
boolean

To-Do List

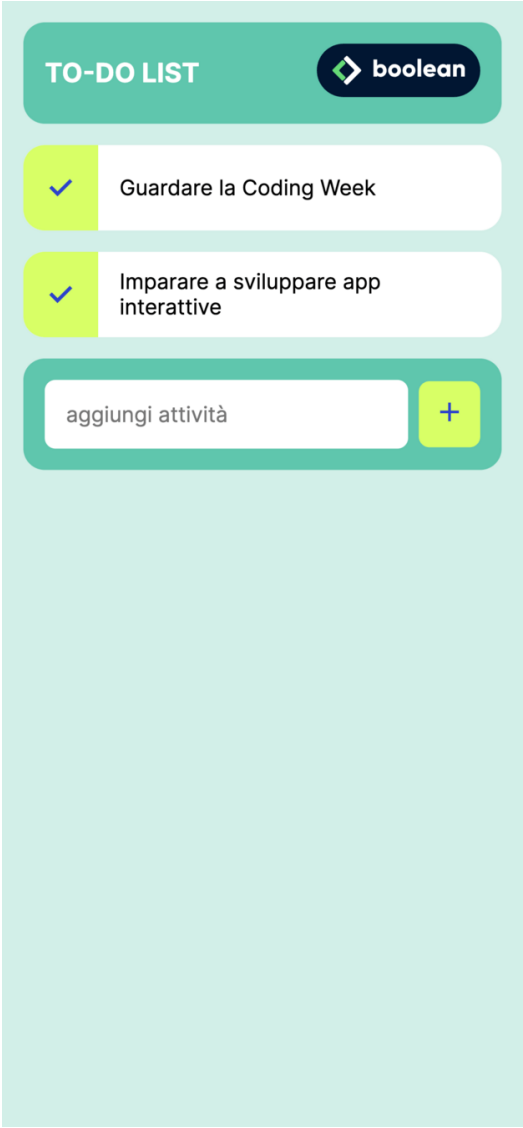
"✅ Imparara a programmare"

To-Do List

Impariamo i concetti base della programmazione e applichiamoli in JavaScript per sviluppare la logica della nostra app.

Contenuti:

- intro alla programmazione
- variabili e dati
- le funzioni
- strutture condizionali
- manipolare html
- eventi

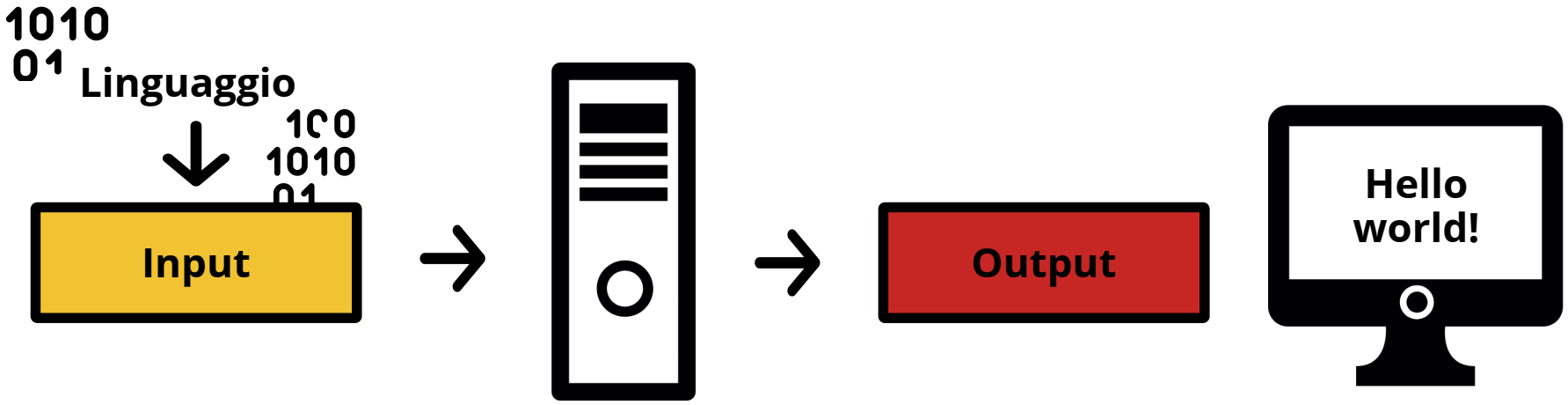


JavaScript

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**.



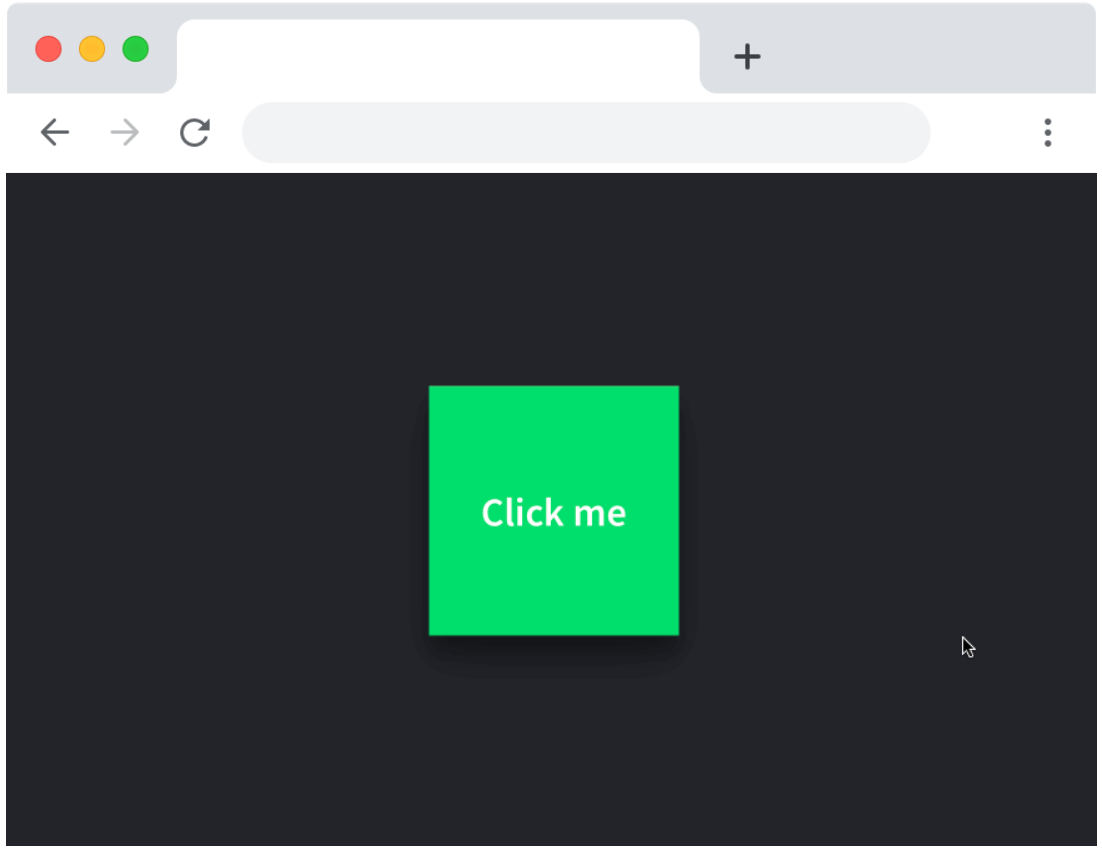
JavaScript

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

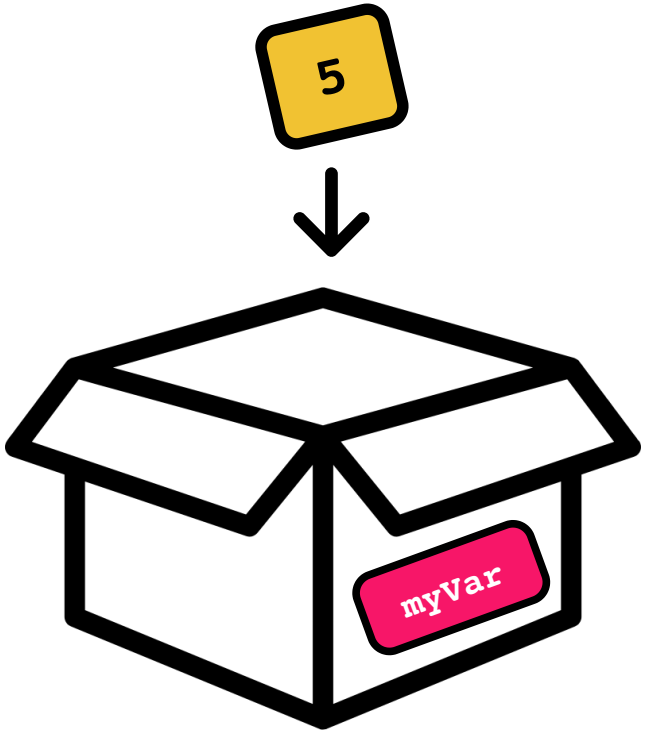
Variabili

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 // dichiarazione di una variabile
2 const myVar = 5;
3
4 // usi successivi (senza 'const' davanti)
5 console.log(myVar)
```

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;
```

Naming delle 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.

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 e lunghezza

Come si accede agli elementi di un array?

nomeArray[IndiceElemento]

Come so quanti elementi ci sono?

nomeArray.length

Attenzione: gli array contano da 0!

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

```
1  const iscritti = ["Luca", "Marco", "Paolo"];
2
3
4  iscritti[0] // Luca
5
6  iscritti[1] // Marco
7
8  iscritti[2] // Paolo
9
10
11 iscritti.length // 3
```

Array

E se voglio aggiungere/rimuovere un elemento?

Dopo la creazione di un array possiamo modificare il suo contenuto:

.push() *per aggiungere*

.splice() *per rimuovere*

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

```
1 // creazione dell'array iniziale
2 const iscritti = ["Luca", "Marco", "Paolo"];
3
4 // rimozione di un elemento dell'array
5 iscritti.splice(0, 1);
6
7  // output dell'array modificato
8  ["Marco", "Paolo"];
```

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.

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

Dichiarare e invocare una funzione

Dichiarare una funzione consiste nel definire il suo nome, ciò di cui ha bisogno e cosa deve fare.

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 // INVOCAZIONE
2 //
3 nomeFunzione();  //Senza argomento
4
5 nomeFunzione('marco'); //Con argomento
```

JavaScript

Istruzioni condizionali

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.

Se si verifica la (condizione) {

// allora viene eseguito questo blocco di istruzioni 1 (notare le graffe)

} **altrimenti** {

// viene eseguito questo blocco di istruzioni 2 (notare le graffe)

}

```
1 if (condizione) {
2
3  // blocco di istruzioni 1
4
5 } else {
6
7  // blocco di istruzioni 3
8
9 }
```

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*	

JavaScript

Array e funzioni

Array e funzioni

Un metodo per iterare sugli elementi

forEach ci aiuta a iterare sugli array
in maniera concisa e automatica

- **keyword**
  forEach
- **function**
  All'interno delle parentesi tonde inseriamo una funzione che riceverà come argomenti:
  element - l'elemento dell'array sul quale stiamo girando
  index - l'indice di quell'elemento
- **codice:**
  che verrà eseguito ad ogni giro

```
1  const students = ['Paolo', 'Giulia', 'Marco'];
2
3  students.forEach(function(element, index) {
4   console.log(element, index);
5  });
6
7
8  // Paolo 0
9  // Giulia 1
10 // Marco 2
```

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

JavaScript

Template Literals

La via tradizionale

Concatenazione con operatore "+"

Spesso si presenta l'esigenza di mescolare dei pezzi di testo (stringhe) statici ad alcuni pezzi dinamici.

L'operatore "+" permette di farlo ma con qualche seccatura, tra cui la gestione degli spazi vuoti:

```
1 const firstName = 'Marco';
2 const lastName = 'Lancellotti';
3 const job = 'Web Developer'
4
5 const presentazione = 'Ciao, mi chiamo ' + firstName + ' ' + lastName + ' e lavoro come ' + job;
6
7 // RISULTATO: "Ciao mi chiamo Marco Lancellotti e lavoro come Web Developer";
```

Template literals

Stringhe e variabili

Le specifiche ES6 ci permettono di mescolare testo fisso e variabili con una sintassi molto comoda.

Basterà scrivere la nostra frase tra backticks (`) e racchiudere le variabili in ${}!

```
1 const firstName = 'Marco';
2 const lastName = 'Lancellotti';
3 const job = 'Web Developer'
4
5 const presentazione = `Ciao, mi chiamo ${firstName} ${lastName} e lavoro come ${job}`;
6
7 // RISULTATO : "Ciao mi chiamo Marco Lancellotti e lavoro come Web Developer";
```

Template Literals

HTML e variabili!

Inoltre, ci permettono di inserire anche dei veri e propri "template" di HTML

```
1  const firstName = 'Marco';
2  const lastName = 'Lancellotti';
3  const job = 'Web Developer';
4
5
6  const myHTML = `
7      <div class="miaClasse">
8          <h1>Titolo</h1>
9          <p>`Ciao, mi chiamo ${firstName} ${lastName} e lavoro come ${job}`</p>
10     </div>
11 `;
```

Dove trovo il backtick nella tastiera? 🤔

Su Mac:

- digitare **Option** (⌥) **+ ** (backspace)

Su Windows:

- digitare **Alt + 96** (dal tastierino numerico)

JavaScript

Gli eventi

Gli Eventi

Ci sono molti eventi che possiamo intercettare

Tra i più comuni, ci sono le interazioni con la pagina da parte dell'utente:

- click
- submit
- scroll
- resize
- ...

Gli Eventi

Nella pagina ci sono molti eventi che possiamo intercettare.

Tra i più comuni, ci sono le interazioni con la pagina da
parte dell'utente: click, submit, scroll, resize ...

Rimaniamo in **ascolto di un evento** su quell'elemento.
Quale evento? '**click**'

In questo esempio vediamo una **funzione** che viene attivata solo quando avviene l'evento '**click**'

```html
<div class="saluto">CIAO</div>
```

```javascript
const element = document.querySelector(".saluto");

element.addEventListener('click', function() {
   // codice
 });
```

Local Storage

Local Storage

Web Storage API

Con **HTML5** sono state introdotte nuove funzionalità a disposizione degli sviluppatori per la creazione delle web apps.

L'oggetto **localStorage** consente di salvare localmente alcune informazioni che NON vengono cancellate alla chiusura della finestra, ma persistono nel browser senza una scadenza.

Local Storage

Lettura / Scrittura

Le informazioni vengono conservate nella forma **chiave : valore** in cui sia chiave che valore possono essere esclusivamente delle *stringhe*.

I metodi dell'oggetto Local Storage:

- getItem(*chiave*) per recuperare un valore in base alla sua chiave

- setItem(*chiave, valore*) per salvare un valore in base a una chiave

- clear() per svuotare e ripulire tutto il localStorage

```
1 localStorage.setItem('nome', 'Marco');
2 localStorage.setItem('ruolo', 'Developer');
3
4 const nome = localStorage.getItem('nome');
5 const ruolo = localStorage.getItem('ruolo');
6
7 console.log(nome); // Marco
8 console.log(ruolo); // Developer
```

🤔 Se volessi salvare un array?

```
1 const team = ["Filippo", "Marco"];
2 localStorage.setItem('person', JSON.stringify(team));
3
4 const person = localStorage.getItem('person');
5 console.log(JSON.parse(person));
6 // ["Filippo", "Marco"]
```

Challenge

Check that task!

🎨 Lista tematica

Scegli un tema specifico e personalizza l'aspetto. Ad es. app per tracciare libri da leggere, elementi della tua collezione mancanti, argomenti da studiare etc.

🗑️ Reset della lista

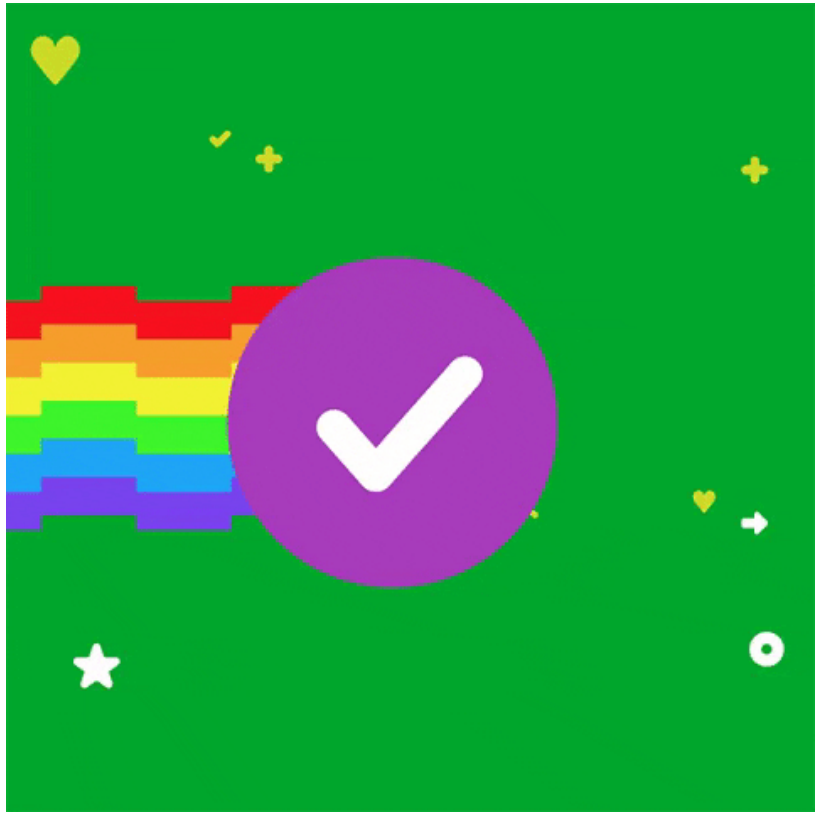
Fai pulizia e cancella tutte le task dalla lista con un solo tasto. **Ricorda di cancellare il LocalStorage!!**

https://www.w3schools.com/jsref/met_storage_clear.asp

☐ Prompt di conferma

Chiediamo all'utente conferma prima di segnare la todo come completata utilizzando la funzione **window.confirm()** del browser

https://developer.mozilla.org/en-US/docs/Web/API/Window/confirm?retiredLocale=it



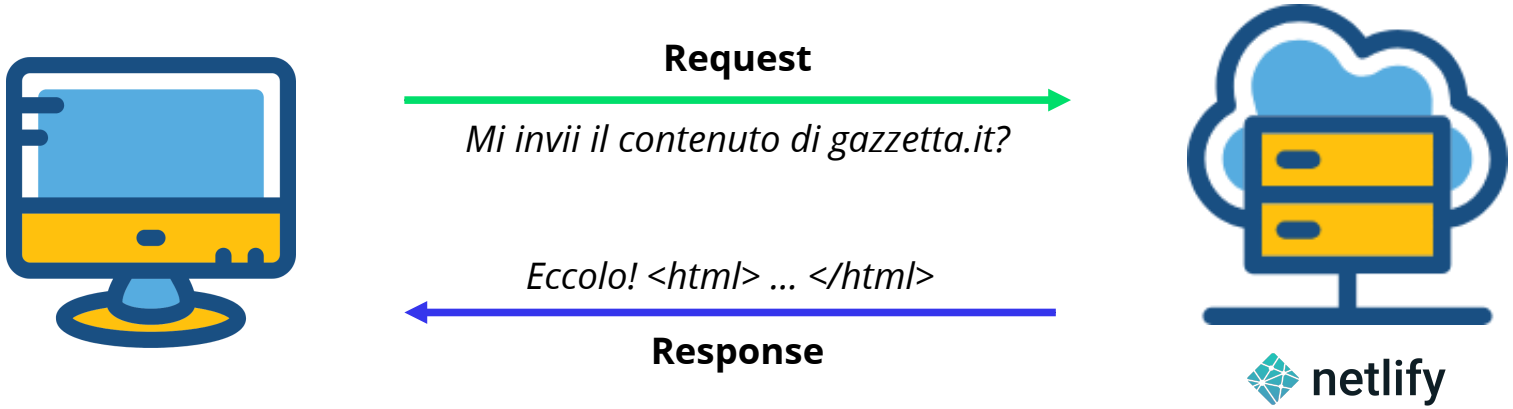
Dev Hints

Deploy

Mettiamo il nostro codice su un server.

Sfruttiamo un servizio gratuito che ci consente di trasferire agevolmente il nostro codice e generare un link pubblico per condividere la nostra web app.

https://app.netlify.com/drop/



Deploy

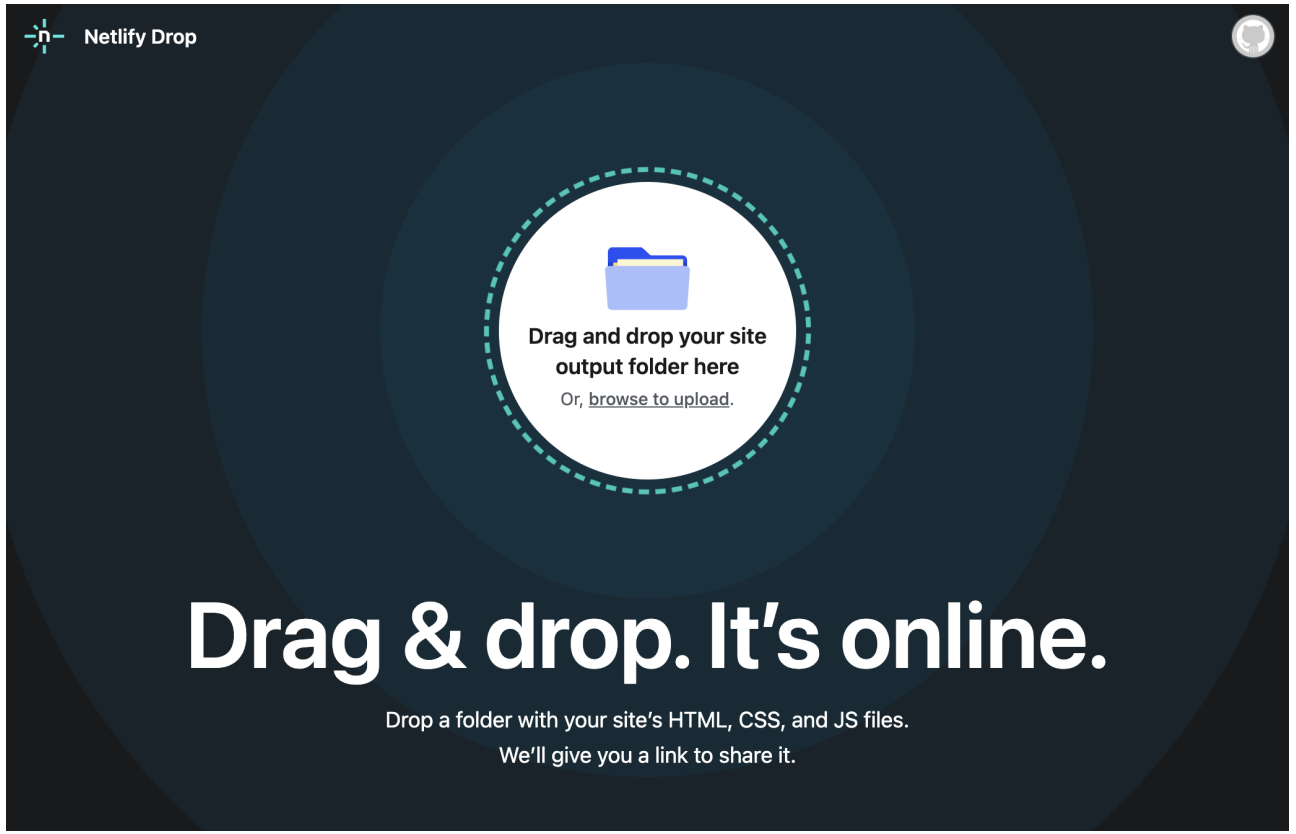
1. Registrati Netlify, in questo modo il tuo progetto sarà pubblicato per un tempo illimitato e non richederà una password

2. Trascina la cartella del tuo progetto dentro Netlify drop

3. Ottieni il link generato e condivisibile

E' fatta!

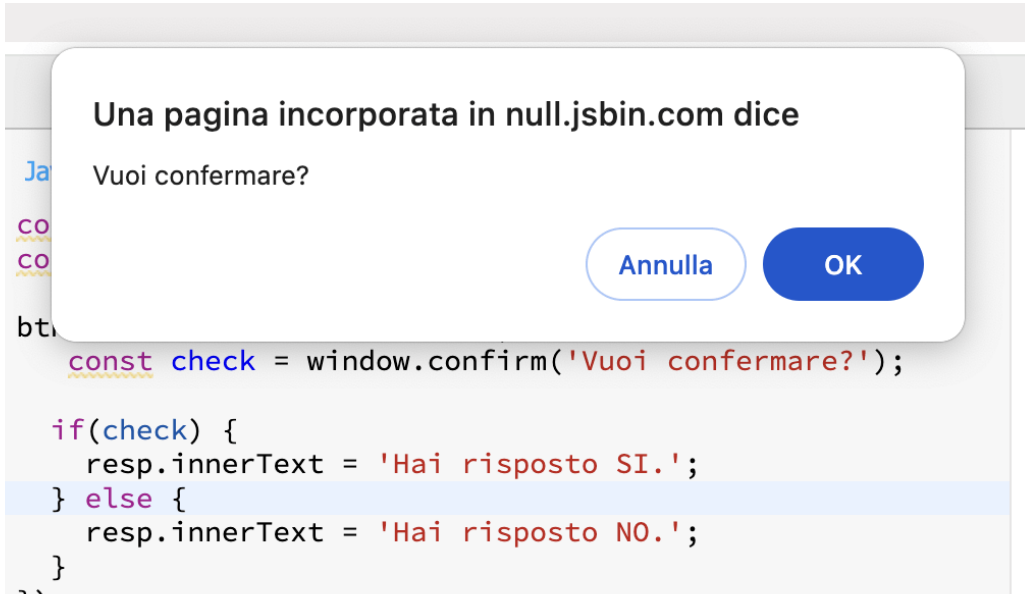
https://app.netlify.com/drop/



window.confirm()

Are you sure?

E' una funzione nativa del browser per chiedere una conferma a un utente. Forse non ottimale come user experience, ma può servire al nostro scopo!



Fai una prova: *https://jsbin.com/pikodigudi/edit?html,js,output*

```html
<button>OPEN</button>
<p class="response"></p>
```

```js
const btn = document.querySelector('button');
const resp = document.querySelector('.response');

btn.addEventListener('click', function() {
   const check = window.confirm('Vuoi confermare?');

  if(check) {
    resp.innerText = 'Hai risposto SI.';
  } else {
    resp.innerText = 'Hai risposto NO.';
  }
});
```