

# Javascript

**Notes by Barry**

# Introduzione a javascript

Javascript è un linguaggio di programmazione ad alto livello, interpretato, è ampiamente utilizzato per lo sviluppo di siti web, applicazioni web e mobile, e applicazioni lato server.

# Introduzione a javascript

Ora andrò ad esaminare il linguaggio, JavaScript è un linguaggio di programmazione:

- ad alto livello;
- dinamico;
- dinamicamente tipizzato;
- debolmente tipizzato;
- interpretato.

## Ad alto livello

Javascript fornisce concetti generali che consentono di trascurare i dettagli tecnici dell'esecuzione del programma. Gestisce automaticamente la memoria tramite un garbage collector, consentendoti di concentrarti sul codice senza doverti preoccupare troppo della gestione della memoria come avviene in altri linguaggi come C e C++ con i puntatori.

Inoltre, offre una vasta gamma di strutture che ti consentono di manipolare variabili e oggetti di grande importanza.

## Dinamico

Javascript è un linguaggio di programmazione dinamico che si differenzia dai linguaggi di programmazione statici perché esegue molte operazioni durante il runtime anziché durante la fase di compilazione. Ciò comporta una serie di vantaggi e svantaggi e si traduce in importanti caratteristiche come la tipizzazione dinamica, il binding dinamico, la riflessione, la programmazione funzionale, la modifica delle proprietà di un oggetto durante l'esecuzione, la chiusura e altro ancora.

# Dinamicamente tipizzato

In Javascript il tipo di dato di una variabile non è vincolato, il che significa che è possibile assegnare a una variabile qualsiasi tipo di dato, indipendentemente dal tipo di dato precedentemente contenuto.

## Debolmente tipizzato

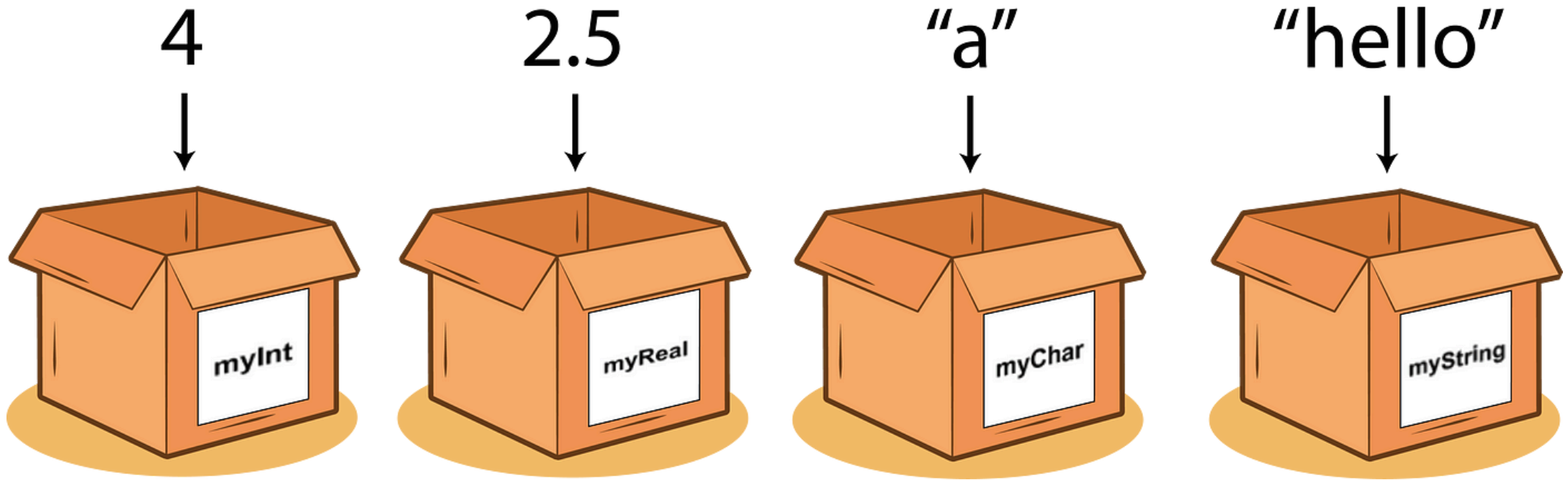
in opposizione alla tipizzazione forte, nei linguaggi debolmente tipizzati il tipo di oggetto non è fisso.

## Interpretato

Javascript è un linguaggio interpretato, che vuol dire che non necessita di una fase di compilazione prima dell'esecuzione del programma, al contrario di C++ per esempio. In pratica, per una questione di performance, i browser compilano JavaScript prima di eseguirlo senza bisogno di nessun un altro step.



## Variabili



# Variabili

Una variabile è un oggetto che rappresenta una posizione di memoria in cui è possibile memorizzare e recuperare un valore. Una variabile ha un nome univoco che viene utilizzato per identificarla nel programma e un tipo di dato che definisce il tipo di valore che può essere memorizzato al suo interno.

## Naming delle variabili

Ogni variabile ha un nome univoco che la identifica nel contesto del programma, rispettando le regole di denominazione specifiche del linguaggio di programmazione utilizzato. Il nome della variabile è ciò che permette di fare riferimento ad essa nel codice.

## Dichiarazione

In javascript le variabili vengono dichiarate con le parole chiave **var**, **let** e **const**.

## var

Con var, la variabile può essere dichiarata senza assegnare un valore iniziale e può essere successivamente assegnata con un valore.

```
var = variabelName;
```

## var

Tuttavia la parola chiave `var` è deprecata, è stata la modalità di dichiarazione delle variabili in JavaScript ed è stata utilizzata nelle versioni precedenti dello standard [ecmascript](#).

## let

La parola chiave `let` viene utilizzata per dichiarare una variabile con uno scope di blocco, ovvero è una variabile il cui accesso è limitato al blocco di codice in cui è stata dichiarata.

```
let variabelName="sono una variabile";
```

## const

La parola chiave `const` viene utilizzata per dichiarare una variabile con un valore costante. Una volta assegnato un valore alla variabile `const`, quel valore non può essere modificato successivamente.

```
const pi = 3.14;
```



# Dati primitivi

I dati primitivi sono un insieme di valori sul quale sono definite delle operazioni, e in js includono:

## Number

Number: il tipo di dato numerico è generico in javascript, ed è rappresentato da :

- numeri interi;
- numeri decimali;
- $]-\infty, +\infty[$

## Stringhe

Rappresentano una sequenza di caratteri e vengono delimitati da apici singoli o doppi.

Ad esempio:

```
let str = " Hello world";  
let str2 = 'Hello world'
```

## Booleani

Rappresentano un valore di verità possono essere true o false.

```
let isTrue = true;  
let isFalse = false;
```

## Null

Rappresenta l'assenza intenzionale di un valore e può essere assegnato solo esplicitamente.

```
let nullValue = null;
```

# Undefined

Indica che una variabile è stata dichiarata, ma non gli è stato assegnato un valore.

```
let undefValue ;
```

# Operatori

Gli operatori sono simboli speciali o sequenze di simboli, utilizzati per eseguire operazioni sugli operandi, che possono essere valori o variabili. Gli operatori possono essere classificati in diverse categorie a seconda del tipo di operazioni che eseguono. Alcune delle categorie comuni degli operatori includono:

## Operatori aritmetici

Gli operatori aritmetici in JavaScript comprendono:

- somma  $+$ ;
- sottrazione  $-$  ;
- moltiplicazione  $*$ ;
- divisione  $\div$ ;
- esponenziale  $n^x$  ;
- modulo  $\%$  ;
- decremento  $--$  ;



## Operatori aritmetici

```
let a = 3;  
let b = 2;  
let result;  
result = a + b; // 5  
result = a - b; // 1  
result = a * b; // 6  
result = a / b; // 1,5  
result = a ** b; // 9  
result = a % b; // 1  
result = a++; // 3  
result = a--; // 2
```

## Operatori di confronto

Un operatore di confronto confronta due operandi e restituisce un valore booleano. Gli operatori di confronto JavaScript sono i seguenti:

- uguale (`==`);
- disuguaglianza (`≠`);
- uguaglianza stretta (`===`);
- disuguaglianza stretta (`≠≠`);

- strettamente maggiore ( $>$ );
- maggiore o uguale ( $\geq$ );
- strettamente minore ( $<$ );
- minore o uguale ( $\leq$ ).

# operatori

```
let a = 10;  
let b = 12;  
let c = 10;  
a == c; //true  
a != b; //true  
10 == "10" //true  
10 === "10" //false  
10 !== "10" //true  
a > b // false  
b < c // true  
a >= b // true
```

## Operatori di assegnamento

L'operatore di assegnamento è rappresentato dal simbolo `=` e viene utilizzato per assegnare un valore a una variabile. Gli operatori di assegnamento JavaScript sono i seguenti:

- assegnamento (`=`);
- assegnamento di addizione e concatenazione (`+=`);
- assegnamento di sottrazione (`-=`);

# operatori

- assegnamento di moltiplicazione ( $* =$ );
- assegnamento di divisione ( $/ =$ );
- assegnamento di modulo ( $\% =$ );

# operatori

```
let numero = 5;
numero += 3;
console.log(numero); // Output: 8
let numero = 10;
numero -= 4;
console.log(numero); // Output: 6
let numero = 3;
numero *= 5;
console.log(numero); // Output: 15
let numero = 20;
numero /= 4;
console.log(numero); // Output: 5
let numero = 17;
numero %= 5;
console.log(numero); // Output: 2
let testo = "Hello";
testo += " world";
console.log(testo); // Output: "Hello world"
```

# Operatori di logici

Gli operatori logici sono strumenti utilizzati per combinare espressioni booleane al fine di valutarne la verità. Essi si basano sull'algebra di Bool e sulla logica matematica per effettuare tali operazioni. In JavaScript, i principali operatori logici sono AND (&&), OR (||) e NOT (!). L'operatore AND (&&) restituisce true solo se entrambe le espressioni booleane che combina sono true. Ad esempio, nell'espressione "A && B", entrambe le variabili A e B devono essere true affinché l'intera espressione sia valutata come true.



L'operatore OR (||) restituisce true se almeno una delle espressioni booleane combinate è true. In altre parole, nell'espressione "A || B", se una delle variabili A o B è true, l'intera espressione sarà valutata come true. L'operatore NOT (!) è un operatore unario che inverte il valore di verità di un'espressione booleana. Ad esempio, se "A" è true, allora "\$ !A\$" sarà false e viceversa.

Prima di utilizzare questi operatori all'interno di un programma JavaScript, è utile comprendere il loro funzionamento matematico e come vengono applicati nell'algebra di Bool e nella logica matematica. Questo aiuterà a utilizzarli correttamente per valutare le espressioni booleane e ottenere i risultati desiderati.

# Algebra booleana

L'algebra booleana è un'area della matematica e dell'informatica che si occupa dello studio delle operazioni e delle proprietà degli oggetti logici. L'algebra è definita su un insieme di due soli elementi vero (V) e falso (F).

# Predicati logici

I predicati logici sono espressioni o proposizioni che coinvolgono variabili e operatori. Consideriamo come predicati gli operatori confronto. Poichè Quando viene utilizzato un operatore di confronto, viene effettuato un test logico sulla relazione tra i due valori. Il risultato di tale test logico può essere considerato come un predicato logico che afferma una proposizione di verità o falsità.

Ad esempio, l'espressione  $3 < 5$  utilizza l'operatore di confronto minore per confrontare il valore con il valore , Il risultato di questo confronto è vero.

## Connettivi logici

I connettivi logici sono strumenti utilizzati per combinare proposizioni o espressioni logiche al fine di valutare la loro verità. Se  $P$  e  $Q$  sono due predicati possiamo creare nuovi predicati tramite tre operatori fondamentali:

- $\vee$  **or**;
- $\wedge$  **and**;
- $\neg$  **not**;

## Regole dei conettivi logici

- $P \vee Q$  è vero se almeno uno tra  $P$  e  $Q$  è vero, altrimenti è falso;
- $P \wedge Q$  è vero se  $P$  e  $Q$  sono \$\$veri, altrimenti
- $\neg P$  è vero se  $P$  è falso
- $\neg P$  è falso se  $P$  è vero

## Operatori logici in javascript

Dopo aver visto matematicamente la logica booleana, vediamo la sintassi in javascript:

```
let a = true;
let b = false;
console.log(a && b); // Output: false
let x = true;
let y = false;
console.log(x || y); // Output: true
let z = false;
console.log(!z); // Output: true
```



# Conditional statement

I conditional statement sono i punti decisionali del codice. Questo costrutto di controllo del flusso del programma offre la possibilità di eseguire determinati blocchi di codice solo se una determinata condizione è vera.

### if statement

L'istruzione `if` è l'istruzione di controllo che consente all'interprete di prendere decisioni o, più precisamente, di eseguire istruzioni in modo condizionale. L'istruzione è seguita da una condizione racchiusa tra parentesi tonde `()`. Questa condizione può essere qualsiasi espressione che può essere valutata come `true` o `false`.

# Conditional statement

Se la condizione è valutata come true, il blocco di codice all'interno delle parentesi graffe che segue l'istruzione "if" viene eseguito. Se la condizione è valutata come false, il blocco di codice viene saltato e l'esecuzione continua con il codice successivo dopo il blocco "if".

```
let numero = 1;  
if (numero > 0) {  
    console.log("Il numero è positivo.");  
}
```

## Conditional statement

In questo esempio, la condizione `numero > 0` viene valutata. Se il valore della variabile `numero` è maggiore di zero, allora la condizione è vera e verrà stampato a schermo:

```
Il numero è positivo.
```

# L'istruzione else

L'istruzione else è un'opzione facoltativa che può essere utilizzata insieme all'istruzione if per gestire un caso alternativo quando la condizione dell'istruzione "if" è valutata come falsa. L'istruzione else viene eseguita solo se la condizione dell'istruzione if è valutata come falsa, altrimenti il blocco di codice all'interno dell'istruzione else viene saltato.

# Conditional statement

```
let numero = -1;  
if (numero > 0) {  
  console.log("Il numero è positivo.");  
} else {  
  console.log("Il numero è negativo o zero.");  
}
```

In questo esempio, se il valore della variabile numero è maggiore di zero, la condizione dell'istruzione viene valutata come falsa viene eseguito il blocco di codice all'interno dell'istruzione else , ovvero:

```
Il numero è negativo o zero
```

# Operatore ternario

L'operatore ternario , è un'operatore che permette di scrivere un'istruzione condizionale in modo più compatto rispetto all'istruzione if else. L'operatore ternario è rappresentato dal simbolo ? e viene utilizzato per valutare una condizione e restituire un valore in base al risultato della valutazione.

```
condizione ? valore_se_vera : valore_se_falsa;
```

# Conditional statement

Quando l'operatore ternario vien utilizzato, la condizione valutata se la condizione è vera, se la condizione è vera viene estituito il valore\_se\_vera, altrimenti viene restituito il valore\_se\_falsa.

```
let numero = 10;  
let risultato = (numero > 0) ? "Il numero è positivo" : "Il numero è negativo o zero";  
console.log(risultato);
```

Come per l'esempio visto in precedenza la condizione `numero > 0` viene valutata vera. è verrà stampato a schermo:

```
Il numero è positivo.
```



# Conditional statement

Lo Switch statement è un costrutto di controllo del flusso che permette di eseguire diverse azioni a seconda del valore di una variabile o di un'espressione.

Viene utilizzato per semplificare la gestione di molteplici casi o condizioni e può essere un'alternativa più leggibile rispetto a una serie di istruzioni if else.

# Conditional statement

```
let giorno = "lunedì";
switch (giorno) {
case "lunedì":
console.log("Oggi è lunedì.");
break;
case "martedì":
console.log("Oggi è martedì.");
break;
case "mercoledì":
console.log("Oggi è mercoledì.");
break;
case "giovedì":
console.log("Oggi è giovedì.");
break;
case "venerdì":
console.log("Oggi è venerdì.");
break;
default:
console.log("Oggi non è un giorno della settimana.");
}
```

# Conditional statement

In questo esempio, viene definita una variabile `giorno` con il valore `lunedì`. Lo statment viene utilizzato per confrontare il valore di `giorno` con i casi definiti. Poiché `giorno` corrisponde al caso `"lunedì"`, viene eseguito il blocco di codice corrispondente, stampando sulla console:

```
Oggi è lunedì.
```

# Cicli

I cicli vengono utilizzati per eseguire attività ripetute in base a una condizione. Le condizioni generalmente restituiscono valori di verità. Un ciclo continuerà a funzionare fino a quando la condizione da noi definita ritorna il valore di verità falso.

## Ciclo for

I cicli for sono comunemente usati per eseguire un blocco di codice un determinato numero di volte.

Il ciclo for è costituito da tre espressioni facoltative, seguite da un blocco di codice.

```
for (inizializzazione; condizione; incremento/decremento) {  
    // blocco di codice da eseguire  
}
```

Il ciclo for è costituito da tre espressioni facoltative, seguite da un blocco di codice:

1. **Inizializzazione:** Eseguita all'inizio del ciclo, solitamente per dichiarare e inizializzare una variabile di controllo.
2. **Condizione:** Valutata all'inizio di ogni iterazione. Se vera, il blocco di codice viene eseguito. Se falsa, il ciclo termina.
3. **Incremento/decremento:** Eseguito alla fine di ogni iterazione per modificare il valore della variabile di controllo.

```
// Calcola la somma dei numeri da 1 a 10  
let sum = 0;  
for (let i = 1; i <= 10; i++) {  
    sum += i;  
}  
console.log("La somma dei numeri da 1 a 10 è: " + sum);
```

Questo esempio mostra un ciclo for che calcola la somma dei numeri da 1 a 10. Include:

## 1. Inizializzazione

- `let i = 1`: Inizializza la variabile di controllo `i`;
- Viene eseguita solo una volta all'inizio del ciclo;
- Può includere la dichiarazione della variabile `(let i)` o usare una variabile esistente;  
Può inizializzare più variabili: `for (let i = 0, j = 10; ...)`.

In questo caso, iniziamo da 1 perché vogliamo sommare i numeri da 1 a 10.



## 2. Condizione

- `i <= 10`: Condizione che determina se il ciclo continua
- Valutata all'inizio di ogni iterazione
- Se true, il blocco di codice viene eseguito
- Se false, il ciclo termina

Questa condizione assicura che sommiamo solo i numeri da 1 a 10.

## 3. Incremento

- `i++`: Incrementa la variabile di controllo dopo ogni iterazione.
- Eseguito alla fine di ogni iterazione, prima di valutare nuovamente la condizione.

L'incremento `i++` assicura che passiamo attraverso tutti i numeri da 1 a 10.

## 4. Corpo del ciclo e Sommatoria

- `sum += i`: Aggiunge il valore corrente di `i` a `sum` ad ogni iterazione.
- Rappresenta la sommatoria :

$$\sum_{i=1}^{10} i$$

- La sommatoria calcola la somma di una sequenza di numeri
- In questo caso, calcola la somma dei primi 10 numeri naturali
- Il risultato di questa sommatoria è:

$$\frac{n(n+1)}{2} = \frac{10(10+1)}{2} = 55$$