PROGRAMMAZIONE

ASINCRONA

COS'È LA PROGRAMMAZIONE ASINCRONA?

La **programmazione asincrona** in JavaScript è
un modo di gestire le operazioni che richiedono del tempo senza
bloccare il resto del codice.

Invece di aspettare che un'operazione si completi prima di
passare alla successiva, il programma può continuare a eseguire
altre attività mentre l'operazione in background procede.

DEFINIZIONE

- La programmazione asincrona si riferisce all'esecuzione di operazioni senza dover attendere il completamento di ciascuna operazione prima di passare alla successiva.

- In un contesto sincrono, ogni operazione viene eseguita una dopo l'altra, bloccando l'esecuzione del programma fino al completamento dell'operazione corrente.

IMPORTANZA NELLO SVILUPPO WEB

- Nell'ambito dello sviluppo web, molte operazioni richiedono tempo per essere eseguite, come il download di file, l'accesso a database o il recupero di risorse esterne.

- La programmazione asincrona consente di gestire queste operazioni in modo efficiente, migliorando la reattività delle applicazioni web.

CALLBACKS

CALLBACKS

- Un **callback** è una funzione che viene passata come argomento a un'altra funzione.

Viene chiamato una volta che l'operazione asincrona è stata completata.

- I **callbacks** consentono di gestire il flusso di esecuzione in situazioni in cui l'ordine delle operazioni non è noto a priori.

```
function eseguiOperazioneAsync(callback) {
    // Simuliamo un'operazione asincrona
    setTimeout(function() {
        console.log("Operazione asincrona completata");
        callback(); // Chiamiamo il callback dopo il completamento
    }, 2000);
}

function gestisciCallback() {
    console.log("Callback eseguito con successo!");
}

// Utilizziamo la funzione con un callback
eseguiOperazioneAsync(gestisciCallback);
```

CALLBACK HELL

- Quando si gestiscono molteplici operazioni asincrone, i callbacks possono portare a una struttura di codice annidata, nota come "callback hell" o "pyramid of doom".

- Questo rende il codice difficile da leggere e da mantenere.

PROMISES

PROMISES - AKA “MIGLIORAMENTO DELLE CALLBACKS”

- Una **Promise** è un oggetto che rappresenta il risultato futuro di un'operazione asincrona.

- Può essere in uno degli stati: **pending**, **fulfilled** (risolta con successo), o **rejected** (rifiutata con errore).

- Le Promises sono progettate per semplificare la gestione delle operazioni asincrone e migliorare la leggibilità del codice.

```
function eseguiOperazioneAsync() {
    return new Promise(function(resolve, reject) {
        // Simuliamo un'operazione asincrona
        setTimeout(function() {
            console.log("Operazione asincrona completata");
            resolve(); // Indichiamo che l'operazione è completata con successo
        }, 2000);
    });
}

function gestisciPromise() {
    console.log("Promise risolta con successo!");
}

// Utilizziamo le Promises
eseguiOperazioneAsync()
    .then(gestisciPromise)
    .catch(function(error) {
        console.error("Si è verificato un errore:", error);
    });
```

THEN & CATCH

- **.then()** viene utilizzato per specificare cosa fare dopo che la Promise è stata risolta con successo.

- **.catch()** viene utilizzato per gestire eventuali errori durante l'esecuzione dell'operazione asincrona.

ASYNC

AWAIT

ASYNC/AWAIT - SINTASSI MODERNA PER LA PROGRAMMAZIONE ASINCRONA

- **Async/Await** è una sintassi moderna per gestire le operazioni asincrone in JavaScript.

- Introduce le parole chiave **async** e **await** per rendere il codice più leggibile e simile alla programmazione sincrona

```
async function eseguiOperazioneAsync() {
    return new Promise(function(resolve) {
        // Simuliamo un'operazione asincrona
        setTimeout(function() {
            console.log("Operazione asincrona completata");
            resolve(); // Indichiamo che l'operazione è completata con successo
        }, 2000);
    });
}

async function gestisciAsyncAwait() {
    try {
        await eseguiOperazioneAsync(); // Attendiamo il completamento dell'operazione
asincrona
        console.log("Async/Await: Operazione completata con successo!");
    } catch (error) {
        console.error("Si è verificato un errore:", error);
    }
}

// Utilizziamo Async/Await
gestisciAsyncAwait();
```