

Laboratorio I

a.a. 2025/2026

Corso C

Introduzione a JavaScript & tipi di dato

Link al Teams



Questionario



Contenuti

- Breve storia di JavaScript
- Tipi di dato e operatori
- Variabili
- Primi esempi e primi esercizi



JavaScript

- Introdotto in 1995 per Netscape (browser)
- Web programming, supportato da tutti i browser
- Inizialmente client-side
- Adesso utilizzato sia client che server-side
 - Node.js - piattaforma per sviluppo e esecuzione di programmi in JavaScript
- Vari update negli ultimi 25 anni
- ECMAScript 2025 - ultima versione





JavaScript

- Linguaggio interpretato
 - Interprete: browser, Node.js
- Flessibile, con poche restrizioni
 - weakly-typed
 - Svantaggio: difficile trovare gli errori
 - Vantaggio: facile sviluppare soluzioni inedite



Struttura di un programma

- Programma
 - Implementazione di un algoritmo per risolvere un problema
 - Una serie di **comandi (statement)** da eseguire in ordine
 - Comandi
 - devono seguire delle regole di **sintassi** - e.g. finiscono con ‘;’
 - automatic semicolon insertion - inserimento automatico del ;
 - vengono interpretati dall’interprete ed eseguiti - hanno una semantica
 - Esempi di comandi:
 - Calcolo della somma di due numeri
 - Mostrare a video un testo - console.log
 - Memorizzare un valore in una **variabile**

Tipi di dato

- I comandi fanno delle operazioni con dati in memoria
- I dati possono essere di tipo diverso
 - Numeri
 - Valori logici (Boolean)
 - Testo
- Le operazioni si fanno usando operatori con i dati
 - ***Espressioni*** - hanno un valore finale
- L'operatore `typeof` - per trovare il tipo

Espressioni vs comandi

- Espressioni
 - Hanno un valore finale
 - Possono essere usate per comporre altre espressioni, o liste di espressioni
 - $5+4$, $b=6$, $a=b+3$, $a>=b$
- Comandi
 - Espressioni
 - Altri comandi che non hanno un valore finale
 - non possono essere usate per comporre espressioni
 - Hanno un effetto collaterale (di solito)
 - `console.log(5);`

Tipi di dato

- Numeri (*number*)
 - Interi: 2, -5, 0
 - Reali: 2.5, 8.3, -9867.7
 - Numeri speciali:
 - NaN, Infinity, -Infinity
 - Operatori
 - +, -, *, /, % ($a \% b$ = resto divisione intera tra a e b)
 - >, <, >=, <=, ==

Tipi di dato

- Valori logici (Booleani)
 - true, false
 - Valore di verità di espressioni (e.g. confronti tra valori)
 - Operatori:
 - AND: `&&`, OR: `||`, NOT: `!`
- Esempi: tabella di verità di una funzione Booleana

Tipi di dato

- Testo (*stringa*)
 - Letterali delimitati da ‘‘ o ““ o ` `
 - Esempi:
 - 'Siamo al corso di laboratorio', '6', '87.6'
 - "Ciao", "0", "1425.6"

Tipi di dato

- Testo (*stringa*)
 - `Hello world`
 - Espressioni all'interno della stringa (*template literal*): `2+5 = \${2+5}`

Template literals: stringhe delimitate da *backticks* (`), che permettono l'inclusione e la valutazione di espressioni (mediante *sostituzioni*)

Tipi di dato

- Testo (*stringa*)
 - Operatori
 - + concatenazione
 - >, <, ==, <=, >=
 - Regole complesse per gestire questi operatori tra stringhe e numeri
 - <https://262.ecma-international.org/5.1/#sec-11.8.5>
 - <https://262.ecma-international.org/5.1/#sec-9.3>

Variabili (variables, bindings)

- Le espressioni vengono valutate una per una
- Il valore di un'espressione può essere assegnato a un nome - usando `let` o `var` o `const`
- Valori `undefined` e `null`

A variable that has not been assigned a value is of type **`undefined`**

The `null` value represents the intentional absence of any object value. It is one of JavaScript's [primitive values](#) and is treated as [falsy](#) for boolean operations.

Approfondimento: **falsy** values

- Ce ne sono 6:
 - **false** — boolean false
 - 0 — numero zero
 - “” — stringa vuota
 - **null**
 - **undefined**
 - **NaN** — Not A Number

Conversione di tipo

- È possibile trasformare una variabile/ un valore di un tipo in un altro tipo (**casting**)
 - `4 → '4'`
 - `"65.5" → 65.5`
- `Number()`, `String()`, `Boolean()`
- Attenzione: il cast restituisce sempre un valore (che può essere `NaN`)

Approfondimento: Conversione automatica di tipo

- JavaScript permette di fare delle operazioni con dati di tipo diverso
- I valori vengono automaticamente convertiti allo stesso tipo
 - Regole non molto semplici

Approfondimento: Conversione automatica di tipo

Per approfondimenti sui casi più complessi di *type coercion*, potete consultare le seguenti risorse:

- in generale:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#type_coercion
- sul trattamento dei numeri:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number#number_coercion
- sul trattamento delle stringhe:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#string_coercion
- sul trattamento dei booleani:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean#boolean_coercion

Approfondimento: Conversione automatica di tipo

```
let a='5', b=6;  
let c=a+b;  
console.log(c);
```

```
let a='5', b=6;  
let c=a-b;  
console.log(c);
```

Approfondimento: Conversione automatica di tipo

```
let a='5', b=5;  
console.log(a==b);
```

true

```
let a='5', b=5;  
console.log(a!==b);
```

true

```
let a='5', b=5;  
console.log(a===b);  
|
```

false

Operatori di uguaglianza

```
let a='5', b=5;  
console.log(a==b);
```

```
let a='5', b=5;  
console.log(a!==b);
```

```
let a='5', b=5;  
console.log(a===b);  
|
```

- == e !=
 - Uguaglianza e disuguaglianza (del valore)
 - Fa anche **type coercion**: due valori sono confrontati solo dopo conversione a un tipo comune
- === e !==
 - Uguaglianza e disuguaglianza **stretta** (di *valore e tipo*)

Null vs undefined vs NaN

- `null == null // true`
- `undefined == undefined // true`
- `null == undefined // true`

Confrontando `null` con qualsiasi altro valore, si otterrà sempre `false`

```
NaN == null // false  
NaN == undefined // false  
NaN == NaN // false (!!!!!)
```

Approfondimento: BigInt e tipi di dati non primitivi

- **BigInt:**
 - Numeri di lunghezza arbitraria, $-(10^{53}-1) \leq n \leq 10^{53}-1$
 - `let a = 63876534274856...564545863423n;`
- Tipi di dati non primitivi
 - Array: `let x=[3, 6, 2, 8, 9];`
 - Oggetti

```
let studente={nome: "Toto", cognome:"Bruno"};
console.log(studente.cognome);
```

Input and output

- Un programma produce la soluzione a un problema
- Il problema può avere dei parametri - dati iniziali su cui lavorare
 - INPUT
- Utile passare questi parametri dall'esterno - non serve cambiare il codice per ogni istanza di problema
 - E.g. `prompt()` - prende input da tastiera ← **non disponibile su tutti i sistemi**
- OUTPUT - risultato del programma va mostrato o salvato in qualche modo
 - E.g. `console.log()` - mostra a video nella console

Input and output: esempio

- Scriviamo un programma che risolve un'equazione di primo grado
 - $a*x+b=c ; x=?$
 - a, b, c sono l'input del programma
 - x è l'output

Esercizi

- Scrivere un programma che stampa la tabella di verità per la funzione Booleana ‘OR’
- Scrivere un programma che, data una temperatura in Celsius, calcola e mostra a video la temperatura in Fahrenheit ($F = C \times 1.8 + 32$)
- **Scrivere un programma che, dato un numero di secondi, calcola e mostra a video il numero di ore, minuti e secondi inclusi.**

Tipi di dato e valori letterali

JavaScript consente di denotare valori letterali di determinati tipi:

- **Boolean**: `true`, `false`
- **Undefined**: `undefined` (→ non è stato definito nessun valore)
- **Null**: `null` (→ so che non c'è un valore)
- **Number**: `1`, `5`, `-53.38`, `12.3e4` (numeri “normali”, $-(10^{53}-1) \leq n \leq 10^{53}-1$)
- **BigInt**: `900719925474344169871610992n` (interi a precisione infinita)
- **String**: `“ciao ugo”`, `‘हिन्दू’`, `‘hello’`, ``x vale ${x}`` (sequenze di caratteri)
- **Object**: `{ nome:“Ugo”, età:19 }` (mappe da chiavi a valori - anche dizionari)
 - **Array**: `[1, 5, 8, 12, 21, 33]`, `[1, “ciao!”, { x:12, y:22 }, [“banana”]]` (liste con indici numerici)
 - **Function**: `x=>2*x`, `(a,b)=>a+b`, `x => { if (x>0) return x; else return -x }` (funzioni)

Una grammatica per i letterali

Finora abbiamo fornito *esempi* di come si scrivono i letterali, ma ciò non è soddisfacente... vogliamo ottenere la massima precisione!

Nel corso di P&A avete visto il concetto di **grammatica**: usiamolo!

La **Backus-Naur Form** (BNF) è un modo di descrivere sinteticamente la grammatica di un linguaggio

La grammatica è data da un insieme di **produzioni**, ciascuna delle quali ha la forma:

classe ::= definizione₁ | definizione₂ | ... | definizione_n

e ogni definizione è una sequenza di simboli terminali e *classi*

Esempio: grammatica per i BigInt (semplificata)

⇒ *letterale_bigint* ::= *intero* n

◦ ◦ ◦

intero ::= *cifra* | *cifra intero*

cifra ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Attenzione! Questa non è la
vera grammatica dei BigInt...
vedremo dopo i dettagli

Partendo dal **simbolo iniziale** *letterale_bigint*, questa grammatica produce frasi come 8374n, 2n, 0n, 00038n, 9823410713087529813874221245345n

FAQ: Qual è la differenza tra let, var e const?



Q & A