

Variabili, Tipi, Valori e Operatori

Variabili

Una variabile è un'area di memoria che contiene informazioni che possono essere modificate durante l'esecuzione di un programma.

Una variabile è caratterizzata da:

- **Nome:** identificatore univoco
- **Valore:** contenuto attuale
- **Tipo:** tipo di dato contenuto

Valori e Tipi di Base

- **Numbers:** 42 , 3.14 , -10
- **Strings:** "Hello" , 'World'
- **Booleans:** true , false
- **Special:** undefined , null , NaN , Infinity

Numbers

// Interi

42

-17

0

// Decimali

3.14

-0.001

Strings

"Hello World"

'JavaScript'

"I'm coding"

'He said "Hi!"'

Caratteri speciali:

- `\n` - nuova linea
- `\t` - tab
- `\'` - apice singolo
- `\\"` - apice doppio

Esercizio – Stringhe

Esercizio: Formattazione del testo

```
// Crea un messaggio che includa:  
// - Un testo su più righe usando  
// - Una citazione con apici doppi dentro apici singoli  
// - Una tabulazione \t per indentare parte del testo
```

Operatori Aritmetici

```
// Operatori base  
5 + 3 // Addition  
10 - 4 // Subtraction  
3 * 4 // Multiplication  
10 / 2 // Division  
9 % 2 // Modulo (resto)  
2 ** 3 // Esponente
```

Esercizio 1 – Calcolatrice base

Crea variabili per due numeri e calcola:

- La loro somma
- Il loro prodotto
- Il quadrato del primo numero
- Il risultato della divisione tra i due numeri
- Il resto della divisione tra i due numeri

Commenti

```
// Commento su una linea  
  
/* Commento  
su più  
righe */  
  
// TODO: implementare questa funzione  
let x = 5; // valore iniziale
```

Nome di una variabile:

- Può contenere **lettere, cifre, underscore e simbolo del dollaro**;
- Devono **iniziare** con una **lettera**;
- Il linguaggio JS è **case-sensitive** (y e Y sono variabili tra loro diversi);
- Le **parole riservate** (come le parole chiave JS) *non* possono essere **usate come** nomi di una **variabile**.

Dichiarazione di una variabile

La dichiarazione di una variabile può essere fatta **prima di assegnarle** un valore (assegnamento o inizializzazione):

```
var nome; // abbiamo dichiarato la variabile  
nome = "studente"; // gli abbiamo assegnato un valore
```

Quando una **variabile** viene **solo dichiarata**, ad essa viene assegnato in automatico il valore **undefined**, il quale permarrà fintanto che non gli verrà assegnato un valore.

Dichiarazione e assegnazione di una variabile

È possibile però inizializzare la variabile direttamente insieme alla dichiarazione.

In questo caso la sintassi generale è:

```
var nome_variabile = espressione;
```

con la quale viene dichiarata una variabile il cui nome è `nome_variabile` con il valore di `espressione`. Per `espressione` qui si intende una sequenza di operatori, variabili e/o dati che restituisca a sua volta un valore. Per esempio:

Esercizio: nomina un giocatore

- Dichiara una variabile `nome`
- Assegnagli un valore
- Dichiara una variabile `cognome` e assegnagli un valore.
- Crea la variabile `nomeGiocatore` e assegnagli il valore di nome e cognome concatenati.

Variabili (let)

Prima dell'introduzione dello standard ES6 del 2015 di JS, l'unico modo per dichiarare una variabile era tramite l'utilizzo della parola chiave `var`.

Oggi abbiamo a disposizione anche le dichiarazioni `let` e `const`.

In particolare, l'uso della dichiarazione `let` è oggi preferito alla dichiarazione con `var`.

In altre parole, la dichiarazione tramite la parola chiave `var` è ancora possibile, seppure ormai non più consigliata (deprecata).

Costanti (const)

Una variabile il cui valore non può cambiare deve essere dichiarata con la parola chiave `const` (costante).

Ad esempio scriviamo le 3 righe di JS:

```
const nome = "Giulio";
nome = "Mario"; // restituisce un msg di errore
const nome = "Mario"; // restituisce un msg di errore
```

La parola chiave `const`, analogamente a `let` è legata al blocco dentro cui è dichiarata. Inoltre, **le variabili definite con const devono essere necessariamente inizializzate nel momento della dichiarazione** (dichiarazione e assegnazione sincrona).

Variabile: dichiarazione globale e locale

Ambito Globale

- La variabile è accessibile **ovunque** nel programma
- Dichiara fuori da funzioni e blocchi
- Utile per valori che servono in più parti del codice

Ambito Locale

- La variabile è accessibile solo in una **specifica porzione** di codice
- Dichiara dentro funzioni o blocchi
- Aiuta a evitare conflitti tra nomi di variabili
- Migliora la gestione della memoria

Variabile: dichiarazione globale e locale

```
var a = 3; // variabile globale visibile in tutto il codice
let c = 0; // variabile globale visibile in tutto il codice
{
  let b = -1; // variabile locale interna al blocco
  console.log(a); // visualizza 3
  console.log(b); // visualizza -1
}
console.log(a); // visualizza 3
console.log(b); // b is not defined
```

Esempio Pratico

```
let globale = "Sono visibile ovunque!" // Globale

{
  let locale = "Sono visibile solo qui!" // Locale
  console.log(globale) // ✓ Funziona
  console.log(locale) // ✓ Funziona
}

console.log(globale) // ✓ Funziona
console.log(locale) // ✗ Errore: locale non esiste qui
```

Esercizio: Scope delle variabili

```
{  
  let punteggio = 100;  
  // 1. Crea una variabile globale per il nome  
  // 2. Crea una variabile locale per il livello  
  // 3. Prova ad accedere alle variabili dentro e fuori dal blocco  
}
```

Nomi riservati

Alcuni nomi sono riservati in JavaScript e non possono essere usati come nomi di variabili o funzioni. Questi includono le parole chiave del linguaggio, come `var`, `let`, `const`, `function`, `if`, `else`, `for`, `while`, `return`, ecc.

```
var var = 5; // SyntaxError: Unexpected token 'var'  
var function = function() {};// SyntaxError: Unexpected token 'function'  
var let = 10; // SyntaxError: Unexpected token 'let'
```

Variabile: tipo di dati

Il JS è un linguaggio **debolmente tipato**, cioè la stessa variabile può contenere dati di tipo diverso.

Il tipo della variabile (o dei parametri/argomenti di una funzione) non viene dichiarato esplicitamente, ma definito implicitamente alla prima assegnazione. Ad esempio:

```
let x = 10 ; //x è una variabile di tipo «numero» ( number )
```

JS converte automaticamente i tipi di variabile durante l'esecuzione (quando necessario).

Variabile: tipo di dati

JS ha a disposizione **8 tipi di dato**.

Il tipo *number* viene utilizzato sia per i numeri interi che per quelli in virgola mobile. Entrambi sono immagazzinati in memoria come numeri decimali (floating point) e sono sempre in double precision (64-bit).

Per avere informazioni sul tipo di una certa variabile possiamo utilizzare l'operatore **typeof**.

- number
- string
- boolean
- object
- null
- undefined
- bigint
- symbol

```
console.log(typeof 0);      // number
console.log(typeof "Nome"); // string
console.log(typeof false);  // boolean
console.log(typeof 10n);    // bigint
console.log(typeof undefined) // undefined
console.log(typeof Math);   // object
console.log(typeof [1,2,3,4]); // object
console.log(typeof {name:'John', age:34}); // object
console.log(typeof new Date()); // object
console.log(typeof null); // object
console.log(typeof Symbol("id")); // symbol
```

Operatori aritmetici

Operatore	Descrizione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
**	Elevamento a potenza (simile al linguaggio fortran)
%	modulo (o resto intero della divisione) Es. $x=5\%2$ fornisce come risultato $x=1$
++	Incremento (di 1) Es. se $x=2$ e poi pongo $x++$ ottengo come risultato $x=3$
--	Decremento (di 1) Es. se $x=2$ e poi pongo $x--$ ottengo come risultato $x=1$

x++ e **++x** sono entrambi operatori di incremento che vengono utilizzati per aumentare il valore di una variabile di 1, ma c'è una differenza tra i due:

in **x++** il valore della variabile x viene usato prima che venga incrementato

```
let x = 3;  
console.log(x++); // Output atteso: 3  
console.log(x); // Output atteso: 4
```

mentre in **++x** il valore di x viene incrementato prima, e poi viene usato

```
let x = 3;  
console.log(++x); // Output atteso: 4  
console.log(x); // Output atteso: 4
```

Operatori di assegnamento composti

Oltre all'operatore uguale `=` esistono altri operatori di assegnamento (detti operatori di assegnamento composti) derivanti dalla combinazione dell'uguale con degli operatori aritmetici (dalla terza riga in poi in tabella).

L'uso degli operatori di assegnamento composti rende il programma più compatto, seppure non sempre altrettanto leggibile.

Operatori di assegnamento composti

Operatore	Esempio	Equivalente a
=	$x = y$	$x = y$
$+=$	$x += y$	$x = x + y$
$-=$	$x -= y$	$x = x - y$
$*=$	$x *= y$	$x = x * y$
$/=$	$x /= y$	$x = x / y$
$%=$	$x %= y$	$x = x \% y$

Operatori relazionali o di confronto

Tutti gli operatori di confronto restituiscono un valore booleano:

- `true` – significa “si”, “corretto” o “vero”;
- `false` – significa “no”, “sbagliato” o “falso”.

Operatore	Descrizione
==	uguale a
====	uguale valore ed uguale tipo (strettamente uguale)
!=	diverso da
!==	diverso valore e diverso tipo (strettamente diverso)
>	maggiore di
<	minore di
>=	maggiore o uguale di
<=	minore o uguale di
?	operatore ternario (assegna un valore a una variabile in base a una certa condizione)

Operatori logici

Gli operatori logici sono utilizzati per determinare la logica tra le variabili o i valori.

congiunzione logica AND: `&&`

Cosa fa

Richiede che siano vere **entrambe** le espressioni per restituire `true`

```
let x = 7;
if ((x >= 5) && (x <= 8)) {
    console.log("x è compreso tra 5 e 7");
}
```

congiunzione logica OR: ||

Cosa fa

Richiede che sia vera **almeno una** delle espressioni per restituire `true`

```
let ruolo = "admin";
if ((ruolo == "admin") || (ruolo == "editor")) {
    console.log("Hai i permessi necessari");
}
```

congiunzione logica Not: !

Cosa fa

Inverte il valore di verità dell'espressione:

- Se è `false` la cambia in `true`
- Se è `true` la cambia in `false`

```
let isLoggedIn = false;  
if (!isLoggedIn) {  
    console.log("Utente non autenticato");  
}
```

Conversioni di Tipo

```
// Conversioni implicite
```

```
5 + "10"    // "510"
```

```
"5" * 2    // 10
```

```
true + 1   // 2
```

```
false + 1  // 1
```

```
// Conversioni esplicite
```

```
Number("123") // 123
```

```
String(123)   // "123"
```

```
Boolean(1)    // true
```

Esercizio: Concatenazione e conversioni di Tipo

Dati:

```
let numero = 42;  
let stringa = "3";  
let booleano = true;
```

1. Concatena *numero* e *stringa*
2. Moltiplica *stringa* e *numero*
3. Somma *booleano* e *numero*
4. Converti esplicitamente il valore *stringa* in un *numero*