Edgemony

Coding Bootcamp

Web fundamentals

Javascript - Parte I

Programma

Introduzione Javascript

variabili

logiche booleane

istruzioni condizionali

Introduzione a Javascript

Javascript è un linguaggio di programmazione indispensabile per le pagine web e viene utilizzato per rende interattiva la nostra applicazione web.

legato a Javascript si sente parlare spesso di ECMAScript, una delle versioni più citate è ES6.

ECMAScript è uno standard. Javascript è l'implementazione di questo standard.

Tra i principali usi troviamo:

- posizionamento dinamico di elementi HTML;
- validazione campi dei moduli;
- effetti grafici.

Inserimento nell'HTML

Esistono 3 modi per implementare Javascript in una pagina web:

Codice inline

```
<button onclick="alert('Se non scommetti, non vincerai mai');">
  Clicca
</button>
```

Blocchi di codice

```
<script>
  alert('Se non scommetti, non vincerai mai');
</script>
```

File Javascript esterni

```
<script src="app.js"></script>
```

Sintassi e semantica

La sintassi si riferisce alla struttura di un programma scritto in un linguaggio di programmazione.

La **semantica** enfatizza l'interpretazione di un programma in modo che il programmatore possa comprenderlo in modo semplice o prevedere il risultato dell'esecuzione del programma.

```
let firstName = 'Martin' // Sintassi e semantica corretta
let firstName = 5  // Sintassi corretta ma semantica errata
```

"Programma sempre come se il ragazzo che dovrà manutenere il tuo codice sia uno psicopatico violento che sa dove vivi."

John F. Woods

Comandi

Stampare in console o lanciare un alert

console.log('Se non scommetti, non vincerai mai.');
permette di stampare nel Dev Tool di Chrome, all'interno della sezione "console" , dei messaggi o dei valori.
In questo caso stamperà "Se non scommetti, non vincerai mai."

prompt('Quale pizza vuoi? ') ;
apre una finestra di dialogo con un input e ritorna il valore inserito nell'input

alert('Ottima scelta') ;
apre una finestra di avviso

Variabili

le variabili sono dei contenitori che ci permettono di salvare dei dati.

SINTASSI:

let myName = ‘Ignazio’;

in questo modo abbiamo dichiarato una variabile usando **let** ← una parola chiave di Javascript

e tramite “=” le abbiamo assegnato il valore “Matteo”

Tipi di dati Primitivi

le tipologie di dati che possiamo utilizzare sono diverse tra cui quelle di tipo primitivo:

- **string**
- **number**
- **boolean**
- **undefined**
- **null**

Tipi di dati Primitivi - String

le stringhe si distinguono perché sono racchiuse all'interno di virgolette e contengono una sequenza arbitraria di caratteri.

let myName = 'Ignazio'; ← singolo apice o virgoletta singola
let myName = "Ignazio"; ← doppi apici o virgolette doppie
let myname = `Ignazio `; ← apice inverso (backtick)

let paragraph = 'Se pensi che l\'avventura sia pericolosa, prova la routine: è letale.'

NOTA:
il carattere \ (backslash) prima dell'apostrofo, chiamato escape, dice a Javascript che non è la virgoletta di chiusura

Tipi di dati Primitivi - Number

Il dato di tipo Number assume valori numerici interi o decimali

```
let num = 5;          // positivo intero
let num = -3;         // negativo intero
let num = 0.4;        // positivo decimale
let num = -8.3;       // negativo decimale
```

Si possono eseguire tutte le operazioni matematiche:

```
let sum = 5 + 25;
let multiple = 5 * 5;
let div = 25 / 5;
let mod = 25 % 5;
let floatNum = 5.10;
```

Tipi di dati Primitivi - Boolean

Un tipo boolean assume i soli valori della logica booleana vero e falso

```
let isYellow = true;          // valore logico vero
let isAvailable = false;      // valore logico falso
```

NOTA:
quando si creano delle variabili booleane è buona pratica nominarle come se fosse una domanda alla quale si può rispondere solo SI o NO.

Tipi di dati Primitivi - Undefined e null

Si tratta di tipi di dati che possono assumere un unico valore

let firstName; // se non assegnamo un valore ad una variabile javascript assegnerà **undefined** di default

let firstName = **undefined**; // indica una variabile non inizializzata, cioè a cui non è stato ancora assegnato un valore

Let secondName = **null** // indica che alla variabile non è stato assegnato deliberatamente un valore (valore non significativo)

Tipi di dati Primitivi - Conversione tra tipi (string)

Le variabili di tipo **Number e Boolean** hanno un metodo che permette di trasformare il loro valore in **string**

SINTASSI:

nomeVariabile**.toString()**;

```
let num = 5;
let bool = true;

num.toString(); // '5'
bool.toString(); // 'true'
```

Tipi di dati Primitivi - Conversione tra tipi (number)

è possibile convertire una **string** in number tramite **parseInt** o **parseFloat**

SINTASSI:

parseInt('stringa'**)**;

parseFloat('stringa'**)**;

```
let num = parseInt('2');              // num sarà uguale a 2
let num = parseInt('3.14');           // num sarà uguale a 3
let num = parseFloat('3.14');         // num sarà uguale a 3.14
let num = parseInt('36.5gradi');      // num sarà uguale a 36
let num = parseFloat('36.5gradi');    // num sarà uguale a 36.5
let num = parseInt('gradi36.5');      // num sarà uguale a NaN
let num = parseFloat('gradi36.5');    // num sarà uguale a NaN
```

NOTA:
questi metodi fermano la trasformazione appena trovano un carattere che non corrisponde ad un numero, da sinistra a destra.

se il primo valore non è valido tornano **NaN - Not a Number**

Tipi di dati Primitivi - Conversione tra tipi (number)

un metodo alternativo per eseguire la conversione di stringhe a numeri è l'utilizzo di **+**

```
let num = +'2';              // num sarà uguale a 2
let num = +'3.14';           // num sarà uguale a 3.14
let num = +'gradi36.5';      // num sarà uguale a NaN
let num = +'36.5gradi';      // num sarà uguale a NaN
```

NOTA:

a differenza di parseInt o parseFloat, restituisce NaN se trova un carattere non convertibile in qualsiasi posizione della stringa

Tipi di dati Primitivi - Conversione tra tipi (boolean)

un metodo per convertire qualsiasi tipo di dato in boolean è **!!** (doppio punto esclamativo)

```
let str = !!'2';          // str sarà uguale a true
let test = !!undefined;   // test sarà uguale a false
let num = !!9;            // num sarà uguale a true
```

Falsy	**Truthy**
0	+/-23
-0	“stringa”
“”	[] ← array vuoto
null	{} ← oggetto vuoto
undefined	
NaN	

Condizioni - Operatori relazionali

Sono usati per confrontare a livello logico la relazione tra due variabili e restituiscono l'esito come valore booleano

==	Uguaglianza	'Dario' == 'Mario'	
!=	Disuguaglianza	'Dario' != 'Mario'	
===	**Uguaglianza stretta**	2 === '2'	2 === 2
!==	**Disuguaglianza stretta**	2 !== '2'	2 !== 2
<	Minore	age < 18	
>	Maggiore	age > 18	
<=	Minore o uguale	age <= 18	
>=	Maggiore o uguale	age >= 18	

Condizioni - Operatori logici

è possibile costruire condizioni più complesse tramite una serie di operatori logici

&& ← operatore per l'**and** - **tutte** le condizioni devono essere vere per tornare vero

|| ← operatore per l'**or** (due pipeline) - **almeno una** condizione deve essere vera per tornare vero

! ← operatore per **not** (punto esclamativo) - la condizione **deve essere falsa** per tornare vero

esempio

l'input è valido **AND** l'utente è registrato → name !== "" **&&** isRegistered

l'input è valido **OR** l'utente è registrato → name !== "" **||** isRegistered

l'input **NON** è valido → ! l'input è valido → !isRegistered

Istruzioni condizionali - if, else, else if

È possibile definire delle condizioni al verificarsi delle quali deve o meno accadere qualcosa.

Per fare ciò ci si serve degli operatori **if, else if, else** ("se", "oppure se" e "oppure")

SINTASSI:

```
if ( age <  18 ) {
   // esegui azione A
} else if ( age > 18 ) {
   // esegui condizione B
} else {
   // esegui condizione C
}
```

NOTA:

il blocco **else if** ed **else** sono opzionali

```
if ( name !== 'Stefania' ) {
   console.log('Non sei Stefania');
}
```

Istruzioni condizionali - switch

Simile a if/else, lo switch viene utilizzato quando siamo di fronte a diverse alternative

SINTASSI:

```
switch (variabile) {
   case ipotesi 1:
      azione 1
      break
   case ipotesi2:
      azione2
      break
   case ipotesi3
      azione 3
      break
}
```

NOTA:

aggiungere **break** è importante per uscire dallo switch una volta eseguita l'azione

Istruzioni condizionali - Operatore ternario - ? :

Consiste in un modo "**compatto**" attraverso il quale scrivere un'istruzione condizionale.

Questa sintassi è molto comoda ed utile se si devono affrontare condizioni semplici.

SINTASSI:

condizione **?** istruzione1; **:** istruzione2;

ESEMPIO

let isAdult = age >= 18 **?** true **:** false;