

Questa cosa non comporta irregolarità, tuttavia sarebbe consigliabile impostare un valore di default al parametro.

```
function miaFunzione(a, b){
if(b === undefined){
b = 5;
}
}
```

In JavaScript una funzione può essere definita come un'espressione:

```
var somma = function(a, b){
    return a + b;
}
```

La variabile somma può essere usata come una vera e propria funzione invocandola:

```
somma(10, 5);
```

Le funzioni così definite vengono chiamate **anonime** .

Le funzioni possono essere utilizzate anche come valore di una variabile:

```
function somma(a, b){
    return a + b;
}
var sommaFunzione = somma(10,
5);
```

Le funzioni ad espressione possono anche essere auto invocate (self-invoking).
Ciò significa che dopo la definizione non devono essere invocate.

La sintassi prevede di racchiudere la funzione in ():

```
(function(){
var somma = 3 + 2;
}) ();
```

In relazione alle variabili le funzioni possono essere definite anche con la keyword **new** nel metodo costruttore Function():

```
var somma = new Function("x", "y", "return x + y");

somma(10,5);
var z = somma(10,5);
```

Nel corso avremo modo di approfondire e comprendere meglio questa particolare uso del metodo costruttore.

Scope (ambito) delle variabili

Lo scope delle variabili indica il loro livello di accessibilità o visibilità.

I livelli di accessibilità sono due:

- **Locale**
- Globale

Ogni funzione definita genera, rispetto alle variabili, un livello di accessibilità.

Se una variabile è creata all'interno di una funzione, il suo ambito è locale alla funzione.

Se la variabile è creata all'esterno e indipendentemente da una funzione, il suo ambito è globale.

```
var luogo = "Firenze";

function utente(){
        var nome = "mario";
        //la variabile nome è accessibile (locale)
        //la variabile luogo è accessibile (globale)
}
```

La variabile nome non è accessibile (locale)
La variabile luogo è accessibile (globale)

Una closure, ovvero chiusura, descrive un ambito che una funzione può richiamare in riferimento al contesto dalla quale proviene.

Nell'esempio la funzione **somma()** è innestata e fa closure nel contesto della funzione **operazioni()** a cui appartiene.

La chiusura in questo caso quindi è data dal fatto di poter accedere ad un ambito richiamato grazie ad un riferimento ad esso.

Esempi di output

```
function operazioni(){
    var numero = 10;
    function somma(){
        console.log(numero + numero);
    }
    return somma;
}
var numero1 = operazioni();
output di numero1() => 10
```

Funzioni a freccia Arrow functions

Con tale definizione si indica una metodologia sintattica che permette una scrittura più rapida e breve del corpo delle funzioni.

Se la funzione ha una sola istruzione e il return la sintassi è ancora più stringata:

```
saluto = () => “Hello world!”;
```

Esempi di output

```
unction saluto(){
return “Hello world!”;
}
```

La stessa funzione può essere scritta:

```
saluto = () => {return “Hello world!”}
```

I parametri possono ugualmente essere utilizzati:

```
saluto = (stringa) => “Hello world!” + “a tutti”;
```

Le stringhe: approfondimenti

Le stringhe

La stringa è una sequenza di caratteri all'interno di apici singoli o doppi.
Le stringhe sono usate per archiviare e manipolare i tipi di dati testuali.

Escape

Caratteri escape attualmente in uso:
\ => corretta interpretazione degli apici e degli apostrofi
\n => nuova linea
\b => backspace

Esempi:

var stringa = 'L\'esempio di stringa';
Var stringa = 'Questo \\ è un backslash';

Break delle linee di codice

Per buona pratica di scrittura del codice, non dovrebbero essere scritte linee di codice troppo lunghe.

In genere ci si attiene agli 80 caratteri.

Esiste quindi una convenzione di scrittura che prevede l'uso di \ o di + (concatenazione) per interrompere le linee

Esempi

```
document.getElementById("test").innerHTML = "Hello \ world!"; => sintassi corretta

document.getElementById("test").innerHTML = "Hello world!"; => sintassi non corretta
```

```
document.getElementById("test").innerHTML = "Hello world!"; => sintassi corretta

Metodo consigliato e preferibile:

document.getElementById("test").innerHTML = "Hello + world!"; => sintassi corretta
```

Le variabili come ogg

Le variabili sono normalmente dati primitivi ma in JavaScript possono assumere anche la definizione di oggetto.

Una variabile può definita oggetto con la keyword new.

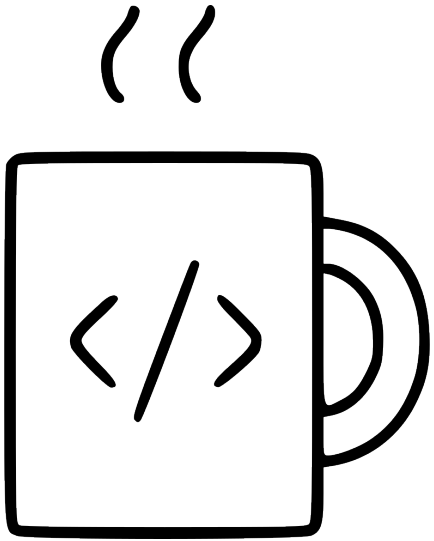
```
var nome = “Mario”; typeof => string

var nome1 = new String(“Mario”); typeof => object
```

L’uso delle variabili come oggetti è tuttavia raccomandata con parsimonia a causa delle maggiori risorse richieste.

Utilizzando gli operatori di comparazione:

nome == nome1; => true perché il valore è lo stesso

nome === nome1; => false perché i tipi di dati sono differenti



PAUSA

Ci vediamo alle ore 14.00

Gli array

Array: definizione

Sostanzialmente l'array è una variabile capace di archiviare e memorizzare una serie di valori anziché uno solo alla volta.

Nel definire l'array utilizziamo la stessa keyword **var**.

Un array appare come una vera e propria lista ed è utile proprio quando i dati da inserire nel nostro programma sono una serie ordinata di valori.

Esempio una lista di animali:

```
var animale1 = "gatto";
var animale2 = "cane";
var animale3 = "coniglio";
……..
```

Array: creazione e uso

Nel definire l'array utilizziamo la stessa keyword **var**.

Indichiamo i valori che formano la lista tramite parentesi quadre e separati da virgole.

La lista di variabili:

```
var animale1 = "gatto";
var animale2 = "cane";
var animale3 = "coniglio";
```

Possiamo indicarla come array:

```
var animali = ["gatto", "cane", "coniglio"];
```

Array: creazione e uso

Possiamo indicare un array con sintassi alternative che in alcuni casi rendono il codice più leggibile

```
var animali = [
“gatto”,
“cane”,
“coniglio”
];
```

```
var anni = [
1920,
1965,
2013
];
```

La virgola di separazione, nell’ultimo elemento non è necessaria.

Array: uso della Keyword new

Per creare un array possiamo usare la keyword new.

In questo caso la lista degli elementi è creata all'interno di parentesi tonde.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
var animali = new Array (
"gatto",
"cane",
"coniglio"
);
```

```
var anni = [
1920,
1965,
2013
];
```

```
var anni = new Array (
1920,
1965,
2013
);
```

Array come oggetti

Gli array in JS sono da considerarsi come oggetti.

L'output di typeof su un array object come tipo di dato anche la sua descrizione logica è di lista ordinata di elementi.

A questo proposito si evidenzia che in programmazione l'ordine inizia dallo 0.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
typeof(animali);

"object"
```

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
Ordine degli elementi
gatto          => 0
cane           => 1
coniglio       => 2
```

Array: accesso agli elementi singoli

La possibilità di ordinare gli elementi di un array ci permette di accedere ad essi singolarmente.

Per accedere agli elementi facciamo al **numero di indice** .

```
var tipoAnimale = animali[0];
console.log(tipoAnimale); => “gatto”

var tipoAnimale = animali[1];
console.log(tipoAnimale); => “cane”

var tipoAnimale = animali[2];
console.log(tipoAnimale); => “undefined”
```

```
var animali = [
“gatto”,
“cane”,
];
```

Array: accesso agli elementi singoli

All'interno di un documento possiamo accedere agli elementi dell'array selezionando un elemento target nel quale visualizzare il nostro dato.

```
var animali = [
"gatto",
"cane",
];
```

```
gatto
```

```
<h1 id="test"></h1>
document.getElementById("test").innerHTML =
animali[0];
```

Convertire una stringa in array

Come abbiamo visto l'indice numerico ci permette di accedere facilmente al singolo elemento.

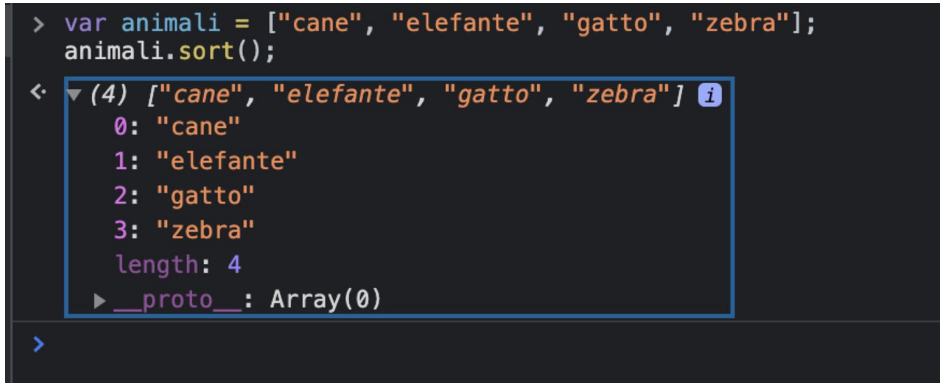
Possiamo però anche accedere a tutti gli elementi della lista usando il metodo **toString()** ;

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
<h1 id="test"></h1>
document.getElementById("test
").innerHTML =
animali.toString();

"gatto, cane, coniglio"
```

Utilizzando il metodo **sort();** possiamo ordinare alfabeticamente la lista degli elementi di un array.

```
var animali = [
“gatto”,
“elefante”,
“Zebra”,
“cane”
];
```



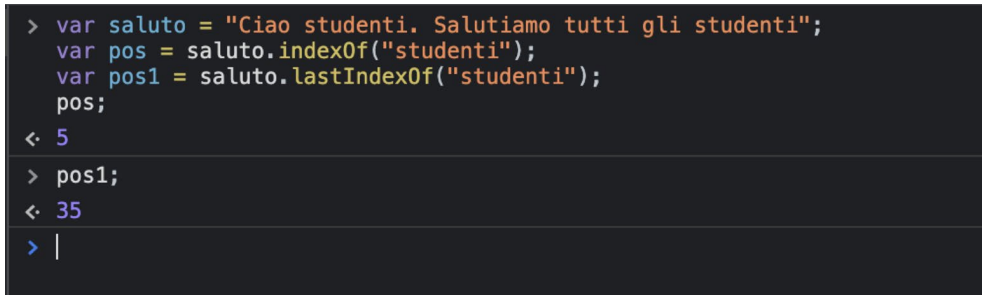
Metodi delle stringhe

I metodi delle stringhe: cercare nelle stringhe

JS, come tutti i linguaggi di programmazione, possiede numerosi metodi predefiniti per la manipolazione dei tipi di dati.

Una serie di metodi interessano la manipolazione delle stringhe.

Metodo indexOf();
Ci permette di trovare la prima posizione della parte di stringa indicata. Vediamo l'esempio:



Metodo indexOf();
Ci permette di trovare l'ultima posizione della parte di stringa indicata. Vediamo l'esempio:

I metodi delle stringhe: cercare nelle stringhe

I metodi indexOf() e lastIndexOf(), accettano un secondo parametro che indica a partire da quale posizione iniziare la ricerca:

```
var pos = saluto.indexOf("saluto", 10);
var pos = saluto.lastIndexOf("saluto", 10);
```

```
> var saluto = "Ciao a tutti gli studenti";
  var pos = saluto.search("Ciao");
  pos;
<· 0
>
```

Analogo a questi è il metodo search() che a differenza degli altri due metodi, non accetta un secondo parametro ma accetta le ricerche con espressioni regolari, quindi ha una capacità di ricerca di maggior dettaglio.

I metodi delle stringhe: lunghezza delle stringhe

Particolarmente utilizzato è il metodo **length** con il quale possiamo ottenere la lunghezza degli elementi che costituiscono una stringa.

```
> var saluto = "Ciao a tutti gli studenti";
  var pos = saluto.length;
  pos;
<· 25
>
```

Per lunghezza della stringa si intende il numero dei caratteri di cui è composto un elemento testo, compresi gli spazi.

Nell'esempio la stringa è composta da 21 caratteri e 4 spazi.

I metodi delle stringhe: estrazione di parte della stringa

Per estrarre parte della stringa utilizziamo i metodi:

- **slice();**
- substring();
- substr();

Il metodo slice(); accetta due parametri: posizione di inizio e fine dell'estrazione, ammettendo diverse opzioni:

slice(10, 15);

slice(-6 , -4); => l'estrazione inizia dalla fine

slice(8); => si estrae tutto fino alla posizione 8 e si restituisce tutto il resto

```
> var saluto = "Ciao a tutti gli studenti";
  var pos = saluto.slice(7 , 12);
  var pos1 = saluto.slice(-12 , -9);
  var pos2 = saluto.slice(7);
  pos;
< "tutti"
> pos1;
< "gli"
> pos2;
< "tutti gli studenti"
>
```

I metodi delle stringhe: estrazione di parte della stringa

Per estrarre parte della stringa utilizziamo i metodi:

- slice();
- **substring();**
- substr();

```
> var saluto = "Ciao a tutti gli studenti";
  var pos = saluto.substring(7 , 12);
  var pos1 = saluto.substring(7);
  pos;
<· "tutti"
> pos1;
<· "tutti gli studenti"
>
```

Il metodo substring(); è analogo al metodo slice(); ma non accetta l'indicazione di posizione negativa

substring(4 , 6);

substring(8); => si estrae tutto fino alla posizione 8 e si restituisce tutto il resto

I metodi delle stringhe: estrazione di parte della stringa

Per estrarre parte della stringa utilizziamo i metodi:

- slice();
- substring();
- **substr();**

Il metodo substr(); è analogo al metodo slice();
ma il secondo parametro indica la lunghezza della stringa estratta
substr(4 , 6); => estrae a partire da 4 per una lunghezza di 6 caratteri
substr(-4);
substr(8);

```
> var saluto = "Ciao a tutti gli studenti";
  var pos = saluto.substr(7 , 1);
  var pos1 = saluto.substr(-12 , 5);
  var pos2 = saluto.substr(7);
  pos;
< "t"
> pos1;
< "gli s"
> pos2;
< "tutti gli studenti"
>
```

I metodi delle stringhe: sostituzione delle stringhe

Il metodo fondamentale per la sostituzione di un carattere in una stringa è:

replace("parola da sostituire", "parola che sostituisce");

Il metodo replace() ha delle impostazioni di default:

- crea una nuova stringa con il carattere sostituito;
- è case sensitive; => vedi flag I
- sostituisce solo la prima occorrenza della parola specificata; => vedi metodo replaceAll();

```
> var saluto = "Ciao Studenti, un saluto a tutti gli studenti";
  var nuovaStr = saluto.replace("studenti" , "discenti");
  var nuovaStr1 = saluto.replace(/studenti/i , "discenti");
  nuovaStr;
<- "Ciao Studenti, un saluto a tutti gli discenti"
> nuovaStr1;
<- "Ciao discenti, un saluto a tutti gli studenti"
> var nuovaStr2 = saluto.replace("gli studenti" , "i discenti");
  nuovaStr2;
<- "Ciao Studenti, un saluto a tutti i discenti"
> var saluto = "Ciao studenti, un saluto a tutti gli studenti";
  var nuovaStr3 = saluto.replaceAll("studenti" , "discenti");
  nuovaStr3;
<- "Ciao discenti, un saluto a tutti gli discenti"
>
```

I metodi delle stringhe: sostituzione delle stringhe

Due metodi:

- toLowerCase();
- toUpperCase();

Possono essere compresi nei metodi di sostituzione.

Il metodo toUpperCase(); trasforma una stringa in maiuscolo.

Il metodo toLowerCase(); trasforma una stringa in minuscolo.

```
> var saluto = "Ciao Studenti, un saluto a tutti gli studenti";
  var maiuscolo = saluto.toUpperCase();
  var saluto = "CIAO STUDENTI, un saluto a tutti gli studenti";
  var minuscolo = saluto.toLowerCase();
  maiuscolo;
<· "CIAO STUDENTI, UN SALUTO A TUTTI GLI STUDENTI"
> minuscolo;
<· "ciao studenti, un saluto a tutti gli studenti"
>
```

I metodi delle stringhe: la concatenazione

Abbiamo già visto come sia possibile concatenare due o stringhe o parti con l'operatore +.
Un metodo alternativo molto versatile, è l'uso del metodo **concat();**

```
> var stringa = "Ciao a tutti.";
  var nuovaStringa = stringa.concat(" Buon corso");
  nuovaStringa;
< "Ciao a tutti. Buon corso"
> nuovaStringa = "Un saluto a tutti gli studenti".concat(" Buon corso!");
  nuovaStringa;
< "Un saluto a tutti gli studenti Buon corso!"
>
```

```
var stringa = "Ciao a tutti";

var nuovaStringa = stringa.concat("stringa da aggiungere");
var nuovaStringa = "Ciao a tutti".concat("Buon corso");
```

I metodi delle stringhe: trim e accesso per posizione

Il metodo **trim();** ci permette di eliminare gli eventuali spazi vuoti di una stringa.

Mentre con il metodo **charAt();** possiamo indicare un carattere relativo ad una posizione indicata

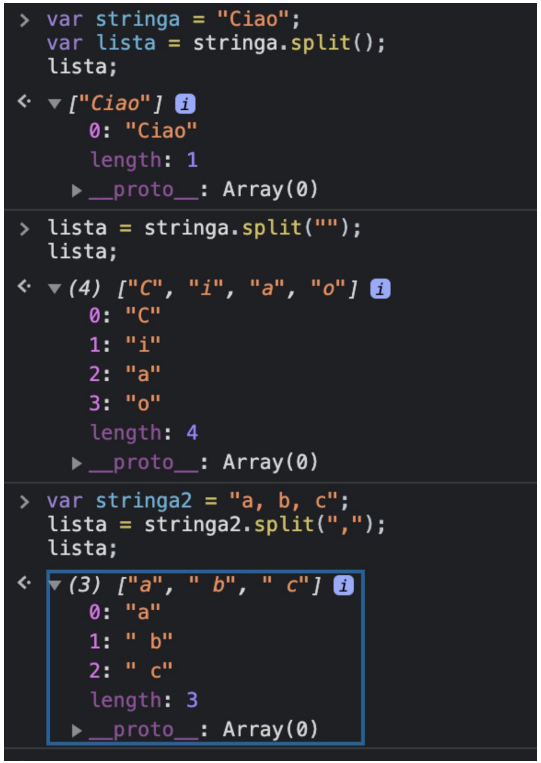
```
var stringa = “Ciao a tutti”;
var stringaOttimizzata = stringa.trim();
var carattere = stringa.charAt(0); => C
```

```
> var stringa = "     Ciao a tutti.";
  var stringa1 = stringa.trim();
  stringa1;
  "Ciao a tutti."
  var stringa = "     Ciao a tutti.      ";
  var stringa2 = stringa.trim();
  stringa2;
<· "Ciao a tutti."
> var stringa3 = stringa.charAt(3);
  stringa3;
<· " "
> var stringa3 = stringa.trim().charAt(3);
  stringa3;
<· "o"
>
```

Il metodo **split();** ci permette di suddividere la stringa nei suoi elementi.

In questo modo possiamo facilmente ottenere un array composto dai singoli caratteri della stringa.

```
var stringa = “Ciao”;
var lista = stringa.split(“”);
```



Metodi degli array

I metodi degli array: pop e push

Il metodo **pop();** ci permette di estrarre l'ultimo elemento da una lista per eliminarlo;

Mentre con il metodo **push();** possiamo aggiungere un elemento alla lista. Il nuovo elemento è aggiunto alla fine.

```
var fiori = [“rosa”, “geranio”, “tulipano”];
var elimina = fiori.pop(“ultimo elemento”);
var aggiungi = fiori.push(“ultimo elemento”);
```

```
> var fiori = ["rosa", "geranio", "tulipano"];
  var elimina = fiori.pop();
  elimina;
<- "tulipano"
> fiori.length;
<- 2
> var aggiungi = fiori.push("viola");
  fiori.length;
<- 3
> fiori;
<- ▼(3) ["rosa", "geranio", "viola"] i
     0: "rosa"
     1: "geranio"
     2: "viola"
     length: 3
   ▶ __proto__: Array(0)
```

I metodi degli array: pop e push

Il metodo **shift();** è analogo a pop(); ma seleziona e elimina il primo elemento dell'array.

Mentre con il metodo **unshift();** possiamo aggiungere in prima posizione un elemento alla lista.

```
var fiori = ["rosa", "geranio", "tulipano"];
var rimuovere  = fiori.shift();
rimuovere;
"rosa"
fiori.length;
2
fiori;
▶ (2) ["geranio", "tulipano"]
fiori.unshift("viola");
fiori.length;
3
fiori;
▶ (3) ["viola", "geranio", "tulipano"]
```

```
var fiori = ["rosa", "geranio", "tulipano"];
var elimina = fiori.shift("primo elemento");
var aggiungi = fiori.unshift("primo elemento");
```

I metodi degli array: il metodo delete()

Come suggerisce la keyword del metodo stesso, **delete();** cancella un elemento dell'array.

Tuttavia a differenza di pop() e shift(), questo metodo non ricostruisce un'indicizzazione dell'array.

var fiori = ["rosa", "geranio", "tulipano"];
delete fiori[1];
Elemento cancellato => geranio
L'elemento verrà sostituito da un indefinito empty e la posizione 1 rimarrà vuota

Il metodo **splice();** è piuttosto versatile applicato agli array.

Crea una nuova lista aggiungendo elementi nella posizione indicata e rimuovendone altri.

```
var fiori = [“rosa”, “geranio”, “tulipano”, “viola”];
var modifica = fiori.splice(1, 2, ”gerbera”, “papavero”]; => aggiungo 2 elementi indicati in posizione 1 e cancello 2 elementi
```

I metodi degli array: concatenazione

È possibile concatenare 2 o più array fra di loro con il metodo concat();

```
var unito = array1.concat(array2);
var unito =
array1.concat(array2,array3);
```

I metodi degli array: slice

Il metodo **slice();** su un array estrae l'elemento o gli elementi indicati dai parametri assegnati.

var fiori = ["rosa", "geranio", "tulipano", "viola"];

fiori.slice(1,2); => "geranio"

La selezione comprenderà solo l'elemento "geranio" poiché il parametro 2 indica il limite e non deve essere incluso

Operatori matematici e metodi math

Aritmetica con JS: operazioni fondamentali

È possibile eseguire tutte le operazioni aritmetiche, anche strutturate tramite espressioni, usando gli **operatori** matematici fondamentali:

+ => somma

- => sottrazione

* => moltiplicazione

/ => divisione

```
> var operando1 = 5;
  var operando2 = 4;
  var somma = 5 + 4;
  somma;
< 9
> somma = operando1 + 3;
  somma;
< 8
> somma = operando1 + operando2;
  somma;
< 9
> var sottrazione = somma - 2;
  sottrazione;
< 7
> sottrazione = somma - operando2;
  sottrazione;
< 5
> var moltiplicazione = operando1 * operando2;
  moltiplicazione;
< 20
> var divisione = operando2 / operando1;
  divisione;
< 0.8
> sottrazione = operando2 - operando1;
  sottrazione;
< -1
>
```

Aritmetica con JS: modulo ed operatore esponenziale

Per **modulo** si intende il rimanente risultante da una divisione. Il modulo viene indicato dalla keyword **%**

var divisione = 10 % 3; => **1**

L'operando 3 è compreso 3 volte nell'operando 10 con il **rimanente** di 1.

È possibile usare il simbolo ****** per indicare l'elevamenti a potenza.

var potenza = 5 ** 2;

```
> var modulo = 10 % 3;
  modulo;
<· 1
> modulo = 10 % 2;
  modulo;
<· 0
> var potenza = 5 ** 2;
  potenza;
<· 25
> var operando1 = 3;
  var operando2 = 4;
  potenza = operando2 ** operando1;
  potenza;
<· 64
>
```

Aritmetica con JS: incremento e decremento

Per **incrementare** un numero possiamo utilizzare l'operatore **++**:

5++ => incremento di una unità => 6;

numero++ => la variabile numero viene aumentata di una unità.

Per **decrementare** un numero possiamo utilizzare l'operatore **—** (doppio trattino):

5 — => decremento di una unità => 6;

numero — => la variabile numero viene diminuita di una unità.

```
> var numero = 5;
  numero++;
  var aumento = numero;
  aumento;
< 6
> numero--;
< 6
> var diminuisco = numero;
  diminuisco;
< 5
> var numero2 = 7;
  numero2--;
  diminuisco = numero2;
  diminuisco;
< 6
```

Aritmetica con JS: precedenza delle operazioni

Le operazioni vengono eseguite seguendo le consuete regole di precedenza matematica:

var risultato = 5 + 6 * 3; => 23

Si può indicare una diversa precedenza delle operazioni attraverso l'uso di parentesi:

var risultato = (5 + 6) * 3; => 33

```
> var numero1 = 5;
  var numero2 = 4;
  var risultato = numero1 + numero2 * numero1;
  risultato;
< 25
> risultato = (numero1 - 2) * numero2;
  risultato;
< 12
> risultato = (10 + 2) * (10 - 2);
  risultato;
< 96
>
```

Math è un oggetto che offre molti strumenti per la gestione delle operazioni matematiche e in generale per la manipolazione dei numeri.

L'oggetto Math agisce attraverso l'uso di numerosi metodi.

Questa è la sintassi di base:

```
Math.metodo(numero);
<
```

Metodi per l'approssimazione:

round() => approssimazione all'intero più vicino
ceil() => approssimazione all'intero superiore
floor() => approssimazione all'intero inferiore
trunc() => ritorna l'intero di un decimale

Metodo per l'identificazione del numero:

sign()
Con i valori:
-1 in caso di numero negativo
0 in caso di null
1 in caso di numero positivo

Potenza, radice quadrata e assoluto:

pow(a, b) => risulta il valore di a alla potenza di b
sqrt()
abs() => assoluto positivo del numero

Aritmetica con JS: Math

```
> Math.round(5.3);
< 5
> Math.round(5.7);
< 6
> Math.ceil(5.3);
< 6
> Math.floor(5.7);
< 5
> Math.trunc(5.72);
< 5
> Math.round(-5.3);
< -5
> Math.round(-5.7);
< -6
> Math.ceil(-5.3);
< -5
> Math.floor(-5.7);
< -6
>
```

```
> Math.sign(10);
< 1
> Math.sign(0);
< 0
> Math.sign(-9);
< -1
> Math.sign(-10.45);
< -1
>
```

```
> Math.pow(2,3);
< 8
> Math.pow(-2,3);
< -8
> Math.sqrt(16);
< 4
> Math.abs(5.6);
< 5.6
> Math.abs(-5.6);
< 5.6
>
```

challenge
NETWORK

shaping the skills of tomorrow

challengenetwork.it