Umberto Emanuele

JavaScript 1

Novembre 2023

Introduzione a Javascript

Introduzione al corso

- Cosa è JS e i suoi campi applicativi
- Sintassi e integrazione nei progetti
- Concetti base e avanzati
- OOP in JavaScript
- Vanilla JS ed ECMA
- Applicazioni pratiche

Tutto nacque nel lontano 1995...

- JavaScript nasce per l'esigenza di offrire una maggiore interattività con gli oggetti del DOM.
- L'iniziativa nasce dal team di sviluppo del browser Netscape.
- Questa sua genesi determina dall'inizio delle caratteristiche che ancora oggi contraddistinguono il nostro linguaggio.

JavaScript è

- Un linguaggio normalmente letto ed eseguito con compatibilità in tutti i browser nelle ultime release
- Un linguaggio lato client
- Un linguaggio di scripting
- Un linguaggio per lo sviluppo del front end di siti e applicazioni

Cosa posso fare con JavaScript

- Il campo di applicazione nativo è lo sviluppo lato front end.
- JS viene utilizzato per applicare strutture di programmazione ad una architettura web.
- Quindi posso utilizzare JS per interagire con tag HTML e regole CSS per elevare il grado di interattività con una pagina web di un sito o di una applicazione.

Un confronto ci sarà utile

HTML5

COSA È
L'OGGETTO
WEB

JS

COSA FA
L'OGGETTO
WEB

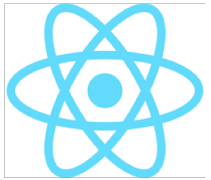
CSS3

COME È
L'OGGETTO
WEB

L’ ecosistema di JavaScript

È alla base di molti framework utilizzati nelle più moderne tecnologie:

- Angular
- React / React Native
- Vue.js



Utilizzo di Javascript nei progetti frontend

Creazione di un file javascript ed esecuzione

Gli script JS possono essere scritti in file esterni che devono essere integrati nel progetto.

I file js possono contenere una singola applicazione o racchiudere più script indipendenti.

Una volta richiamato il file, lo script sarà a disposizione del progetto.

Esempio

file.js =>

```
function funzione() {
  var x = 2;
  var y = 3;
  var somma = x + y;
}
```

Inseriamo il file esterno nel progetto

Il file deve essere inserito nel tag <script> con l'attributo src per indicare il percorso del file.

Anche se il file può essere richiamato pure dal <head> è consigliabile collocare lo script al livello del body, immediatamente prima della sua chiusura.

Questo permette il completo caricamento del DOM prima di invocare lo script e con positivi effetti sulle performance del progetto.

```
<body>
———
    <script src="file.js"> </script>
</body>
```

Inseriamo lo script nel progetto

Lo script può essere inserito sia nel **<head>** sia nel **<body>** all'interno del tag **<script>**

```
<head>
_____
    <script>
     function somma()        {..}
    </script>
_____
</head>
```

```
<body>
_____
   <script>
function somma()
{..}
   </script>
</body>
```

Gestione dell’output

Direttive di output

Metodi

innerHtml => elemento html

document.write() => documento

window.alert() => box pop up

console.log() => debug nel browser

Per direttive di output si intendono quelle istruzioni specifiche che restituiscono a display il risultato di uno script, tipicamente di un metodo applicato ad un elemento.

Ogni direttiva di output ha un suo target di riferimento

```
<h1 id='prova'></h1>

<script>

document.getElementById
('prova').innerHTML = 3 + 4;

</script>
```

Elemento html

```
<script>

document.write(3 + 4);

</script>
```

Documento

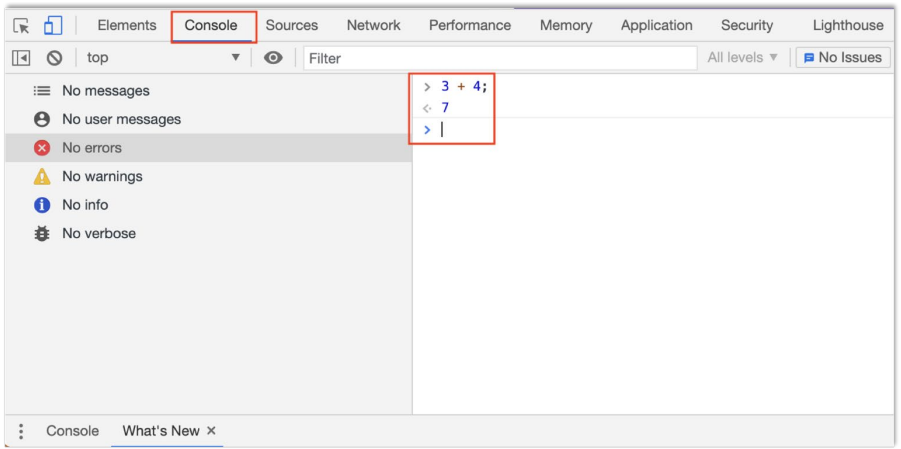
```
<script>

window.alert(3 + 4);

</script>
```

la keyword window può essere omessa

Pop up

console.log

```
<script>
console.log(3 + 4);
</script>
```



La visualizzazione tramite la console del browser è uno degli strumenti più utili in fase di sviluppo.

Viene utilizzata come forma di controllo della correttezza del codice.

È una forma di debug.

Sintassi e buone pratiche di scrittura

La sintassi

Keywords : le parole con le quali indichiamo le strutture di programmazione sono riservate, come tali non possono essere usate per definire una variabile o una funzione.

Esempio: **var** o **function** sono keyowords riservate.

Numeri e **Stringhe** : sono i due tipi principali di valori. I numeri, interi o decimali, vengono indicati senza apici. Le stringhe, cioè sequenze di caratteri, devono essere indicate fra apici.

```
var numero = 5;
```

```
var nome = “Simona”;
```

```
var numero = “5”;
```

```
var nome = Simona;
```

Per sintassi si intende una serie di regole utilizzate dal linguaggio di programmazione per costruire le sue strutture.

JavaScript è un linguaggio case sensitive, perciò:

Nome e nome saranno intesi come tipi differenti.

Nominare i costrutti

Sintassi corretta

```
var mioNome;
var mio_Nome;
var MioNome;
var Nome;
var mionome;
var nome;
var nome1;
```

Sintassi non corretta

```
var mio Nome;
var 1Nome;
var mio-Nome;
var mioNome;
```

I nomi scelti per funzioni, variabili e le altre strutture di programmazione, devono essere indicativi dello scopo, chiari, semplici e facilmente interpretabili.

Non è buona pratica nominare con singole lettere e numeri.

Altra importante regola è la corretta e precisa indentazione del codice.

Commenti al codice

Commenti a singola linea

```
var nome = “Simona” // questa è una variabile

// questa è una variabile
var nome = “Simona”

//var nome = “Simona”
```

Commenti a linea multipla

```
/*
Questo è un commento
su più linee
*/
```

I commenti al codice sono stringhe di testo usate per descrivere o spiegare parte del codice.

I commenti non hanno alcuna influenza sullo script e non sono visualizzati a display.

I commenti vengono usati anche per annullare una parte del codice per motivi di controllo e debug.

Le keywords

var	const	let	function
return	if	else	this
switch	while	do	for
case	class	true	false
extends	null	break	continue
default	new	super	

Avremo modo di conoscere l'uso e la funzione di ogni singola keywords nel procedere delle nostre lezioni.

Tutte le keywords si indicano in minuscolo.

Le variabili

Le variabili

- La variabile è una partizione di memoria dedicata alla conservazione di un valore.
- La variabile deve essere definita per essere utilizzata nel nostro codice.
- Il valore di una variabile può essere riassegnato.

```
var nome; // variabile definita
var nome = 'Mario'; // variabile definita e assegnata
```

La keyword per la variabile è var.

Vedremo però che il concetto e la funzione della variabile si può esprimere anche in altri modi.

Definire una variabile

La variabile è composta da:

keyword => var

nome definitore => nome

operatore di assegnazione => =

valore => Mario

```
var nome = 'Mario';
```

Lo script deve essere chiuso da ;

- Nelle variabili possiamo memorizzare valori di vario tipo: numeri, stringhe, booleani...
- I valori di tipo stringa (sequenza di caratteri), devono essere scritti fra apici, singoli o doppi.
- I valori di tipo numerico e booleani devono essere senza apici.

Variabili con valore numerico:

```
var numeroIntero = 10;
var numeroDecimale = 10.50;
```

Variabili con valore stringa:

```
var nome = 'Mario';
var cognome = "Rossi";
```

Variabili con valori booleani:

```
var altezza = true;
var lunghezza = false;
```

La variabile

```
var numero = "10";
```

sarà interpretata come una sequenza di carattere, quindi il valore non sarà di tipo numerico ma di tipo stringa

Ad una variabile possiamo assegnare anche il valore di una o più variabili:

```
var a = 10;
var b = a;
var c = a + b // in questo caso alla variabile è assegnato il valore della somma di a e b cioè 20
```

Possiamo eseguire operazioni matematiche utilizzando i valori di variabile:

```
var numero1 = 5;
var numero2 = 7;

Alcuni esempi:
console.log(numero1 + numero2); => 12
console.log(numero1 - numero2); => -2
Console.log(numero2 + 4); => 11

In questo caso, + e - sono interpretati come usuali operatori matematici.
```

È possibile mostrare il valore assegnato ad una variabile richiamandola all'interno di una direttiva di output oppure all'interno di una funzione, come vedremo.

```
var nome = 'Mario';
var anni = 30;
var luogo = 'Roma';

window.alert(nome); // sarà mostrato in una finestra il valore della variabile nome che è Mario

document.write(anni); // sarà mostrato a schermo il valore della variabile anno che è 30

console.log(luogo); // sarà mostrato in console il valore della variabile luogo che è Roma
```

La possibilità della riassegnazione è una delle caratteristiche principali nell'uso delle variabili

```
var numero1 = 5;

document.write(numero1);
Output => 5

numero1 = 3;
document.write(numero1);
Output => 3
```

```
var numero2 = 10;
numero2 = 15;

document.write(numero2);
```

L'output non sarà 10 ma 15 poiché il codice leggerà l'ultimo valore della variabile aggiornando l'output.

Le variabili possono essere concatenate, unendo nell'output i valori assegnati.

```
var nome = 'Mario';
var anni = 30;
var luogo = 'Roma';
```

Esempio di concatenazione:

```
document.write(nome+anni+luogo);
```

L'output visualizzerà Mario30Roma

In questo caso + non è interpretato come operatore matematico ma come operatore di concatenazione dei diversi elementi.

JS non tiene conto degli spazi quindi bisognerà indicarli nel codice. Per esempio:

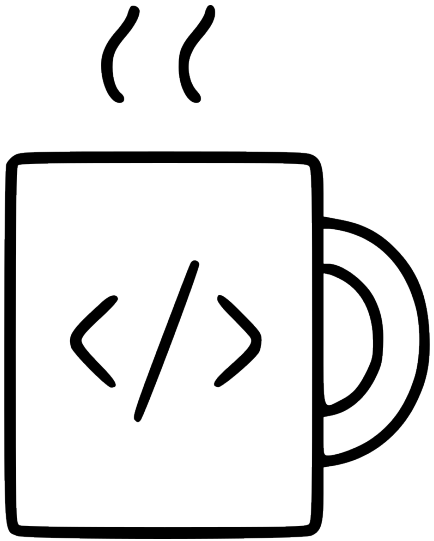
```
document.write(nome + " "+ anni+ " "+
luogo);
```

In questo caso avremo come output: Mario 30 Roma

Possiamo comporre frasi di senso compiuto:

```
document.write(
"Nome utente: "+ nome + ", anni utente: "+
anni+ ", luogo di nascita utente: "+ luogo);
```

In questo caso avremo come output: Nome utente: Mario, anni utente: 30, luogo di nascita utente: Roma



PAUSA

Ci vediamo alle ore 11.17

Let e const

Per le variabili possiamo usare anche le keywords:

let

const

Negli approfondimenti di ECMA avremo la possibilità di comprendere meglio i vantaggi dell'introduzione di queste due keyword.

let è utile quando vogliamo attribuire ad una variabile uno scopo (visibilità, accessibilità) legata al blocco, **Block Scope** , di codice all'interno della quale la variabile è dichiarata.

Esempio

```
var nome = "Mario";
Output => Mario
{
var nome = "Carla";
Output => Carla
}
Output => Carla

var nome = "Mario";
Output => Mario
{
let nome = "Carla";
Output => Carla
}
Output => Mario;
```

Una variabile dichiarata const si comporta nel medesimo modo di let in caso di Block Scope, ma non può essere riassegnata, quindi il suo valore non potrà che avere un riferimento costante.

```
const numero = 34567;
numero = 23876;  // il codice genera errore
numero =  numero + 10; // il codice genera errore

Una variabile const deve essere assegnata al momento delle definizione:

const numero;
numero = 25; // non consentito

Una variabile const ha la possibilità di essere usata in un blocco di
codice

var numero = 20;
Output => 20
{
const numero = 15;
Output => 15
}
Output => 20;
```

Data Types

Concetto di tipizzazione

In programmazione per tipizzazione si intende la distinzione dei diversi tipi di dati utilizzati nel codice.

La tipizzazione segue regole differenti nei linguaggi di programmazione.

In ogni linguaggio esistono modalità di corretta interpretazione dei tipi di dato.

In alcuni linguaggi è necessario che il tipo di dato sia espressamente indicato con una specifica keyword => in questo caso parliamo di tipizzazione forte.

La tipizzazione può essere **statica** o **dinamica** .

Nella tipizzazione statica una variabile rimane strettamente legata al tipo di valore assegnato.

Esempi di tipizzazione forte:

```
String nome = “Mario”;

int numero = 21;

float numero1 = 12;

bool esempio = true;
```

Tipizzazione dinamica

Nella tipizzazione dinamica una variabile nel corso del programma può modificare il tipo di dato assegnato.

```
var numero = 20;
console.log(numero); => 20

numero = 34;

numero = “parola”;
console.log(numero); => parola
```

JavaScript non è un linguaggio a tipizzazione forte, infatti, per la corretta interpretazione del codice, non abbiamo bisogno di specificare il tipo di dato.

Inoltre ammette la tipizzazione dinamica.

Nell’associare i valori di variabili diversamente tipizzate, JavaScript non segnala errori ma esegue una concatenazione:

```
var numero =  13;
var numero1 = 7;
var anno = “5”;

console.log(numero + anno); => 135
console.log(numero + numero1);
console.log(anno + numero + numero1); =>
5137
```

Dato **Stringa** in Javascript indica sia il singolo carattere che la sequenza finita di caratteri.
Le stringhe possono essere manipolate tramite metodi predefiniti che impareremo a conoscere e usare.

Dato **Numerico** in Javascript viene indicato senza distinzione fra intero e decimale. Il decimale si esprime con il punto.

var numero = 21.23;

Dato **Booleano** ammette due soli valori: true e false.
Questo tipo di dato è tipico dei controlli condizionali, come poi vedremo.

Caratteri escape e di nuova linea

Uso di apici interni alla stringa:
var stringa = "Mario ha detto: 'Ciao'!";

Gestione dell'apostrofo:
var stringa = "l'apice doppio"; => sintassi corretta
var stringa = 'l'apice doppio'; => sintassi non corretta
stringa = 'l\'apice doppio'; => sintassi non corretta con carattere escape

Il carattere escape \ può essere usato anche per creare una nuova riga.

Nulled e undefined

Null è una keyword utilizza che segnala l'assenza intenzionale di un oggetto.

È diverso dal valore numerico 0 in quanto indica una mancanza di identificazione.

Nella logica booleana (vero/falso) indica una condizione false

var nome = null;

Undefined è una proprietà che indica che una variabile non ha un valore assegnato oppure non è affatto definita.

Per esempio, usare una variabile non precedentemente definita all'interno di una funzione, genera un avviso di indefined.

Operatori di assegnazione e logici

Gli operatori

Gli **operatori** di programmazione rappresentano uno dei maggiori ausili nello sviluppo.

Determinano quale regola applicare relativamente ad una condizione in una porzione di codice.

Esempio: assegnare il simbolo + agli elementi di una variabile indica che vogliamo attribuire o una regola di concatenazione o una regola matematica di somma.

Tipi di operatori:

- Assegnazione
- Aritmetici
- Comparazione
- Logici

Gli operatori di assegnazione

OPERATORE	ESEMPIO	È UGUALE A
=	numero = numero1;	
+=	var numero1 += numero; => 12	var numero1 = numero + numero1;
-=	var numero1 -= numero; => 6	var numero1 = numero1 - numero;
*=	var numero1 *= numero; => 27	var numero1 = numero1 * numero;
/=	var numero1 /= numero; => 3	var numero1 = numero1 / numero;

```
var numero = 3;
var numero1 = 9;
```

Gli operatori di assegnazione nelle stringhe

```
var saluto = “Ciao”;

var saluto1 = “a tutti!”;
```

```
Saluto3 = saluto + ‘ ‘+ saluto1; => Ciao a tutti!

saluto = saluto + ‘ ‘+ saluto1; => Ciao a tutti!

saluto +=  ‘ ‘+ saluto1; => Ciao a tutti!
```

Gli operatori logici

Gli operatori logici sono utilizzati per regolare le condizioni in maniera strutturata, ottenendo una comparazione fra condizioni differenti.

```
numero == numero1;

numero < numero1 &&numero1 > numero;

numero > numero1 || numero < numero1;

!(numero == numero1);
```

Gli operatori logici indicano le condizioni logiche tra due variabili o due valori.

And => tutte le condizioni indicate devono essere vere. Si esprime con il simbolo &&

Or o l'una o l'altra delle condizioni deve essere vera. Si esprime con il simbolo ||

Not la condizione indicazione non deve essere vera. Si esprime con il simbolo !

```
var numero = 3;
var numero1 = 9;
```

challenge
NETWORK

shaping the skills of tomorrow

challengenetwork.it