Introduzione a JavaScript

JavaScript è un linguaggio di programmazione dinamico e interpretato, originariamente sviluppato per aggiungere interattività alle pagine web. È il linguaggio di scripting più popolare per il web, ed è supportato da tutti i principali browser (come Chrome, Firefox, Safari, e Edge). Negli anni, JavaScript si è evoluto notevolmente e oggi è alla base di molte applicazioni moderne, sia lato client (frontend) che lato server (backend, grazie a tecnologie come Node.js).

Caratteristiche Principali di JavaScript

1. **Linguaggio Interpretato**: A differenza di altri linguaggi che richiedono la compilazione (come Java o C++), il codice JavaScript viene eseguito direttamente dal browser, linea per linea, grazie a un interprete integrato nel browser stesso. Questo permette un rapido ciclo di sviluppo e test.
2. **Linguaggio Lato Client e Lato Server**:
    - Lato **client**: JavaScript è principalmente utilizzato per manipolare gli elementi delle pagine web, migliorare l'esperienza utente e rendere le pagine interattive.
    - Lato **server**: Con Node.js, JavaScript può essere eseguito sul server, permettendo di sviluppare applicazioni complete (full-stack) solo con JavaScript.
3. **Multi-paradigma**: JavaScript supporta vari paradigmi di programmazione, tra cui:
    - **Programmazione procedurale**: Scrittura di codice in sequenza con istruzioni.
    - **Programmazione funzionale**: Uso di funzioni come "cittadini di prima classe" (funzioni che possono essere passate come parametri o restituite da altre funzioni).
    - **Programmazione orientata agli oggetti**: Con l'introduzione di classi e prototipi, JavaScript permette di creare oggetti e gestire il loro stato e comportamento.
4. **Dinamicamente Tipizzato**: Non è necessario dichiarare il tipo di variabile in JavaScript. Una variabile può contenere qualsiasi tipo di dato e cambiare tipo durante l'esecuzione del programma.

Storia di JavaScript

JavaScript è stato sviluppato nel 1995 da **Brendan Eich** in soli 10 giorni, inizialmente con il nome **Mocha**, poi **LiveScript**, e infine **JavaScript**. Anche se il nome sembra simile, non ha alcuna relazione diretta con il linguaggio Java; fu chiamato così per motivi di marketing. JavaScript fu standardizzato da **ECMA International** e divenne noto ufficialmente come **ECMAScript** (con la prima versione rilasciata come ECMAScript 1 nel 1997).

Differenze tra JavaScript, HTML e CSS

In una pagina web, **HTML**, **CSS**, e **JavaScript** svolgono ruoli distinti ma complementari:

- **HTML (HyperText Markup Language)**: Fornisce la struttura di base di una pagina web (definendo titoli, paragrafi, link, immagini, ecc.).
- **CSS (Cascading Style Sheets)**: Gestisce l'aspetto visivo della pagina (colore, layout, font, ecc.).
- **JavaScript**: Aggiunge interattività e dinamicità alla pagina (ad esempio, animazioni, risposte a eventi come clic e movimenti del mouse, aggiornamento dei contenuti senza ricaricare la pagina, ecc.).

Come Includere JavaScript in una Pagina HTML

Esistono vari modi per includere JavaScript in una pagina HTML:

1. **Script Inline**: Si può scrivere JavaScript direttamente all'interno del tag `<script>` all'interno dell'HTML.

```
<!DOCTYPE html>
<html lang="it">
<head>
    <title>Script Inline</title>
</head>
<body>
    <h1>Benvenuti in JavaScript</h1>
    <script>
        alert("Ciao dal JavaScript inline!");
    </script>
</body>
</html>
```

2. **Script in un File Esterno**: Consigliato per organizzare meglio il codice, specialmente per progetti più grandi. Si crea un file `.js` separato e lo si collega all'HTML.
    - **HTML**:

```
<!DOCTYPE html>
<html lang="it">
<head>
    <title>Script Esterno</title>
    <script src="script.js"></script> <!-- collegamento al file JS -
->
</head>
<body>
    <h1>Esempio con Script Esterno</h1>
</body>
</html>
```

    - **JavaScript (script.js)**:

```
console.log("Esecuzione di JavaScript da un file esterno!");
```

3. **Posizionamento dello Script nel `<head>` o Prima della Fine del `<body>`**:
    - Inserire lo script alla fine del `<body>` (proprio prima del tag di chiusura `</body>`) è una pratica comune per garantire che l'HTML sia caricato prima che il JavaScript venga eseguito.
    - In alternativa, si può usare l'attributo `defer` nel `<script>` se si desidera posizionarlo nel `<head>`. `defer` indica al browser di caricare il JavaScript senza bloccare il caricamento della pagina e di eseguirlo solo dopo che l'intero HTML è stato caricato.

```
<!DOCTYPE html>
<html lang="it">
<head>
    <title>Script con Defer</title>
    <script src="script.js" defer></script> <!-- lo script verrà eseguito
dopo il caricamento della pagina -->
</head>
<body>
    <h1>Pagina con Script Defer</h1>
</body>
</html>
```

Utilizzo di JavaScript per Interagire con l'Utente

JavaScript è progettato per rispondere alle azioni dell'utente (detti **eventi**), come il clic di un pulsante, il passaggio del mouse su un elemento o la pressione di un tasto sulla tastiera.

Esempio: Gestione di Eventi con JavaScript

Supponiamo di voler cambiare il testo di un paragrafo quando l'utente fa clic su un pulsante:

```
<!DOCTYPE html>
<html lang="it">
<head>
    <title>Esempio di Evento</title>
</head>
<body>
    <h2>Clicca il pulsante per cambiare il testo</h2>
    <p id="testo">Questo è il testo originale.</p>
    <button onclick="cambiaTesto()">Clicca qui</button>

    <script>
        function cambiaTesto() {
            document.getElementById("testo").textContent = "Hai cambiato il
testo!";
        }
    </script>
</body>
</html>
```

In questo esempio, quando l'utente fa clic sul pulsante, la funzione `cambiaTesto()` viene eseguita, cambiando il contenuto del paragrafo.

Esempi di Applicazioni in JavaScript

JavaScript permette di realizzare una vasta gamma di applicazioni, dalle animazioni alle interfacce utente avanzate. Ecco alcuni esempi di cosa può fare JavaScript:

1. **Validazione dei Dati**: JavaScript può verificare la correttezza dei dati inseriti dall'utente prima che siano inviati al server, ad esempio per assicurarsi che tutti i campi obbligatori siano compilati.
2. **Effetti Visivi e Animazioni**: Librerie come **jQuery** o **CSS combinato con JavaScript** permettono di creare animazioni, come effetti di scorrimento o cambi di colore e forma dinamici.
3. **Interazione Asincrona (AJAX)**: Con AJAX, JavaScript può recuperare dati dal server senza ricaricare l'intera pagina, rendendo le applicazioni più rapide e interattive (ad esempio, come quando aggiorni un feed sui social media).
4. **Single Page Applications (SPA)**: JavaScript consente di creare SPA, applicazioni che simulano un'applicazione desktop interamente su una singola pagina web, con framework come **React**, **Vue**, e **Angular**.

JavaScript come Parte dello Stack Tecnologico

Grazie alla sua flessibilità e alla compatibilità universale, JavaScript è una parte centrale del cosiddetto **stack MEAN** o **stack MERN** per applicazioni web:

- **MongoDB** (database)
- **Express** (framework per server)
- **Angular** o **React** (framework/libreria frontend)
- **Node.js** (JavaScript per il server)

Con questi strumenti, JavaScript può essere utilizzato per costruire applicazioni moderne full-stack, permettendo a uno sviluppatore di lavorare sia lato server che lato client.

Variabili e Tipi di Dato

Dichiarazione di Variabili

In JavaScript, le variabili possono essere dichiarate con tre parole chiave principali: `let`, `const`, e `var`. Ognuna ha delle caratteristiche specifiche, soprattutto riguardo allo **scope** (ambito di visibilità) e alla **mutabilità** della variabile.

- **`let`**: Utilizzato per dichiarare variabili il cui valore può cambiare. Ha uno scope limitato al **blocco** in cui è dichiarato (scope di blocco).

  ```
  let nome = "Mario";
  nome = "Luigi"; // è possibile riassegnare la variabile
  ```

- **`const`**: Utilizzato per dichiarare variabili il cui valore non dovrebbe cambiare. Come `let`, ha scope di blocco. Una volta assegnato un valore, non può essere riassegnato.

  ```
  const PI = 3.1415;
  // PI = 3; // ERRORE! Non si può riassegnare una costante
  ```

- **`var`**: Questa è la dichiarazione originale di JavaScript. Ha scope **funzionale**, quindi è visibile in tutta la funzione in cui è dichiarata e tende a generare confusione e bug in situazioni più complesse. Per questo motivo, il suo uso è **sconsigliato** rispetto a `let` e `const`.

  ```
  var eta = 25;
  ```

Tipi di Dato in JavaScript

JavaScript supporta diversi tipi di dati fondamentali:

1. **Stringhe** (`String`): Serie di caratteri racchiusi tra virgolette semplici (`'`), doppie (`"`), o backtick (``) per template literal.

   ```
   let saluto = "Ciao!";
   let messaggio = `Il mio nome è ${nome}`; // Template literal, permette interpolazione
   ```

2. **Numeri** (`Number`): Rappresentano sia interi che decimali.

   ```
   let intero = 10;
   let decimale = 3.14;
   ```

3. **Booleani** (`Boolean`): Valori `true` o `false`.

```
let attivo = true;
let maggioreEta = false;
```

4. **Array**: Una collezione ordinata di elementi, racchiusa in parentesi quadre `[]`. Gli array possono contenere qualsiasi tipo di dato e persino altri array.

```
let numeri = [1, 2, 3, 4];
let misto = ["Mario", 30, true];
```

5. **Oggetti** (`Object`): Collezione di coppie chiave-valore. Gli oggetti permettono di modellare entità complesse.

```
let persona = {
    nome: "Mario",
    eta: 30,
    attivo: true
};
```

6. **Null e Undefined**:
    - **`null`** rappresenta intenzionalmente un valore vuoto o inesistente.
    - **`undefined`** rappresenta una variabile dichiarata ma a cui non è stato assegnato un valore.
7. **Symbol**: Tipo speciale introdotto in ES6, utilizzato principalmente per creare identificatori unici.

Operatori (Aritmetici, Logici, di Confronto)

JavaScript dispone di numerosi operatori per lavorare con variabili e valori. Questi includono operatori aritmetici, logici, e di confronto.

Operatori Aritmetici

Utilizzati per eseguire calcoli matematici.

- `+` Somma
- `-` Sottrazione
- `*` Moltiplicazione
- `/` Divisione
- `%` Modulo (resto della divisione)
- `++` Incremento
- `--` Decremento

Esempi:

```
let a = 10;
let b = 5;
let somma = a + b;      // 15
let prodotto = a * b;   // 50
let resto = a % b;      // 0
a++;                    // Incrementa a di 1, ora a è 11
```

Operatori di Confronto

Utilizzati per confrontare valori e restituire un valore booleano (`true` o `false`).

- `==` Uguaglianza (converte i tipi di dato se necessario)
- `===` Uguaglianza stretta (non converte i tipi di dato)
- `!=` Diversità
- `!==` Diversità stretta
- `>` Maggiore
- `<` Minore
- `>=` Maggiore o uguale
- `<=` Minore o uguale

Esempi:

```
let x = 10;
let y = 5;
let confronto = x > y;       // true
let uguale = x === 10;       // true
let diverso = x !== "10";    // true (il tipo è diverso)
```

Operatori Logici

Utilizzati per combinare o invertire espressioni logiche.

- `&&` AND logico (vero se entrambi gli operandi sono veri)
- `||` OR logico (vero se almeno uno degli operandi è vero)
- `!` NOT logico (inverte il valore booleano)

Esempi:

```
let eta = 20;
let patente = true;
let maggiorenneConPatente = (eta >= 18) && patente; // true
let minoreOHaPatente = (eta < 18) || patente;       // true
let senzaPatente = !patente;                        // false
```

Strutture di Controllo

Le strutture di controllo permettono di decidere quali blocchi di codice eseguire a seconda di condizioni specifiche.

Condizione `if` e `else`

Permette di eseguire codice condizionalmente. `else` gestisce il caso in cui la condizione di `if` non sia soddisfatta.

```
let eta = 20;

if (eta >= 18) {
    console.log("Sei maggiorenne");
} else {
    console.log("Sei minorenne");
}
```

Condizioni Multiple: `else if`

Quando sono necessarie più condizioni, si può aggiungere un blocco `else if`.

```
let ora = 14;

if (ora < 12) {
    console.log("Buongiorno");
} else if (ora < 18) {
    console.log("Buon pomeriggio");
} else {
    console.log("Buonasera");
}
```

Cicli: `for`, `while`

I cicli consentono di eseguire ripetutamente un blocco di codice.

Esempio: Ciclo `for`

```
for (let i = 0; i < 5; i++) {
    console.log("Iterazione numero", i);
}
```

Esempio: Ciclo `while`

```
let i = 0;
while (i < 5) {
    console.log("Iterazione numero", i);
    i++;
}
```

Funzioni e Scope

Le funzioni sono blocchi di codice riutilizzabili che eseguono un compito specifico. In JavaScript, le funzioni possono essere assegnate a variabili, passate come argomenti, e ritornate da altre funzioni.

Dichiarazione di Funzioni

Una funzione si dichiara con la parola chiave `function`, seguita dal nome, dalle parentesi tonde `()` (per eventuali parametri) e dalle parentesi graffe `{}` per definire il corpo della funzione.

```
function saluta(nome) {
    return "Ciao " + nome;
}
console.log(saluta("Luca")); // Output: "Ciao Luca"
```

Funzioni Freccia (Arrow Functions)

Le **arrow functions** sono una sintassi alternativa e più compatta introdotta con ES6, utile per le funzioni semplici e anonime.

```
const saluta = (nome) => "Ciao " + nome;
```

```
console.log(saluta("Luca")); // Output: "Ciao Luca"
```

Scope (Ambito delle Variabili)

Lo **scope** determina dove una variabile è accessibile nel codice:

- **Scope Globale**: Variabili dichiarate fuori da una funzione o da un blocco `{}` sono globali e accessibili ovunque nel codice.
- **Scope di Funzione**: Variabili dichiarate dentro una funzione sono accessibili solo all'interno di quella funzione.
- **Scope di Blocco**: Variabili dichiarate con `let` o `const` all'interno di un blocco `{}` sono visibili solo all'interno di quel blocco.

Esempio di Scope:

```
let messaggio = "Ciao"; // Variabile globale

function stampaSaluto() {
    let messaggioInterno = "Salve"; // Scope della funzione
    console.log(messaggio);          // Accede a "messaggio" globale
    console.log(messaggioInterno);    // Accede a "messaggioInterno"
}

stampaSaluto();
// console.log(messaggioInterno); // ERRORE! Fuori dallo scope di funzione
```

JavaScript ha una sintassi versatile, semplice da comprendere ma al tempo stesso robusta e avanzata per supportare programmi complessi.

Sintassi e delle istruzioni JavaScript.

1. Case Sensitivity (Sensibilità a Maiuscole e Minuscole)

JavaScript distingue tra maiuscole e minuscole. Questo significa che `Nome` e `nome` sono considerati due variabili diverse.

```
let Nome = "Mario";
let nome = "Luigi";
console.log(Nome); // Mario
console.log(nome); // Luigi
```

2. Parole Chiave e Identificatori

JavaScript ha parole chiave riservate (come `let`, `const`, `if`, `return`, ecc.) che non possono essere usate come nomi di variabili o funzioni. Gli identificatori (nomi di variabili, funzioni, ecc.) devono iniziare con una lettera, un simbolo `$`, o `_`, e possono contenere solo lettere, cifre e simboli `_` o `$`.

```
let $conto = 100;
let _nome = "Mario";
let valore2 = 30;
```

3. Punto e Virgola

In JavaScript, l'uso del punto e virgola `;` per terminare una linea è opzionale, ma consigliato per evitare errori di interpretazione. JavaScript utilizza una tecnica chiamata **Automatic Semicolon Insertion (ASI)** per aggiungere i `;` quando omessi, ma questo non sempre funziona come previsto, quindi è meglio inserirli manualmente.

```
let nome = "Mario";
console.log(nome);
```

4. Tipi di Commento

I commenti sono fondamentali per rendere il codice leggibile e documentato. JavaScript supporta commenti su una sola linea e su più linee.

- **Commento su una linea**: inizia con `//`
- **Commento su più linee**: delimitato da `/* */`

```
// Questo è un commento su una sola linea

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

5. Variabili e Costanti

In JavaScript, le variabili possono essere dichiarate con `let`, `const`, o `var` (anche se quest'ultimo è ormai obsoleto e sconsigliato).

- **`let`**: Dichiarazione di variabili il cui valore può cambiare.
- **`const`**: Dichiarazione di variabili costanti, il cui valore non cambia.
- **`var`**: Variabile con visibilità di funzione, obsoleta e meno sicura.

```
let eta = 25;          // Variabile modificabile
const pi = 3.14159;    // Costante non modificabile
// pi = 3.14;           // ERRORE! Non si può riassegnare una costante
```

6. Tipi di Dato e Literali

JavaScript riconosce vari tipi di dato:

- **String**: Testo racchiuso in apici singoli o doppi, o con backtick per template literals.
- **Number**: Numeri interi o decimali.
- **Boolean**: Valori `true` o `false`.
- **Array**: Elenco ordinato di valori racchiusi in `[]`.
- **Object**: Collezione di coppie chiave-valore.
- **Null**: Valore intenzionalmente vuoto.
- **Undefined**: Variabile dichiarata ma senza valore.
- **Symbol**: Identificatore unico introdotto in ES6.

Esempio:

```
let nome = "Mario";             // Stringa
let eta = 30;                   // Numero
```

```
let attivo = true;              // Booleano
let colori = ["rosso", "blu"];  // Array
let persona = { nome: "Mario", eta: 30 };  // Oggetto
let nulla = null;               // Null
let sconosciuto;                // Undefined
```

7. Operatori

JavaScript include vari operatori per lavorare con i dati.

Operatori Aritmetici

```
let somma = 5 + 3;       // 8
let differenza = 5 - 3;  // 2
let prodotto = 5 * 3;    // 15
let quoziente = 5 / 3;   // 1.6666
let resto = 5 % 3;       // 2
```

Operatori di Confronto

Confrontano due valori e restituiscono un booleano.

```
5 == "5";   // true, confronto non rigoroso
5 === "5";  // false, confronto rigoroso (controlla anche il tipo)
5 != 3;     // true
5 !== "5";  // true
```

Operatori Logici

Utilizzati per combinare o invertire espressioni logiche.

```
true && false;  // false
true || false;  // true
!true;          // false
```

8. Strutture di Controllo e Cicli

Le strutture di controllo (come `if`, `else`, `switch`) e i cicli (`for`, `while`, `do...while`) permettono di controllare il flusso del programma.

Condizione `if` e `else`

```
let eta = 20;

if (eta >= 18) {
    console.log("Sei maggiorenne");
} else {
    console.log("Sei minorenne");
}
```

Ciclo `for`

```
for (let i = 0; i < 5; i++) {
    console.log(i); // Stampa 0, 1, 2, 3, 4
}
```

9. Funzioni

Le funzioni sono blocchi di codice che eseguono un compito specifico. Si possono dichiarare con `function`, e in ES6 è stata introdotta la **sintassi a freccia**.

```
javascript
Copia codice
function somma(a, b) {
    return a + b;
}

const sottrazione = (a, b) => a - b; // Funzione freccia
```

10. Template Literal

I **template literal** permettono di inserire variabili direttamente all'interno delle stringhe, senza bisogno di concatenare. Sono racchiusi in backtick (``) e supportano l'interpolazione con `${}`.

```
let nome = "Mario";
let messaggio = `Ciao, ${nome}! Come stai?`; // Ciao, Mario! Come stai?
```

11. Oggetti e Array

Gli oggetti sono collezioni di coppie chiave-valore, mentre gli array sono elenchi di valori.

```
let persona = { nome: "Mario", eta: 30 };
let colori = ["rosso", "blu", "verde"];
console.log(persona.nome);  // Mario
console.log(colori[1]);     // blu
```

12. Il DOM (Document Object Model)

JavaScript permette di manipolare il **DOM** per interagire con una pagina web. È possibile selezionare elementi, modificarli o gestire eventi su di essi.

Selezione di un elemento

```
let titolo = document.getElementById("titolo");  // Seleziona un elemento con ID
"titolo"
let paragrafi = document.querySelectorAll("p");  // Seleziona tutti i tag <p>
```

Modifica di contenuto ed eventi

```
titolo.textContent = "Nuovo Titolo";  // Cambia il testo
titolo.addEventListener("click", () => {
    alert("Hai cliccato sul titolo!");
});
```

13. JSON (JavaScript Object Notation)

JSON è un formato di testo leggero per scambiare dati. Si utilizza frequentemente per trasmettere dati tra server e client.

```
let persona = {
```

```
    nome: "Mario",
    eta: 30,
    attivo: true
};

let json = JSON.stringify(persona);  // Converte oggetto in stringa JSON
console.log(json); // {"nome":"Mario","eta":30,"attivo":true}

let obj = JSON.parse(json);  // Converte stringa JSON in oggetto
console.log(obj.nome);  // Mario
```

14. Asincronia con Callbacks, Promises e Async/Await

JavaScript supporta operazioni asincrone, come le richieste a un server. Questo avviene attraverso callback, `Promise`, e `async/await`.

Callback

Funzioni passate come argomento per gestire eventi asincroni.

```
function saluta(callback) {
    console.log("Ciao!");
    callback();
}

saluta(() => console.log("Come stai?"));
```

Promises

Una `Promise` rappresenta un'operazione asincrona che può concludersi con successo (`resolve`) o fallire (`reject`).

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Operazione completata"), 2000);
});

promise.then(valore => console.log(valore));  // Stampa "Operazione completata"
dopo 2 secondi
```

Async/Await

Introduzione di ES8, consente di scrivere codice asincrono in modo più semplice e leggibile.

```
async function esempio() {
    let valore = await promise;
    console.log(valore);  // "Operazione completata"
}
esempio();
```

Gli oggetti in JavaScript sono una delle componenti fondamentali del linguaggio e rappresentano il nucleo dello sviluppo a oggetti, che è ampiamente utilizzato per strutturare codice e dati in maniera organizzata e riutilizzabile. Vediamo in dettaglio cosa sono gli oggetti, come si creano, come funzionano e come applicare i principi della programmazione orientata agli oggetti (OOP) in JavaScript.

1. Introduzione agli Oggetti in JavaScript

Un **oggetto** in JavaScript è una collezione di **coppie chiave-valore** (dette anche **proprietà**), dove ogni **chiave** è una stringa che funge da nome, e ogni **valore** può essere un dato di qualsiasi tipo (numeri, stringhe, funzioni, array, altri oggetti). Gli oggetti sono estremamente versatili e permettono di rappresentare strutture dati complesse, come informazioni di una persona, un prodotto, o le impostazioni di un'applicazione.

Sintassi Base per Creare un Oggetto

Per creare un oggetto in JavaScript, utilizziamo la sintassi delle parentesi graffe `{}`:

```
let persona = {
    nome: "Mario",
    eta: 30,
    professione: "Ingegnere"
};
```

In questo esempio:

- `nome`, `eta`, e `professione` sono le **proprietà** dell'oggetto `persona`.
- `"Mario"`, `30`, e `"Ingegnere"` sono i **valori** delle proprietà.

Accesso alle Proprietà dell'Oggetto

Per accedere o modificare una proprietà, possiamo usare la notazione **punto** (`.`) o **parentesi quadre** (`[]`).

```
console.log(persona.nome);       // Mario
console.log(persona["professione"]); // Ingegnere

// Modifica di una proprietà
persona.eta = 31;
console.log(persona.eta);        // 31
```

Aggiungere e Rimuovere Proprietà

È possibile aggiungere nuove proprietà a un oggetto o rimuoverle anche dopo che l'oggetto è stato creato.

```
persona.nazionalita = "Italiana";   // Aggiunge una nuova proprietà
delete persona.professione;          // Rimuove una proprietà
```

2. Creazione di Oggetti e Costruttori

Esistono vari modi per creare oggetti in JavaScript, dai più semplici agli avanzati. Vediamo i principali.

a. Oggetti Letterali

Gli **oggetti letterali** sono la modalità più semplice di creare oggetti in JavaScript. Come visto sopra, si utilizzano parentesi graffe per dichiarare direttamente le proprietà e i valori.

```
let automobile = {
    marca: "Fiat",
    modello: "500",
    anno: 2021,
    avvia: function() {
        console.log("Motore avviato");
    }
};
```

In questo esempio, `automobile` è un oggetto con la proprietà `marca`, `modello`, `anno`, e un metodo `avvia`, che è una funzione definita all'interno dell'oggetto stesso.

b. Oggetti tramite Funzione Costruttore

Una **funzione costruttore** è una funzione speciale utilizzata per creare più istanze dello stesso tipo di oggetto. Viene chiamata con la parola chiave `new`, e permette di creare oggetti con proprietà e metodi condivisi.

```
function Persona(nome, eta) {
    this.nome = nome;
    this.eta = eta;
    this.saluta = function() {
        console.log("Ciao, mi chiamo " + this.nome);
    };
}

let mario = new Persona("Mario", 30);
let luca = new Persona("Luca", 25);

mario.saluta(); // Ciao, mi chiamo Mario
luca.saluta();  // Ciao, mi chiamo Luca
```

In questo caso:

- `Persona` è una funzione costruttore.
- `mario` e `luca` sono due istanze dell'oggetto `Persona`.

c. Classi ES6

Con **ES6** (ECMAScript 2015) sono state introdotte le classi, una sintassi che rende più leggibile e naturale la definizione di oggetti e il paradigma OOP in JavaScript.

```
class Persona {
    constructor(nome, eta) {
        this.nome = nome;
        this.eta = eta;
    }

    saluta() {
        console.log("Ciao, mi chiamo " + this.nome);
    }
}

let giulia = new Persona("Giulia", 28);
```

```
giulia.saluta(); // Ciao, mi chiamo Giulia
```

Con le **classi**, è possibile utilizzare la parola chiave `class` e il metodo speciale `constructor`, che inizializza le proprietà dell'oggetto. Il metodo `saluta` è un **metodo di classe**.

3. Programmazione Orientata agli Oggetti (OOP) in JavaScript

JavaScript supporta vari principi della programmazione orientata agli oggetti, tra cui **incapsulamento**, **ereditarietà**, e **polimorfismo**.

a. Incapsulamento

L'incapsulamento permette di raggruppare proprietà e metodi legati a una stessa entità. Le proprietà e i metodi sono incapsulati all'interno dell'oggetto.

Con **classi ES6**, possiamo usare **modificatori di accesso privato** (#) per proteggere le proprietà e rendere disponibile solo l'interfaccia pubblica:

```
class ContoCorrente {
    #saldo = 0;

    deposita(importo) {
        if (importo > 0) this.#saldo += importo;
    }

    preleva(importo) {
        if (importo > 0 && importo <= this.#saldo) this.#saldo -= importo;
    }

    visualizzaSaldo() {
        console.log("Saldo attuale: " + this.#saldo);
    }
}

let conto = new ContoCorrente();
conto.deposita(1000);
conto.visualizzaSaldo();  // Saldo attuale: 1000
// conto.#saldo = 500;    // ERRORE! La proprietà è privata
```

b. Ereditarietà

L'ereditarietà permette di creare classi basate su altre classi. In JavaScript, si può creare una **classe derivata** che eredita le proprietà e i metodi di una **classe base** usando la parola chiave `extends`.

```
class Animale {
    constructor(nome) {
        this.nome = nome;
    }

    muove() {
        console.log(this.nome + " si sta muovendo");
    }
}

class Cane extends Animale {
```

```
    abbaia() {
        console.log(this.nome + " fa Bau Bau!");
    }
}

let fido = new Cane("Fido");
fido.muove();   // Fido si sta muovendo
fido.abbaia();  // Fido fa Bau Bau!
```

c. Polimorfismo

Il polimorfismo consente di utilizzare la stessa interfaccia per oggetti di tipi diversi. Questo è possibile in JavaScript sovrascrivendo i metodi nelle classi figlie.

4. Oggetti Incorporati (Built-in Objects)

JavaScript include molti oggetti incorporati che permettono di lavorare con dati, testo, date, e altro.

a. Oggetto `Math`

L'oggetto `Math` contiene funzioni matematiche e costanti.

```
console.log(Math.PI);              // 3.14159...
console.log(Math.sqrt(16));        // 4
console.log(Math.max(5, 10, 15));  // 15
console.log(Math.random());        // Numero casuale tra 0 e 1
```

b. Oggetto `Date`

L'oggetto `Date` permette di gestire e manipolare date e orari.

```
let oggi = new Date();
console.log(oggi);                     // Data e ora attuale
console.log(oggi.getFullYear());       // Anno attuale
console.log(oggi.getMonth() + 1);      // Mese (da 0 a 11)
console.log(oggi.getDate());           // Giorno del mese
```

c. Oggetto `String`

L'oggetto `String` fornisce metodi per manipolare stringhe.

```
let testo = "Ciao, come stai?";
console.log(testo.length);            // Lunghezza della stringa
console.log(testo.toUpperCase());     // CIAO, COME STAI?
console.log(testo.includes("come"));  // true
```

d. Oggetto `Array`

L'oggetto `Array` contiene una serie di metodi per lavorare con le liste di dati.

```
let numeri = [1, 2, 3, 4, 5];
console.log(numeri.length);             // 5
console.log(numeri.join("-"));          // 1-2-3-4-5
numeri.push(6);                         // Aggiunge 6 alla fine
```

```
console.log(numeri);                    // [1, 2, 3, 4, 5, 6]
```

5. Array e Oggetti come Dati Complessi

Gli **array di oggetti** sono estremamente utili per rappresentare insiemi di dati complessi.

```
let studenti = [
    { nome: "Mario", voto: 28 },
    { nome: "Luca", voto: 30 },
    { nome: "Giulia", voto: 27 }
];

studenti.forEach(studente => console.log(studente.nome + ": " + studente.voto));
```

6. Oggetti Dinamici e Funzioni Costruttore

JavaScript permette di **creare oggetti dinamicamente**, sia con funzioni costruttore sia con **Object.create** per ereditarietà prototipale avanzata.

```
let base = { saluto() { console.log("Ciao!"); } };
let nuovoOggetto = Object.create(base);
nuovoOggetto.saluto();  // Ciao!
```

Crea una applicazione javaScript

Per creare una semplice applicazione web con **login**, **menu di scelta**, e **pagine dinamiche**, svilupperemo una serie di file HTML, CSS e JavaScript. L'applicazione avrà:

1. **Pagina di login**: dove l'utente inserisce il suo nome utente e la password.
2. **Menu di scelta**: che permette all'utente di scegliere tra 5 opzioni.
3. **5 Pagine dinamiche**: che vengono mostrate quando l'utente seleziona un'opzione dal menu.

1. Schema Funzionale e Strutturale

Funzionalità principali:

- **Login**: L'utente inserisce le credenziali (nome utente e password).
- **Menu**: Dopo il login, l'utente viene rediretto a una pagina con un menu contenente 5 bottoni, ciascuno che porta a una pagina diversa.
- **Pagine dinamiche**: Ogni pulsante nel menu visualizza una pagina distinta.

2. Struttura del Progetto

```
/index.html          (Pagina di login)
/menu.html           (Pagina con il menu e i 5 pulsanti)
/pagina1.html        (Contenuto per la pagina 1)
/pagina2.html        (Contenuto per la pagina 2)
/pagina3.html        (Contenuto per la pagina 3)
/pagina4.html        (Contenuto per la pagina 4)
```

```
/pagina5.html        (Contenuto per la pagina 5)
/style.css           (Stili CSS)
/script.js           (Logica JavaScript per la navigazione e login)
```

3. File HTML:

3.1. `index.html` - Pagina di Login

```
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <form id="loginForm">
            <label for="username">Nome utente:</label>
            <input type="text" id="username" required><br>
            <label for="password">Password:</label>
            <input type="password" id="password" required><br>
            <button type="submit">Accedi</button>
        </form>
        <p id="error-message"></p>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

3.2. `menu.html` - Pagina del Menu

```
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Menu</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="menu-container">
        <h2>Benvenuto nel menu</h2>
        <button onclick="window.location.href='pagina1.html'">Pagina 1</button>
        <button onclick="window.location.href='pagina2.html'">Pagina 2</button>
        <button onclick="window.location.href='pagina3.html'">Pagina 3</button>
        <button onclick="window.location.href='pagina4.html'">Pagina 4</button>
        <button onclick="window.location.href='pagina5.html'">Pagina 5</button>
    </div>
</body>
</html>
```

3.3. Esempio di `pagina1.html` (le altre pagine seguiranno lo stesso schema)

```
<!DOCTYPE html>
<html lang="it">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pagina 1</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="page-container">
        <h2>Contenuto della Pagina 1</h2>
        <p>Benvenuto nella pagina 1.</p>
        <a href="menu.html">Torna al menu</a>
    </div>
</body>
</html>
```

Ripeti questa struttura per le altre 4 pagine (`pagina2.html`, `pagina3.html`, `pagina4.html`, `pagina5.html`).

4. File CSS:

`style.css` - Stili Base per l'Applicazione

```
/* Stili di base per il login */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #f0f0f0;
}

h2 {
    text-align: center;
}

.login-container, .menu-container, .page-container {
    background: #fff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    text-align: center;
    width: 300px;
}

input {
    width: 100%;
    padding: 8px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    padding: 10px 20px;
    background-color: #4CAF50;
    color: white;
    border: none;
```

```
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #45a049;
}

button:disabled {
    background-color: #ccc;
}

#error-message {
    color: red;
    font-size: 14px;
}
```

5. File JavaScript:

`script.js` - Logica JavaScript per il Login e la Navigazione

```
// Login
document.getElementById('loginForm').addEventListener('submit', function(event)
{
    event.preventDefault();

    let username = document.getElementById('username').value;
    let password = document.getElementById('password').value;

    if (username === "admin" && password === "password123") {
        // Login riuscito
        window.location.href = 'menu.html';
    } else {
        // Login fallito
        document.getElementById('error-message').innerText = "Credenziali
errate. Riprova.";
    }
});
```

- Il login controlla se il nome utente è "admin" e la password è "password123".
- Se le credenziali sono corrette, l'utente viene reindirizzato alla pagina `menu.html`.
- In caso contrario, viene visualizzato un messaggio di errore.

6. Funzionamento dell'Applicazione:

1. **Pagina di Login** (`index.html`):
    - L'utente inserisce le proprie credenziali (nome utente e password).
    - Se le credenziali sono corrette, l'utente viene reindirizzato alla pagina del menu.
    - Se le credenziali sono errate, viene visualizzato un messaggio di errore.
2. **Menu di Scelta** (`menu.html`):
    - Dopo aver effettuato il login, l'utente vede un menu con 5 bottoni, ciascuno che porta a una pagina diversa (`pagina1.html`, `pagina2.html`, etc.).
3. **Pagine** (`pagina1.html`, `pagina2.html`, etc.):
    - Ogni pagina contiene contenuti specifici e un link che permette di tornare al menu principale.

Conclusioni:

Questa applicazione di esempio offre una base semplice ma funzionale con login, navigazione tramite un menu, e 5 pagine dinamiche.

Andiamo a spiegare ogni parte del file **`script.js`** riga per riga, in modo che si possa capire come funziona il codice.

1. Aggiungere un Listener al Modulo di Login

```
document.getElementById('loginForm').addEventListener('submit', function(event)
{
```

- **`document.getElementById('loginForm')`**:
  - Otteniamo il riferimento all'elemento HTML con l'ID `loginForm`, che è il modulo di login. Questo è il modulo dove l'utente inserisce il nome utente e la password.
- **`.addEventListener('submit', function(event) {...})`**:
  - Aggiungiamo un "event listener" (un ascoltatore di eventi) all'elemento del modulo, in particolare l'evento `submit`.
  - L'evento `submit` viene attivato quando l'utente invia il modulo (cliccando sul pulsante di login).
  - La funzione anonima (senza nome) che passa come secondo argomento viene eseguita quando l'evento `submit` si verifica. In questa funzione, `event` è l'oggetto evento che contiene informazioni sull'evento che si è verificato.

2. Prevenire il Comportamento Predefinito del Form

```
event.preventDefault();
```

- **`event.preventDefault()`**:
  - In JavaScript, i moduli HTML hanno un comportamento predefinito, che consiste nell'inviare i dati al server (di solito a un URL specificato nell'attributo `action` del modulo) e ricaricare la pagina.
  - Utilizzando `event.preventDefault()`, impediamo che il modulo venga effettivamente inviato, così possiamo gestire l'autenticazione tramite JavaScript senza ricaricare la pagina.

3. Recuperare i Valori di Nome Utente e Password

```
let username = document.getElementById('username').value;
let password = document.getElementById('password').value;
```

- **`document.getElementById('username')`**:
  - Otteniamo l'elemento HTML con l'ID `username`, che è il campo di input dove l'utente inserisce il nome utente.
- **`.value`**:
  - `.value` è una proprietà che restituisce il valore attualmente inserito nell'elemento di input. In questo caso, recuperiamo il nome utente inserito.
- **`let username = ...`**:
  - La variabile `username` memorizza il valore del campo di input `username` inserito dall'utente.

- La stessa cosa accade per la variabile `password`, che memorizza il valore del campo di input `password`.

4. Controllare le Credenziali dell'Utente

```
if (username === "admin" && password === "password123") {
```

- **`if (username === "admin" && password === "password123")`**:
  - Qui stiamo verificando se il nome utente inserito dall'utente è `"admin"` e se la password inserita è `"password123"`.
  - La condizione utilizza l'operatore logico **`&&`** (AND), il che significa che entrambe le condizioni devono essere vere affinché il blocco di codice all'interno dell'`if` venga eseguito.
  - Se le credenziali sono corrette (nome utente = `"admin"` e password = `"password123"`), il codice entra all'interno del blocco `if`.

5. Reindirizzare l'Utente alla Pagina del Menu

```
window.location.href = 'menu.html';
```

- **`window.location.href = 'menu.html';`**:
  - `window.location.href` è una proprietà dell'oggetto `window` che ci permette di cambiare la URL della pagina corrente.
  - Impostando `window.location.href` su `'menu.html'`, il browser reindirizza l'utente alla pagina `menu.html`. In altre parole, se le credenziali sono corrette, l'utente verrà portato al menu.

6. Gestire il Caso di Login Fallito

```
} else {
    document.getElementById('error-message').innerText = "Credenziali errate.
Riprova.";
}
```

- **`else`**:
  - Se la condizione nell'`if` è falsa (cioè se le credenziali sono errate), allora il codice all'interno del blocco `else` viene eseguito.
- **`document.getElementById('error-message')`**:
  - Otteniamo l'elemento HTML con l'ID `error-message`, che è un paragrafo (`<p>`) situato nella pagina di login. Questo paragrafo è utilizzato per visualizzare eventuali messaggi di errore.
- **`.innerText = "Credenziali errate. Riprova."`**:
  - La proprietà `.innerText` permette di modificare il contenuto testuale di un elemento HTML.
  - In questo caso, se le credenziali sono errate, mostriamo il messaggio `"Credenziali errate. Riprova."` all'interno dell'elemento con l'ID `error-message`. Questo messaggio aiuta l'utente a capire che qualcosa è andato storto durante il login.

7. Conclusione

Il codice completo del file `script.js` permette di gestire il processo di login. Quando l'utente invia il modulo di login, il codice:

1. Impedisce che la pagina si ricarichi (comportamento predefinito del form).
2. Recupera il nome utente e la password inseriti.
3. Verifica che le credenziali corrispondano a quelle predefinite (in questo caso, `admin` e `password123`).
4. Se le credenziali sono corrette, l'utente viene reindirizzato alla pagina del menu (`menu.html`).
5. Se le credenziali sono errate, viene visualizzato un messaggio di errore.

Riepilogo del Funzionamento:

1. **Input dell'utente**: L'utente inserisce il nome utente e la password.
2. **Verifica delle credenziali**: Le credenziali vengono verificate con un confronto di stringhe.
3. **Reindirizzamento o errore**: Se le credenziali sono corrette, l'utente viene reindirizzato al menu. Altrimenti, viene mostrato un messaggio di errore.

Questa struttura base del login è utile per applicazioni semplici, ma nelle applicazioni reali sarebbe necessario implementare una logica di autenticazione più robusta (ad esempio, con l'uso di un server backend e un database).

Spiegare dettagliatamente questa riga: <label for="username">Nome utente:</label> e il significato e l'utilizzo di for.

```
<label for="username">Nome utente:</label>
```

Cos'è un elemento `<label>`?

L'elemento `<label>` in HTML è utilizzato per definire una **etichetta** associata a un altro elemento di input (ad esempio un campo di testo, una casella di controllo, un pulsante di opzione, ecc.). La funzione principale di un `<label>` è quella di fornire una descrizione visiva che aiuti l'utente a capire cosa deve inserire in un determinato campo.

, `<label>` è usato per associare un testo ("Nome utente:") a un campo di input dove l'utente deve inserire il suo nome utente.

Cosa fa l'attributo `for`?

L'attributo **`for`** è un attributo molto importante per l'elemento `<label>`. Esso specifica quale elemento di input il `<label>` descrive. Il valore dell'attributo `for` deve corrispondere all'**ID** di un altro elemento HTML, di solito un campo di input. In questo caso, il valore di `for` è `username`.

Ecco come funziona:

```
<label for="username">Nome utente:</label>
<input type="text" id="username" name="username">
```

Significato e utilizzo dell'attributo `for`:

1. **Associazione tra la `<label>` e l'elemento di input**:

    - L'attributo `for="username"` crea un collegamento tra l'etichetta (`<label>`) e l'elemento di input con l'attributo `id="username"`.
    - Questo significa che quando l'utente clicca sull'etichetta "Nome utente:", il cursore si sposterà automaticamente nel campo di input corrispondente, migliorando l'interazione con il form.
2. **Accessibilità**:
    - L'uso dell'attributo `for` migliora l'accessibilità per gli utenti con disabilità, in particolare quelli che utilizzano lettori di schermo (screen reader). Se l'elemento `<label>` è associato correttamente a un input tramite l'attributo `for`, i lettori di schermo potranno leggere sia l'etichetta che il campo di input in modo appropriato, facilitando l'esperienza dell'utente.
3. **Aumento dell'area cliccabile**:
    - Senza l'attributo `for`, l'area cliccabile per il campo di input è solo la sua stessa area (dove si trova il cursore). Con l'attributo `for`, però, l'area cliccabile viene estesa all'intero testo dell'etichetta. Questo significa che l'utente può cliccare sul testo "Nome utente:" per spostarsi nel campo di input, anziché dover cliccare direttamente nel campo di input stesso.

Comportamento con l'attributo `for`:

1. **Senso di interazione**:
    - Cliccando sull'etichetta "Nome utente:", il focus si sposterà sul campo di input associato, quindi l'utente potrà subito digitare senza bisogno di cliccare direttamente nel campo.
2. **Esempio pratico**: Supponiamo di avere un modulo con un campo di input per il nome utente:

```
<form>
    <label for="username">Nome utente:</label>
    <input type="text" id="username" name="username">
</form>
```

In questo esempio:

- L'etichetta "Nome utente:" è associata al campo di input con `id="username"`.
- Se l'utente clicca sull'etichetta, il focus (cioè, la posizione in cui l'utente può iniziare a digitare) verrà automaticamente posizionato nel campo di input.