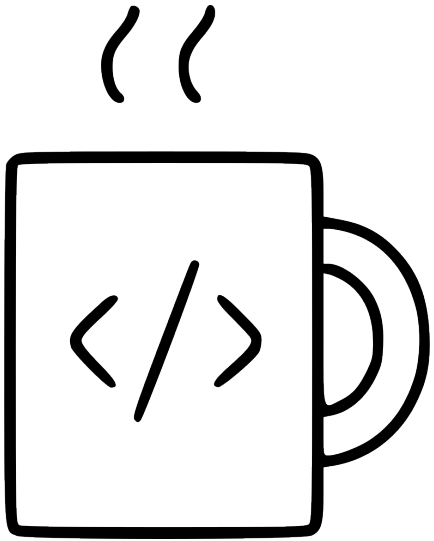
Umberto Emanuele

JavaScript 2
Giorno 3

Novembre 2023



PAUSA

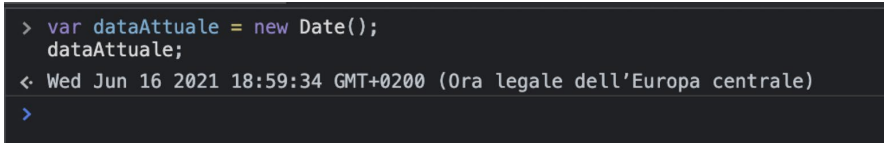
Ci vediamo alle ore 11.27

Date e metodi date

JavaScript gestisce in maniera veloce e dettagliata le date attraverso l'oggetto Date.

L'oggetto di default ritorna la data attuale del Browser.

var dataAttuale = new Date(); =>
mostra una stringa completa con nome del giorno, mese, giorno, anno,
orario completo e indicazione del time zone del browser.

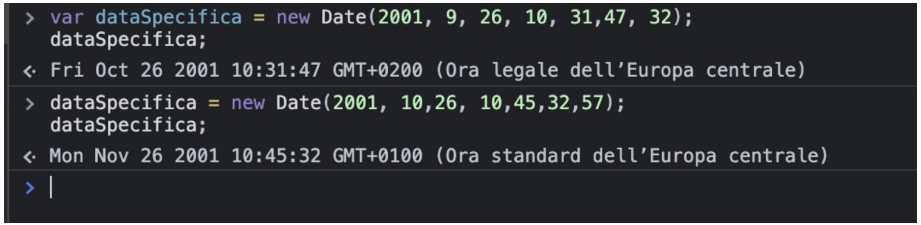


E' possibile creare una data specifica.
I valori che possono essere impostati sono nell'ordine:

Anno
Mese
Giorno
Ore
Minuti
Secondi
Millisecondi

Date(2001, 9, 26, 10, 31, 47, 32);

Attenzione ai mesi perchè vengono ordinati da 0 a 11, quindi gennaio ha ordine 0 e dicembre ordine 11.



E' possibile anche utilizzare parte delle opzioni mantenendo sempre il medesimo ordine.

Esempio:

Date(2001, 9);
indica l'anno e il mese

E’possibile creare una data partendo da una stringa.

Esempio:

Date("February 26, 2003 11:13:00”);

L’indicazione del mese è valida solo in lingua inglese, l’ordine può essere modificato ma l’output seguirà il medesimo ordine.

Esempio:

Date(“26 February 2003 11:13:00”);

```
> var dataStringa = new Date("February 26, 2003, 11:13:00");
  dataStringa;
<· Wed Feb 26 2003 11:13:00 GMT+0100 (Ora standard dell’Europa centrale)
> dataStringa = new Date("26 February 2003, 11:13:00");
  dataStringa;
<· Wed Feb 26 2003 11:13:00 GMT+0100 (Ora standard dell’Europa centrale)
> dataStringa = new Date("Marzo 26, 2003, 11:13:00");
  dataStringa;
<· Wed Mar 26 2003 11:13:00 GMT+0100 (Ora standard dell’Europa centrale)
> dataStringa = new Date("Dicembre 26, 2003, 11:13:00");
  dataStringa;
<· Invalid Date
```

In JavaScript è bene seguire lo standard ISO (ISO 8061), globalmente accettato

Esempio:

Date("2020-02-20");

JS conteggia il tempo da 1 gennaio 1970 e con il metodo parse() ritorna il millisecondi compresi fra questa data e quella indicata.

```
> var millisecondi = Date.parse("Feb 20, 2020");
  millisecondi;
<- 1582153200000
> var data = new Date(1582153200000);
  data;
<- Thu Feb 20 2020 00:00:00 GMT+0100 (Ora standard dell'Europa centrale)
> |
```

Formati ISO:

(YYYY-MM-DD) => anno, mese, giorno
(YYYY-MM) => anno, mese
(YYYY) => anno

(YYYY-MM-DDTHH:MM:SSZ) => orario separato da T e aggiunta del time zone, Z che indica UTC (Universal Time Coordinated)

Il time zone è quello del browser in uso.

Altro formato:
(MM/DD/YYYY)

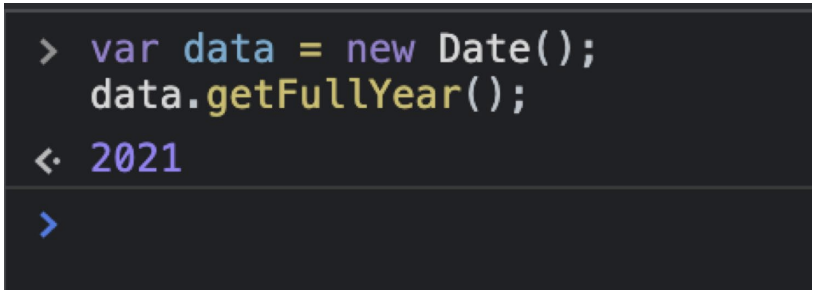
Gestire le date

Metodo get()

E' possibile utilizzare il metodo get() per prelevare delle informazioni dall'oggetto date.

Proprietà del metodo:

getFullYear() => 2020
getMonth() => mesi da 0 a 11
getDate() => giorni 1-31
getHours() => ore da 0 a 23
getMinutes() => minuti da 0 a 59
getSeconds() => secondi da 0 a 59
getMilliseconds() => millisecondi da 0 a 999
getTime() => millisecondi a partire da 01/01/1970
getDay() => giorni della settimana da 0 a 6



Per configurare delle opzioni dell'oggetto Date(), possiamo utilizzare il metodo set() con proprietà corrispettive al metodo get();

```
> var data = new Date();
  data.setFullYear(2019);
  data;
< Tue Jun 18 2019 16:43:16 GMT+0200 (Ora legale dell'Europa centrale)
```

Proprietà del metodo:

setFullYear() => 2020
setMonth() => mesi da 0 a 11
setDate() => giorni 1-31
setHours() => ore da 0 a 23
setMinutes() => minuti da 0 a 59
setSeconds() => secondi da 0 a 59
setMilliseconds() => millisecondi da 0 a 999
setTime() => millisecondi a partire da 01/01/1970

Parametri random e booleani

Metodi aritmetici random

random() è un metodo dell'oggetto Math che ritorna un numero casuale da 0 a 1 escluso.

Quindi il numero generato sarà sempre inferiore a 1

Math.random(); => numero decimale casuale

Per ottenere un numero intero possiamo utilizzare il metodo floor();

Math.floor(Math.random()); => 0

Inoltre moltiplicando per decine e centinaia, possiamo ottenere numeri casuali interi da 0 a 9 e da 0 a 99 rispettivamente.

```
Math.floor(Math.random() * 10);        => 0 - 9
Math.floor(Math.random() * 11);        => 0 - 10
Math.floor(Math.random() * 10) + 1; => 0 - 10
```

```
Math.floor(Math.random() * 100);       => 0 - 99
Math.floor(Math.random() * 101);       => 0 - 100
Math.floor(Math.random() * 100) + 1; => 0 - 100
```

Operatore booleano

Alla base di ogni logica di comparazione, vi è il corretto uso del tipo di dato booleano che ammette due soli valori:

true/false

Alcune regole fondamentali:

- Ogni elemento con un valore è vero;
- Ogni elemento che non ha un valore è falso;
- Una variabile assegnata a 0 o -0 risulta falsa
- Una variabile con valore di stringa vuoto, ritorna falsa var stringa = “ ”;

Così anche:

- Una variabile non definita => var stringa;
- Una variabile con null => var stringa = null;
  - Una variabile che ritorna un NaN (Not a Number)
    => var somma = “Ciao” * 4;

Funzioni booleane

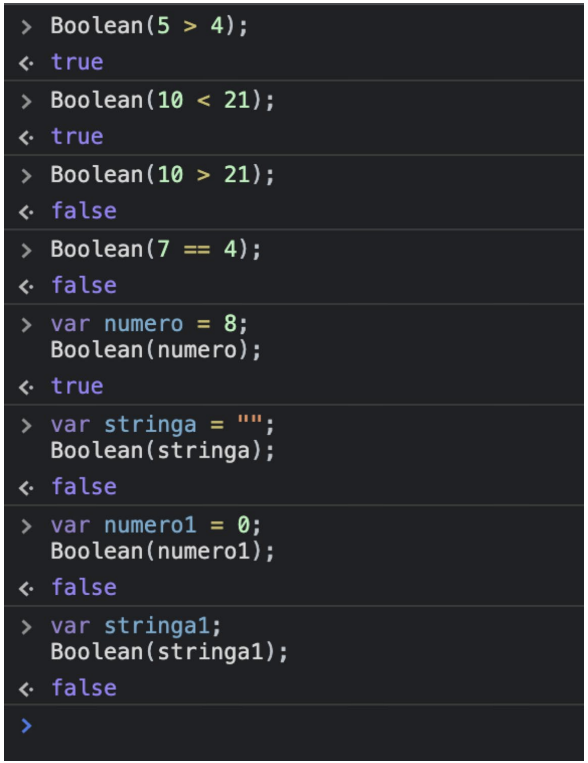
Per la verifica booleana di un elemento possiamo utilizzare il metodo Boolean();

Boolean(5 > 4);

Boolean(10 < 21);

Boolean(7 == 4);

var stringa = “”;
Boolean(stringa); => false



Operatore booleano come oggetto

Normalmente il tipo di dato booleano è definito in maniera letterale:

var numero = false;

Possiamo però anche utilizzare un oggetto:

var numero = new Boolean(false);

```
> var numero = false;
  var numero1 = new Boolean(false);
  typeof(numero);
<· "boolean"
> typeof(numero1);
<· "object"
>
```

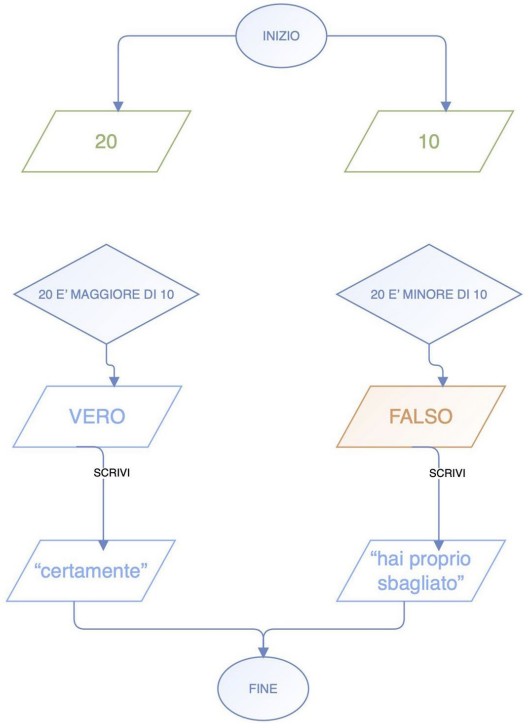
Uguali ma non identici
Stesso valore (false) ma di tipo diverso

```
> Boolean(numero == numero1);
<· true
> Boolean(numero === numero1);
<· false
>
```

Introduzione alla logica condizionale e ai cicli

Attraverso la logica condizionale possiamo controllare un flusso di condizioni sulla base del loro essere vere o false.

In base a questo possiamo configurare una serie di azioni differente.



La comparazione delle condizioni è una delle basi fondamentali delle strutture di logica condizionale.

Esempio, le condizioni che esprimiamo in codice, si presentano come:

Se l'utente ha meno di 18 anni oppure ha più di 16 anni, **scrivi: "può entrare"** ;
Invece se ha meno di 16 anni, **scrivi: "non può entrare"**

Oppure:

Se il numero è uguale a 10 ed è minore di 15, **scrivi: "numero piccolo"** ;
Invece se è uguale a 15 ed è superiore, **scrivi: "numero grande"**

Naturalmente le condizioni comparate, sulla base delle quali vengono eseguite alcune istruzioni, devono essere o vere o false.

Devono rispondere quindi, in maniera non equivoca, ad una logica booleana.

Se l'utente ha meno di 18 anni oppure ha più di 16 anni, **scrivi: "può entrare"** ;

Se questa condizione sarà vera, verrà eseguita l'istruzione prescritta (scrivere il messaggio).

Altrimenti si passerà alla condizione seguente:

Invece se ha meno di 16 anni, **scrivi: "non può entrare"**

Anche la seconda condizione potrebbe essere falsa, quindi la struttura di controllo potrebbe prevedere ancora altre condizioni, fino all'avverarsi di una.

Il criterio di controllo è dato dalla comparazione di un valore tipicamente passato attraverso una variabile.

Il costrutto if

Il costrutto if

Il primo costrutto condizionale è indicato dalla keyword if:

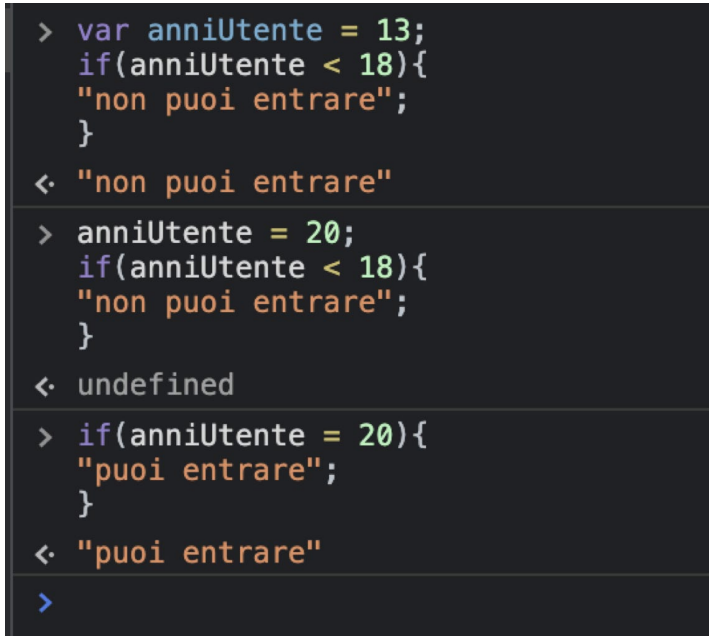
```
if(condizione){
//istruzioni da eseguire;
}
```

Esprime la possibilità di eseguire una parte di codice se la condizione è vera.

var anniUtente = 13;

Se l'utente ha meno di 18 anni, scrivi: "non può entrare";

```
if(anniUtente < 18){
console.log("non puoi entrare");
}
```



I parametri del controllo di una condizione, possono combinare più comparazioni attraverso l'uso degli operatori logici.

```
> var anniUtente = 13;
  if(anniUtente < 18 && anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 17;
  if(anniUtente < 18 && anniUtente < 16){
  "non puoi entrare";
  }
<· undefined
>
```

Avendo utilizzato l'operatore logico &&and, tutte e due le condizioni devono avverarsi perchè possa essere eseguita l'istruzione.

```
> var anniUtente = 13;
  if(anniUtente < 18 || anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 17;
  var anniUtente = 13;
  if(anniUtente < 18 || anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
>
```

In questo caso, invece, avendo utilizzato l'operatore logico || or, una sola delle condizioni deve avverarsi perchè possa essere eseguita l'istruzione.

Il costrutto if/else

Con la keyword else all'interno del costrutto, possiamo definire una istruzione alternativa da eseguire se il controllo di if risulta falso.

In questo caso avendo determinato la condizione controllata da if, tutte le altre possibili condizioni, saranno controllate da else.

```
> var anniUtente = 13;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 20;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "puoi entrare"
>
```

Con la keyword else if all'interno del costrutto, possiamo definire una istruzione alternativa ulteriore rendendo il controllo ancora più dettagliato.

```
if( prima condizione){
//istruzioni
}else if(seconda condizione){
//istruzioni
}else{
//istruzioni
}
```

```
> var anniUtente = 13;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 20;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "puoi entrare"
>
```

Il costrutto switch

Il costrutto switch

Con la keyword switch possiamo creare una struttura di controllo delle condizioni che segue la logica del costrutto if implementando diverse opzioni (case) che al verificarsi eseguono le istruzioni prescritte.

La struttura può essere strutturata con un numero indefinito di casi.
Il valore della variabile o in generale dell'espressione, viene comprata con i casi.
Quando si verifica una combinazione, viene eseguita l'istruzione prescritta.

```
switch(variabile o espressione){
case a:
//istruzioni da eseguire;
case b:
//istruzioni da eseguire;
default:
//istruzioni da eseguire;
}
```

La keyword case esprime una opzione univoca, così che una condizione vera si verifichi una sola volta.

Tipicamente i case seguono un ordine numerico ma possono avere un loro identificativo letterale.
Possiamo associare a più case le stesse istruzioni:

```
case a:
case b:
//istruzioni da eseguire;
case c:
//istruzioni da eseguire;
```

```
var weekend = “sabato”;

switch(giornoSettimana){
case 0:
giorno = “venerdì”;
case 1:
giorno = “sabato”;
case 2:
giorno = “domenica”;
}
```

Il costrutto switch

Caso di default

La keyword default indica un caso valido quando nessuno dei precedenti si è verificato.

Permette così al costrutto switch un controllo in ogni caso possibile con un relativo output.

Solitamente, ma non in maniera obbligata, il default è posto alla fine della catena dei casi.

```
var weekend = “sabato”;

switch(weekend){
case 0:
giorno = “venerdì”;
break;
case 1:
giorno = “sabato”;
case 2:
giorno = “domenica”;
}
```

Il costrutto switch

Caso di default

La keyword default indica un caso valido quando nessuno dei precedenti si è verificato.

Permette così al costrutto switch un controllo in ogni caso possibile con un relativo output.

Solitamente, ma non in maniera obbligata, il default è posto alla fine della catena dei casi.

```
var weekend = “sabato”;

switch(weekend){
case 0:
giorno = “venerdì”;
break;
case 1:
giorno = “sabato”;
break;
case 2:
giorno = “domenica”;
break;
default:
giorno: “feriale”
}
```

Il ciclo while

I cicli rappresentano un'altra fondamentale struttura di programmazione.
Il loro scopo è la lettura a scorrimento di una serie di elementi.

In particolare con la keyword while definiamo un ciclo che scorre un blocco di istruzioni fino a che (while) la condizione rimane vera.

```
while(condizione){

//blocco di istruzioni da eseguire

condizione++
}
```

La mancanza dell'incremento (o decremento) della condizione, genera un ciclo infinito che porta al blocco del browser sul ciclo stesso.

La condizione è passata attraverso un variabile che in può rappresentare l'inizio del ciclo.

Esempio:

var numero = 0;

```
while(numero < 10 ){
numero++;
}
```

In questo caso il nostro ciclo ha inizio da 0 e Scriverà la serie dei numeri fino a 9.

Al 10 il ciclo avrà termine

```
while(numero <= 10 ){
numero++;
}
```

In questo caso il nostro ciclo ha inizio da 0 e scriverà la serie dei numeri fino a 10 incluso.

Il ciclo espresso dalle keyword do / while rappresenta una variante del ciclo while.

In questo caso il blocco di istruzioni viene eseguito una volta prima del controllo della condizione, quindi della verifica booleana della stessa.

```
do{
//blocco di codice da eseguire
condizione++;
}
while(condizione);
```

```
var numero = 0;

do{
text += “Ciclo: “ + numero + “<br>”;
numero++;
}
while(numero < 10);
```

Iterazione degli array

Particolarmente utile è l'applicazione dei cicli sugli array, vista la loro natura di liste.

Il metodo forEach() richiama una funzione per ogni elemento dell'array.

Il metodo map() crea un nuovo array applicando ad ogni elemento una funzione.

```
var numeri = [32, 7, 84, 12];
numeri.forEach(funzione );
```

La funzione può avere 3 parametri:

- valore
- indice
- array

```
function funzione(valore) {
 return valore + 4;
}
```

```
var lista = [45, 4, 9, 16, 25];
var nuovaLista = lista.map(funzione );
```

Il metodo filter() applica una funzione agli elementi di un array che filtra l'output secondo una opzione data.

```
var numeri = [32, 7, 84, 12];
numeri.filter(funzione );
```

```
function funzione(valore) {
  return valore > 30;
}
```

In questo caso l'output è 32 e 84 perchè abbiamo applicato un filtro attraverso una funzione che esegue una ricerca di valori maggiori di 30.

Il metodo reduce() ritorna un unico valore applicando una funzione a tutti gli elementi dell'array.

Di default la direzione di esecuzione del metodo è da sinistra a destra.

Per l'esecuzione da destra a sinistra si può usare l'opzione del metodo reduceRight().

```
var numeri = [2, 4, 7, 6, 3];
var totaleLista = numeri.reduce(funzione );

function funzione (totale, valore) {
 return totale + valore;
}
```

L'output è 22 generato da 2+4+7+6+3

```
function funzione (totale, valore) {
 return totale * valore;
}
```

L'output è 1008 generato da 2x4x7x6x3

E' possibile assegnare un valore iniziale al metodo:

```
var totaleLista = numeri.reduce(funzione, 10);
```

Il metodo every() determina se un array risponde ad un certo requisito in tutti i suoi elementi.

Il metodo some() determina se un array risponde ad un certo requisito in qualcuno dei suoi elementi.

Il metodo indexOf() determina la posizione di un elemento in base ad una chiave di ricerca.

```
var numeri = [3,22,4,76,39];
var ricerca = numeri.every(funzione );

function funzione (valore) {
 return valore > 10;
}
```

L'output darà un valore false perchè non tutti i valori sono maggiori di 10

```
var numeri = [3,22,4,76,39];
var ricerca = numeri.some(funzione );

function funzione (valore) {
 return valore > 10;
}
```

L'output darà un valore true perchè alcuni dei valori sono maggiori di 10

```
var pets = ["cane", "gatto", "criceto", "coniglio"];
var posizione = pets.indexOf("gatto");
```

Il metodo find() determina il primo elemento di un array che risponde ad requisito impostato da una funzione.

Il metodo findIndex() determina la posizione di un elemento di array che risponde ad un certo requisito.

```
var numeri = [4, 9, 16, 25, 29];
var primoItem = numeri.find(funzione );

function funzione (valore) {
 return valore > 10;
}
```

L'output è 16 che è il primo elemento maggiore di 10

```
var numeri = [4, 9, 16, 25, 29];
var primoItem = numeri.findIndex(funzione );

function funzione (valore) {
 return valore > 10;
}
```

L'output è 2 che è la posizione del primo elemento maggiore di 10 (16)

Il ciclo for

Il ciclo for

Con la keyword for indichiamo un ciclo che viene eseguito per un certo numero di volte

Per esempio:

Se in un array volessimo iterare lo stesso codice più e più volte, dovremmo ripetere le stesse istruzioni per ogni elemento.

```
var numeri = [4, 9, 16, 25, 29];

somma += numeri[0];
somma += numeri[1];
somma += numeri[2];
somma += numeri[3];
somma += numeri[4];
```

```
var numeri = [4, 9, 16, 25, 29];

for(var i = 0; i < numeri.length; i++){
somma += numeri[i];
}
```

La condizione controllata dal ciclo for richiede 3 parametri:

```
for(parametro1; parametro2; parametro3){

//blocco di codice da eseguire

}
```

parametro1 = è eseguito una volta e rappresenta il punto di partenza del ciclo

parametro2 = indica la condizione stessa e il numero di volte per cui eseguire il codice

parametro3 = è eseguito dopo il blocco di codice e per tante volte quante quelle indicate dal parametro2

```
for(var i = 0; i < 5; i++){
ciclo += i + "<br>";
}
```

Questo ciclo scriverà la sequenza di numeri da 0 a 4 perchè la variabile i è impostata da 0 con un valore minore di 5. L'incremento sarà eseguito fino al 4. Raggiunto il 5 la condizione sarà false e il ciclo si chiude.

Il ciclo for in

La struttura for in è un ulteriore modo per ciclare un array.

Si serve dell'associazione di una chiave (variabile) al valore rappresentato da ogni singolo elemento dell'array.

```
for(variabile in array){
//blocco di codice da eseguire
}
```

```
var array = [4, 9, 16, 25, 29];

for(var chiave in array){
array[chiave];
}
```

Un modo comune per esprimere il legame fra la chiave e il valore singolo è esprimerla al singolare rispetto al nome dell'array.

Esempio:
array => numeri
chiave => numero

```
for(var numero in numeri)
```

Il ciclo for of

La struttura for of determina una scomposizione di una serie di elementi iterabili, cioè soggetti singolarmente a ciclo.
Il concetto si comprende bene sottoponendo una stringa al metodo.

```
var nome = “MARIO”;
var testo = “ ”;

for(var carattere of nome){
testo += carattere + “<br>”;
}
```

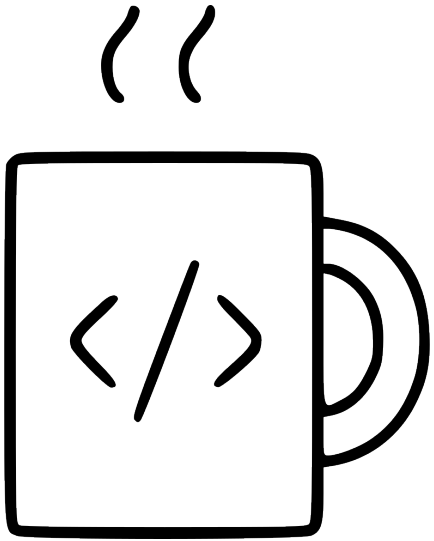
L’output della variabile testo sarà:

M
A
R
I
O

for of è applicabile anche agli array:

```
var numeri = [4, 9, 16, 25, 29];

for(var numero of numeri){
testo += numero;
}
```



PAUSA

Ci vediamo alle ore 14.00

challenge
NETWORK

shaping the skills of tomorrow

challengenetwork.it