--- Argomento 1 Dove inserire lo script---

L'inserimento di **codice inline**, consiste nell'inserire direttamente le istruzioni JavaScript nel codice di un elemento HTML, assegnandolo ad un attributo che rappresenta un evento.

```
<button type="button"
onclick="alert('Ciao!')">Cliccami</button>
```

Abbiamo assegnato all'**attributo onclick** dell'elemento button la stringa alert('Ciao!'). L'attributo onclick **rappresenta l'evento** del clic sul pulsante del mouse, quindi in corrispondenza di questo evento verrà analizzato ed eseguito il codice JavaScript assegnato. Nel caso specifico verrà visualizzato un box con la scritta Ciao!

Un altro approccio per l'inserimento di **codice inline**, utilizzabile però soltanto con i link

```
<a href="javascript:alert('Ciao!')">Cliccami</a>
```

L'approccio inline può risultare immediato perché mette direttamente in relazione il codice da eseguire con un elemento HTML. Risulta però scomodo quando il codice da eseguire è più complesso o abbiamo necessità di definire variabili e funzioni. In questi casi possiamo ricorrere **al tag <script> per inserire blocchi di codice in una pagina HTML**

```
<script>alert('Ciao!')</script>
```

Possiamo inserire blocchi di codice (e i relativi tag <script>) nella sezione <head> o nella sezione <body> della pagina HTML.

Il terzo approccio, quello più consigliato, consiste nel collegare alla pagina HTML, codice JavaScript presente in un **file esterno**. Questa tecnica permette di agganciare script e librerie in modo detto non intrusivo, con il vantaggio di una separazione netta tra la struttura del documento e il codice, come accade per i fogli di stile CSS, che separano struttura e presentazione.

<script src="./4b/script.js"></script>

Verifichiamo quest'indirizzo relativo: ./4b/script.js

dove il . indica dove ci troviamo e nella cartella 4b andiamo ad aprire il file script.

--- Argomento 2 scrittura del codice ---

Il codice JavaScript è composto da una sequenza di istruzioni che viene interpretata ed eseguita dall'engine.
In questa sequenza, ciascuna istruzione (o blocco di istruzioni) è delimitata da un punto e virgola (**;**)

*[a cosa fa riferimento la parola **engine**]*: Tutti i nostri dispositivi hanno un componente chiamato microprocessore che è il cervello degli stessi dispositivi e che si occupa di eseguire le istruzioni che sono in memoria.

Nel caso di Javascript sia lato client che lato server questo lavoro di traduzione viene fatto dagli engine (motori javascript).

```
<script>
let x = 5;
x = x + 1;
alert (x);
</script>
```

Tra le altre cose, mettere il punto e virgola in fondo ad ogni istruzione ci consente di scrivere più istruzioni sulla stessa riga:

```
let y = 5; y = x + 1; alert(y);
```

Un altro aspetto sintattico da tenere in considerazione è il fatto che JavaScript è **case sensitive**, cioè fa distinzione tra maiuscole e minuscole nei nomi di istruzioni, variabili e costanti.

Commenti

singola riga //

righe multiple /* testo commentato */

Gli **spazi bianchi** in JavaScript non assumono significati particolari nelle espressioni e possiamo utilizzarli per aumentare la leggibilità del codice.

--- Argomento 3 Tipi di dato ---

JavaScript prevede **cinque tipi di dato primitivi**, *numeri*, *stringhe*, *booleani*, *null e undefined*, e un un tipo di dato complesso, gli *oggetti*.

Tutti gli altri elementi previsti dal linguaggio, come ad esempio gli array, le espressioni regolari, le funzioni, sono in realtà **oggetti**.

Anche i tipi di dato "primitivi" hanno degli oggetti corrispondenti con relative proprietà e metodi. JavaScript converte automaticamente un tipo primitivo nel corrispondente oggetto quando utilizziamo un suo metodo o una sua proprietà. Consideriamo ad esempio il seguente codice:

```
<script>
let nomeMaiuscolo = "Andrea".toUpperCase();
alert (nomeMaiuscolo);
</script>
```

A partire dal tipo primitivo stringa, JavaScript crea il corrispondente oggetto String e invoca il metodo toUppercase() per ottenere la versione in caratteri maiuscoli della stringa.

Stringa

Una stringa in JavaScript è una sequenza di caratteri delimitata da doppi o singoli apici.

Non c'è una regola per stabilire quale delimitatore utilizzare. L'unica regola è che il delimitatore finale deve essere uguale al delimitatore iniziale.

Per inserire caratteri speciali all'interno di una stringa si fa ricorso al carattere di escaping **** (backslash).
Ad esempio, per inserire un ritorno a capo possiamo utilizzare la sequenza \n, come mostrato nel seguente esempio:

```
<script>
let pioggia = "oggi è una bella giornata. \n Domani piove";
alert (pioggia);
</script>
```

Numeri interi

I seguenti sono esempi di rappresentazione di numeri interi:

```
let interoNegativo = -10;

let  zero = 0;

let interoPositivo = 123;
```

Floating point:

Per rappresentato un numero non intero facciamo ricorso al punto come separatore della parte decimale:

```
let numeroDecimale = 0.52;

let altroNumeroDecimale = 12.34;

let decimaleNegativo = -1.2;

let decimaleZero = 1.0;
```

È possibile rappresentare i valori numerici secondo la **notazione scientifica**:

```
let primoNumero = 12e3;
```

questa notazione comporta l'aggiunta di zeri (12000)

```
let secondoNumero = 3.5e-4;
```

se l'esponente è negativo dobbiamo spostare il numero (verso sinistra) di tante posizioni corrispondenti al valore dell'esponente. (0.00035)

Oltre alla classica notazione in base dieci, possiamo rappresentare i numeri in notazione ottale ed esadecimale e binaria:

```
let num1 = 011;

alert ("numero ottale: " +num1);
```

i numeri in notazione ottale devono iniziare con uno 0. L'ottale è un modo compatto per rappresentare numeri binari con tre bit alla volta.

Come va letto: la prima cifra (si parte sempre con la lettura da destra verso sinistra) vale quello che sta indicato cioè nel nostro caso 1; il secondo 1 corrisponde a 1*8^1 cioè 8: quindi 1+8 =9

```
let num2 = 0x11;

alert ("numero esadecimale: " + num2);
```

Quindi i numeri che iniziano con 0x rappresentano i numeri esadecimali.

Siccome i nostri numeri arabi terminano a 9 dobbiamo utilizzare delle lettere

(dalla A che vale 10 fino alla F che vale 15)

Come va letto: la prima cifra è 1; la seconda cifra è 1*16^1 cioè 16: 1 + 16 = 17

```
let num3 = 0b00000011 ;

alert ("numero binario: " + num3);
```

0b00000011 rappresenta un numero binario, quindi un numero che inizia con 0b rappresenta un numero binario.

Come va letto: la prima cifra è 1; la seconda cifra è 1*2^1 cioè 2: 1 + 2 = 3

Un altro valore numerico speciale è **NaN**, acronimo di Not a Number, che indica un valore numerico non definito.

Differenza sostanziale tra ***null*** e ***undefined***:
null: indica che alla variabile non è stato assegnato deliberatamente un valore.
undefined: indica, invece, una variabile non inizializzata, cioè a cui non è stato ancora assegnato un valore.

Tipizzazione debole

JavaScript è un linguaggio a tipizzazione debole o dinamica. Ciò significa che quando dichiariamo una variabile, non ne specifichiamo il tipo di dato e che il tipo di dato che può contenere una variabile può cambiare durante l'esecuzione dello script.

Quando dichiariamo una variabile senza specificare un valore, a questa viene assegnato il valore undefined. Il suo valore può cambiare tramite le nostre istruzioni ed assumere tipi di dato diversi, come nel seguente esempio:

```
let miaVariabile;

miaVariabile = 1;

miaVariabile = null;

miaVariabile = "uno";

miaVariabile = true;
```