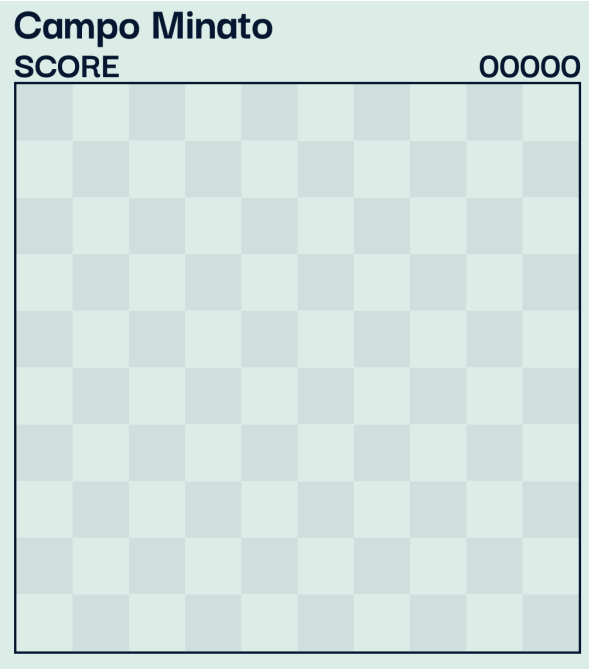

boolean

Campo Minato

"Click, click... boom!"

Cosa impariamo:

- JS Fundamentals
- Manipolare HTML e CSS con JS

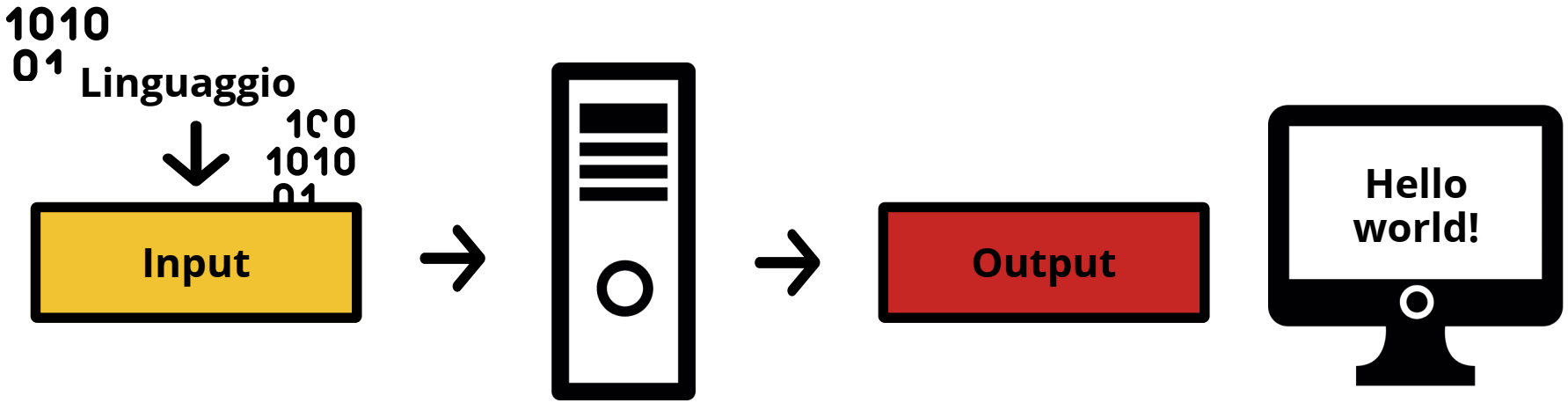


JavaScript

intro alla programmazione

Cos’è un linguaggio di programmazione?

Un **linguaggio di programmazione** consente di scrivere **istruzioni eseguibili** da un **computer** per svolgere una determinata operazione attraverso precise **regole grammaticali e sintattiche**.



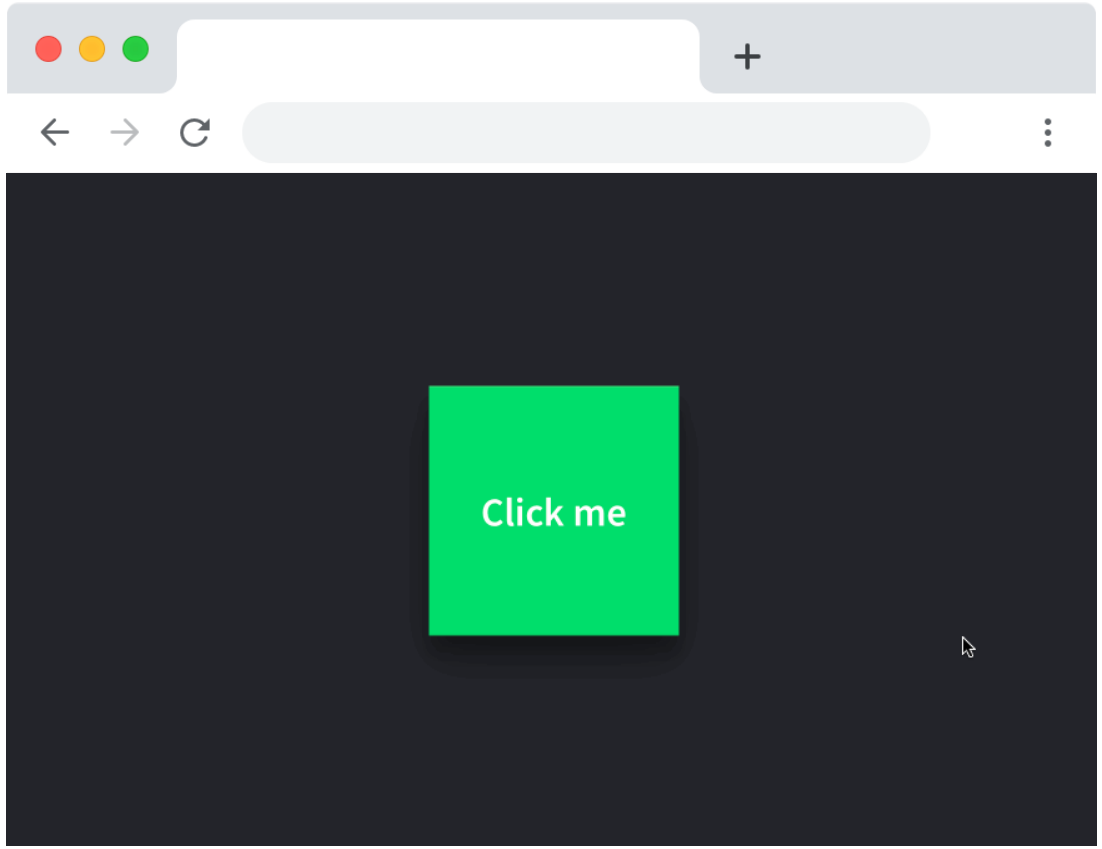
JavaScript

Per dare dinamicità alle pagine web

JavaScript consente di:

- accedere e modificare il contenuto di una pagina HTML
- reagire ad eventi generati dall'utente (ad es. il click del mouse)
- richiamare dati da fonti esterne (API)

... e molto altro!



JavaScript

JS

Hello JS!

Per aggiungere del codice JavaScript ad una pagina web è necessario creare **un file con estensione .js** e collegarlo al file html inserendo un tag **<script>** alla fine del <body>.

index.html

```html
<html>
<head>
</head>
<body>

  <script type="text/javascript" src="script.js">
  </script>

</body>
</html>
```

script.js

```javascript
// qui va il codice Javascript
console.log('Hello World!');
```

JavaScript

le variabili

Variabili

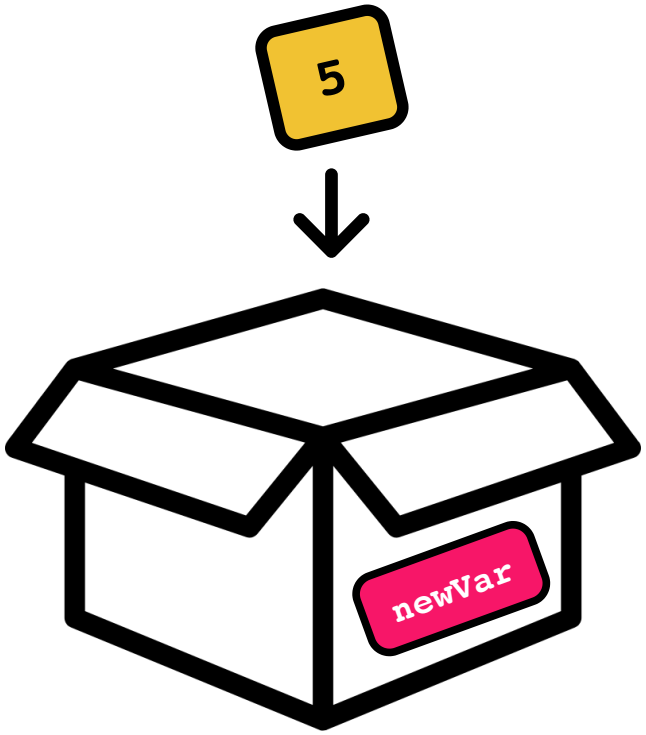
Come possiamo conservare dei dati?

Una variabile è uno spazio di memoria riservato per contenere un valore.

Quando vogliamo richiamare una variabile si fa riferimento al suo nome per accedere al valore memorizzato all'interno.

È come una scatola con una etichetta sopra!

```
1 const myVar = 5;
```



I Tipi Variabili

Il codice JavaScript manipola dei valori ed ogni valore appartiene ad un Tipo.

Alcune tipologie di dati:

Number: Numero

String: Stringa, sequenza di caratteri

Boolean: true|false

Array: struttura dati in grado di contenere più valori, una sorta di "archivio"

Objects: struttura dati complessa fatta di attributo e metodi con cui rappresentare elementi della realtà

Variabili

Dichiarazione

Le variabili in **JavaScript** sono dichiarate attraverso una specifica **parola chiave** seguita dal **nome della variabile**.

Queste parole chiave sono:

- ***const*** per dichiarare variabili che non possono essere modificate una volta inizializzate
- ***let*** per variabili riassegnabili

```
1 // dichiarazione variabile costante
2 const myVar1 = 5;
3
4 // dichiarazione variabile riassegnabile
5 let myVar2 = 5;
```

Variabili

Naming variabili

Il nome di una variabile e completamente arbitrario, ma ci sono alcune regole da seguire:

- il nome deve contenere solo lettere, numeri, simboli $ e _ (underscore)
- il primo carattere non può essere un numero
- le variabili in JavaScript sono case sensitive

```
1 const myVar1 = 'pippo';
2
3 let myVar2 = 2;
4
5
```

JavaScript

JS + HTML

JavaScript e l'HTML

Accedere agli elementi della pagina

Ci sono diversi metodi per ritrovare specifici elementi all'interno del documento HTML,
in particolare è possibile recuperare un elemento e salvare il suo riferimento in una variabile.

Alcuni metodi per accedere agli elementi:

- **document.querySelector():** recupera un singolo elemento sulla base di un selettore CSS
- **document.querySelectorAll():** recupera una lista di elementi sulla base di un selettore CSS

Naturalmente, possiamo conservare gli elementi HTML nelle variabili!

```
1 const title = document.querySelector('h1');
```

DOM Element Object

Cosa si può fare su un Element?

Più di 80 operazioni, vediamone alcune:

- element.innerText: permette di modificare il contenuto di un elemento
- element.innerHTML: permette di modificare il contenuto di un elemento
- element.classList.add: permette di aggiungere dinamicamente classi css
- element.classList.remove: permette di rimuovere dinamicamente classi css
- element.classList.contains: permette di sapere se una classe è presente in un elemento

JavaScript

Gli array

Array

Salviamo più dati assieme

L'array ci permette di salvare liste di valori, invece che un singolo valore!

Gli elementi di un array
si elencano tra parentesi quadre
e si separano con una virgola.

```
1 // Questo array contiene tre elementi di tipo stringa
2 const iscritti = ["Luca", "Marco", "Paolo"];
3
```

Nell'array possiamo conservare
ogni tipo di dato!

Array

Accesso ai dati

Come si accede agli elementi di un array?

nomeArray[IndiceElemento]

```
1 const iscritti = ["Luca", "Marco", "Paolo"];
2
3
4 iscritti[0] // Luca
5
6 iscritti[1] // Marco
7
8 iscritti[2] // Paolo
```

🧐 **Attenzione: gli array contano da 0!**

La prima **posizione/indice** quindi sarà 0 e non 1.

Array

E se voglio aggiungere un elemento?

Dopo la creazione di un array possiamo anche aggiungere degli altri dati.

Ad esempio qui gli iscritti iniziali sono 3, ma ad un certo punto potremmo aver bisogno di aggiungere un quarto elemento.

```
1 // creazione dell'array iniziale
2 const iscritti = ["Luca", "Marco", "Paolo"];
3
4 // aggiunta di un elemento all'array
5 iscritti.push('Michele');
6
```

```
// output dell'array modificato
["Luca", "Marco", "Paolo", "Michele"];
// .push() aggiunge un elemento alla fine
```

Array

Cos'altro posso fare con gli array?

Possiamo usare la proprietà **length** per conoscere quanti elementi ci sono al suo interno.

Possiamo usare il metodo **includes** per sapere se un elemento è presente nell'array.

E tanto altro...

```
1  // creazione dell'array iniziale
2  const iscritti = ["Luca", "Marco", "Paolo"];
3
4
5
6  // recuperiamo il numero di elementi
7  iscritti.length // 3
8
9
10
11 // Verificare se un valore è presente in un array
12 iscritti.includes("Marco"); // true
```

JavaScript

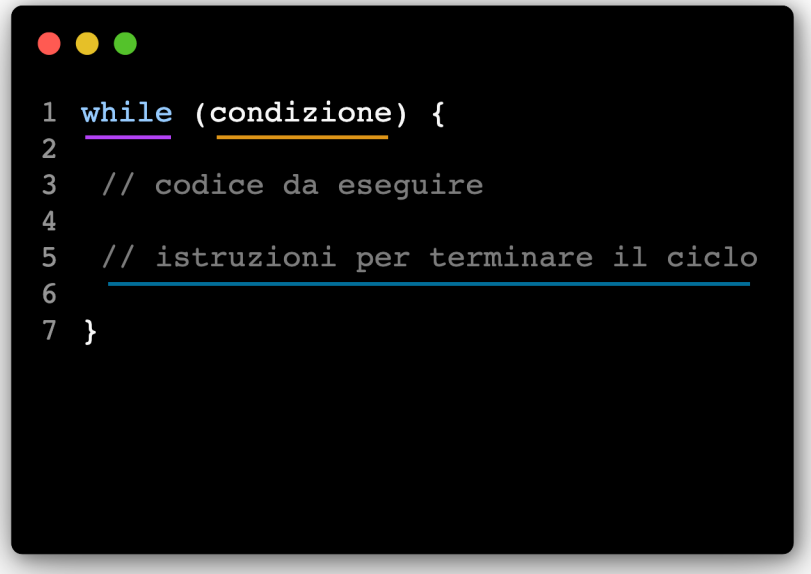
Cicli e condizioni

Ciclo while

Come si scrive un ciclo while?

Se non sappiamo quante volte deve ripetersi il nostro ciclo, possiamo utilizzare un **while**.

- **keyword:**
  while
- **condizione:**
  poi si imposta la condizione da verificare per poter eseguire il codice, che sarà ripetuto finché la condizione tra parentesi è vera.
- **Istruzioni per terminare il ciclo:**
  All'interno delle parentesi graffe dobbiamo fare in modo che prima o poi la condizione diventi falsa e il ciclo si interrompa!



Una **condizione** è la verifica che un valore o il confronto tra più valori corrisponda a **true** o **false**

Operatori relazionali

Come confrontiamo due valori

Vengono utilizzati per confrontare due valori e restituire un valore booleano che indica se il confronto è vero o falso.

`===`	*uguale a*	`3 === 3`	✅
`!==`	*diverso da*	`'a' !== 'b'`	✅
`>`	*maggiore di*	`3 > 2`	✅
`<`	*minore di*	`3 < 2`	❌

`>=` *maggiore o uguale*

`<=` *minore o uguale*

Operatori Logici

Uso avanzato delle condizioni

Gli operatori logici consentono di costruire **condizioni più complesse**.

&&	(AND) tutte le condizioni concatenate devono risultare VERE
||	(OR) almeno una delle condizioni concatenate deve risultare VERA
!	(NOT) inverte un valore booleano, utile ad esempio, verificare se un certo valore è falso

Istruzioni Condizionali

E se?

Se volessimo far “accadere una cosa” solo in un caso specifico?

Le **istruzioni condizionali** eseguono un certo blocco di codice **se si verifica una precisa condizione.**

La parte “altrimenti” non è obbligatoria.

```
1  if (condizione) {
2
3   // blocco di istruzioni 1
4
5  } else if (altra condizione ) {
6
7   // blocco di istruzioni 2
8
9  } else {
10
11  // blocco di istruzioni 3
12
13 }
14
```

Ciclo for

La sintassi - Come si scrive un ciclo for?

- **keyword:**
  for
- **contatore:**
  per prima cosa si imposta una variabile (in questo esempio *i* come *index*)
  e le si assegna un valore iniziale
  (qui *0*, ma potrebbe essere *1*, *5*, *20*...)
- **condizione:**
  poi si imposta la condizione da verificare per poter eseguire il codice;
  se la condizione è falsa si esce dal ciclo
  (in questo esempio la condizione è vera se il contatore è minore di 10)
- **incremento/decremento:**
  infine si incrementa/decrementa il valore del contatore che dovrà nuovamente essere testato nella condizione al giro successivo.

```
1 for (let i = 0; i < 10; i++) {
2
3   // Istruzioni da eseguire
4
5 }
```

JavaScript

Le funzioni, nel dettaglio

Le Funzioni

Le funzioni sono **blocchi di istruzioni** raggruppati insieme che possono essere eseguiti a piacimento.

Non vengono eseguiti non appena il browser arriva al loro rigo, ma in un secondo momento, deciso da noi.

Sono utilissime per vari motivi:

- Permettono di **organizzare il codice** in maniera più ordinata, **spezzettando le parti complesse**.

- Possiamo **riutilizzare più volte le istruzioni** al loro interno più volte nel nostro codice, senza riscriverlo.

- Il loro codice viene eseguito **quando decidiamo noi,** non nell'esatto momento in cui browser le vede.

Anonymous functions

In alcuni casi, come ad esempio in un EventListener, è possibile utilizzare una **funzione anonima** (*anonymous function*) cioè una funzione senza nome.

```
1 const btn = document.getElementById('button');
2
3 // Funzione anonima
4 btn.addEventListener('click', function() {
5   console.log('Hello');
6 });
```

```
1 const btn = document.getElementById('button');
2
3 // Funzione con nome
4  function sayHello() {
5    alert('Hello');
6  }
7
8 btn.addEventListener('click', sayHello);
```

Named Functions

Come si scrive una funzione completa?

- **keyword:**
  function
- **Nome funzione:**
  si imposta il nome della funzione
  con la quale potremo poi richiamarla.
- **Codice da eseguire:**
  Tra le parentesi graffe inseriamo
  il codice che vogliamo eseguire.
- **Parametri:** dati in ingresso che possono essere elaborati per restituire un risultato.
- **Valore restituito:** il risultato di una operazione.

```
1  function miaFunzione(num1, num2) {
2
3    // blocco di codice
4    const risultato = num1 + num2;
5
6    return risultato;
7
8    // eventuali altre istruzioni dopo il return
9    // non verranno eseguite
10
11 }
```

🧐 **Non è obbligatorio avere dei parametri, né tantomeno restituire un risultato**.

Le Funzioni

Invocare una funzione

Invocare una funzione consiste nel:
scriverne il nome seguito dalle parentesi tonde
nel punto di codice in cui vogliamo usarla.

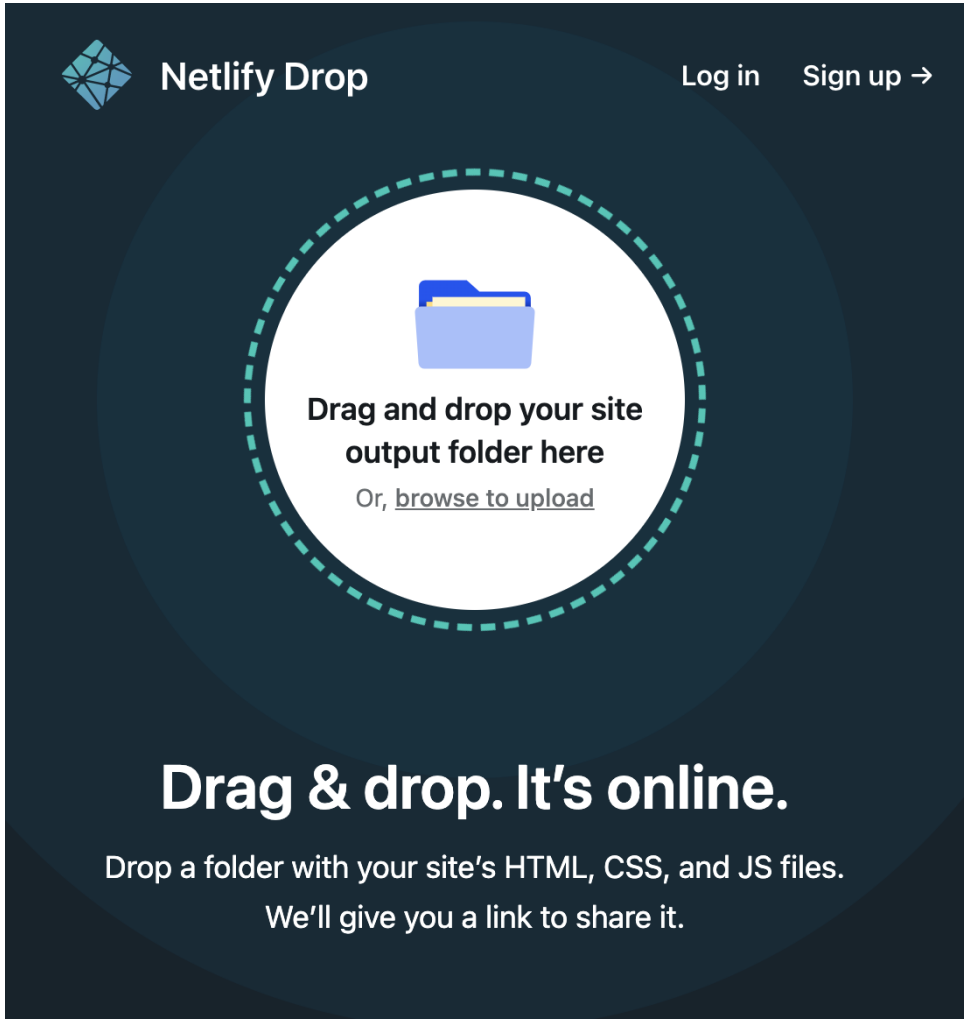
Una volta invocata, la funzione **eseguirà il codice** in essa contenuto.

```
1 nomeFunzione();  //Senza argomento
2
3 nomeFunzione(argomento); //Con argomento
```

Deploy

Mettiamo il nostro codice su un server

https://app.netlify.com/drop



CHALLENGE

Caccia al tesoro

"argh! non avrai mai il mio tesoro..."

Aggiungi un **tesoro** tra le bombe
e chi lo trova vince il gioco!

Cosa può essere il premio? Scegli tu:

- una gif/meme
- un link a youtube/spotify
- una citazione

o se vuoi tutto in modo random!