Corso JavaScript

salvatore.laspata@cubemail.it

Definizione

JavaScript è un linguaggio di **scripting** orientato agli **oggetti** e agli **eventi** che viene **interpretato** ed eseguito dal **Client** (Browser)

Tutte le istruzione **JS** devono essere inserite all'interno del tag `<script>. . .</script>` in una pagina **HTML** oppure in un file dedicato con estensione **.js**

```
<!DOCTYPE html>
<html>
<script type="text/javascript"
src="./main.js"></script>
<head>
   <title>corso javascript</title>
</head>
<body>
   <script>. . .</script>
</body>
</html>
```

Browser e non solo

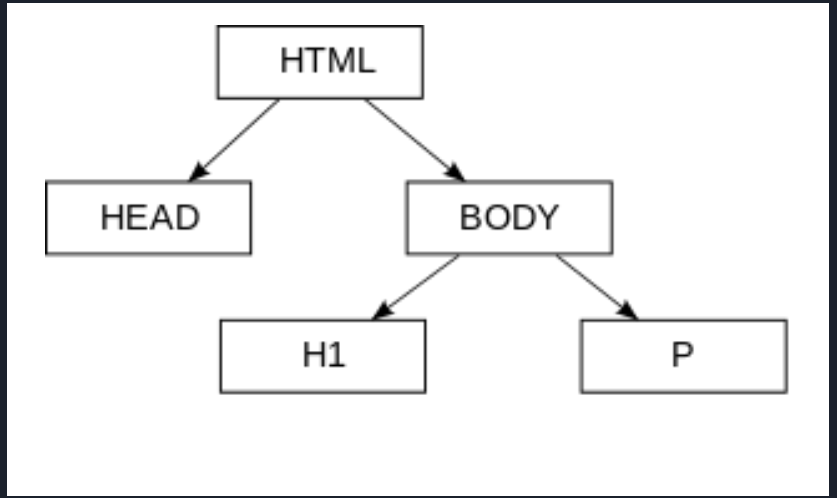
Ad oggi il javascript viene usato ampiamente nel mondo web, sia vanilla che tramite l'utilizzo di **framework web** (sempre più numerosi).

Grazie all'astrazione del motore V8 di Chrome che permette l'esecuzione del javascript anche lato server **(Node JS).**

Javascript nel browser

DOM
Document Object Model

È lo standard ufficiale del **W3C** che permette la rappresentazione dell'html in modo strutturato come modello orientato agli oggetti.



Commenti

Come si scrivono?

JavaScript prevede delle sequenze di caratteri per inserire commenti nel codice.

```
// commento singolo
/*
   commento su
   più righe
*/
```

Variabili → var | let
Costanti → const

Le variabili **var** possono essere aggiornate o ri-dichiarate dentro il loro ambito; le variabili **let** possono essere aggiornate ma non re-dichiarate; le variabili **const** non possono essere né aggiornate né re-dichiarate.

L'hoisting porta tutte le dichiarazioni in cima al loro ambito. Ma mentre le varabili var sono inizializzate con undefined, le variabili let e const non sono inizializzate.

Mentre var e let possono essere dichiarate senza essere inizializzate, const deve essere inizializzato durante la dichiarazione.

Come si scrivono?

Le variabili sono delle **“scatole”** che possono contenere un qualsivoglia valore, numerico testuale o strutturale.

Le variabili, per definizione, possono variare durante l’esecuzione dell’applicazione.

```
var var1 = 1;

let var2 = "testo";

const PI = 3.14;
```

Tipi di Dati

i tipi dati dati in JavaScript sono:

- **boolean** (operatore logico)
- **number** (numeri interi o reali)
  - Infinity, -Infinity e NaN
- **bigInt**
- **string** (caratteri e stringhe)
- **undefined**
- **null**
- **Object**
- **Array**
- **...**

Come si scrivono?

```
const booleano    = true;
const numero      = 1;
const bigNumero   = 1231231n;
const stringa     = "testo";
let indefinita;   // undefined
const nonDefinita = null
const oggetto = {
  prop1: 1,
  prop2: '2'
}
const array = [1,'2',3,'quattro']
```

Operatori

Gli operatori di controllo **matematici** verificano se la condizione può essere vera.

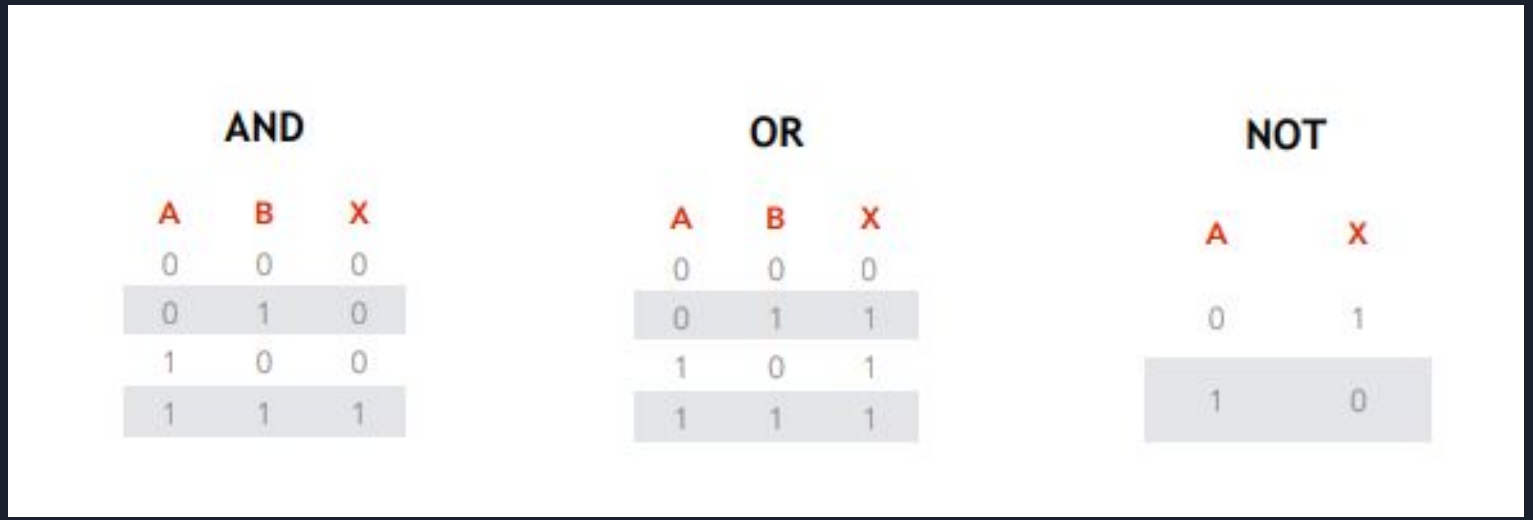
== (uguaglianza)
=== (uguaglianza di valore e di tipo)
< (minore)
<= (minore e uguale)
> (maggiore)
>= (maggiore e uguale)
!= (diverso)
!== (differente valore e tipo

Gli operatori di incremento servono ad incrementare o decrementare di 1 una variabile.

++ (incremento - prefisso e postfisso)

-- (decremento - prefisso e postfisso)

SCHEMA OPERATORE BOOLEANO



AND

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

NOT

A	X
0	1
1	0

Strutture Condizionali - if

Come si scrivono?

Le strutture condizionali permettono di eseguire un blocco di codice se le condizion sono soddisfatte vera.

```
// condizione semplice
if (condition) {
  console.log('condition soddisfatta')
} else if (condition2)  {
  console.log('condition2 soddisfatta')
} else {
  console.log('nessuna delle due condizioni'+
              'sono state soddisfatte')
}
// operatore ternario
condizione ?
  console.log('condizione soddisfatta') :
  console.log('condizione non soddisfatta (else)')
```

Strutture Condizionali - switch

switch permette in modo più chiaro e pulito l'esecuzione di **più if in cascata**, a patto che la condizione rimanga sempre la medesima.

Come si scrivono?

```
switch (condition) {
    case value:
        . . .
        break;

    default:
        break;
}
```

Strutture di dati - Object

Essendo JavaScript basato sul DOM, gli elementi stessi del DOM costituiscono degli oggetti lato client, ovvero degli elementi presenti nella pagina HTML che hanno delle proprietà e dei metodi predefiniti.

Un oggetto non è altro che una struttura con delle proprietà e dei metodi.

Come si scrivono?

```
var obj = new Object();
obj.name = "Pippo";
obj.surname = "Pluto";
console.log(obj.name); // Pippo

var obj = {
   name: "Pippo",
   surname: "Pluto",
   fullName: function(){
     return this.name + " " + this.surname;
   }
};
obj.fullName(); // Pippo Pluto
```

Strutture di dati - Array

Un array è una collezione di dati che permette l'immagazzinamento in una struttura ordinata di dati eterogeni.

Un array non è altro che un

Come si definiscono gli Array?

```
const array = new Array()

const array2 = []

const array3 = [1,2,3]

const array4 = [1, 'antani', 3, false]

const array5 = [{a: 1, b: 2}, {a: 2, b: 3}]
```

Come si leggono gli Array?

→

Cicli

L'istruzione all'interno di **while** viene eseguita fintantoché la condizione risulterà essere vera.

Il do while permette di eseguire la primissima operazione del do per poi verificare la veridicità del while

Un'altra tipologia di ciclo è il cliclo **for** che permette di scorrere in maniere ordinata una lista di valori (tramite indici)

Esistono anche for-of for-in forEach

```
const list = ['a', 'b', 'c']
let i = 0
while (i < list.length) {
  console.log(list[i]) //value
  console.log(i) //index
  i = i + 1
}

let u = 0
do {
  console.log(list[u]) //value
  console.log(u) //index
  u = u + 1
} while (u < list.length)

for (let ii = 0; ii < list.length; ii++) {
  console.log(list[ii]) //value
  console.log(ii) //index
}

// per uscire del loop ⟶ break & continue
```

Strutture di dati - Function

Una funzione permette di creare un blocco di istruzioni che può essere contraddistinte da un nome in modo da poter essere richiamato ed eseguito successivamente.

Esistono anche funzioni “anonime” che non sono identificate da nessun nome

NB lo scope del **this**

```
// Come si scrivono?
function getFullName(nome, cognome) {
  return nome + " " + cognome
}
getFullName("salvatore", "la spata");
// salvatore la spata

const somma = function name(a, b) {
  return a + b;
}
somma(44,22)
// 66

// arrow function
const arrow = (...args) => { return args }
arrow(1,2,3,4,5)
// [1, 2, 3, 4, 5]
```

Strutture di dati - Classi

Una funzione permette di creare un blocco di istruzioni in modo da poter essere richiamato ed eseguito successivamente.

Questo principio è alla base della programmazione ad oggetti (**OOP**)

```
class Rettangolo {
  constructor(base, altezza) {
    this.base = base;
    this.altezza = altezza;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.base * this.altezza;
  }
}

const quadrato = new Rettangolo(10, 10);
quadrato.area(); // 100
```

Strutture di dati Bonus track

Esistono altri oggetti, più o meno complessi, che permettono la gestione della base dati in modo efficace.

```
const mySet1 = new Set()
const myMap1 = new Map()
const myPromise1 = new Promise((resolve, reject) => {
  resolve('ok')
})
const mySymbol = Symbol('mySymbol')
const myGenerator = function* () {
  yield 1
  yield 2
  yield 3
}
const myProxy = new Proxy({}, {
  get: function (target, name) {
    return name in target ? target[name] : 42
  }
})
```

Documentazione

https://developer.mozilla.org

Grazie per l’attenzione!