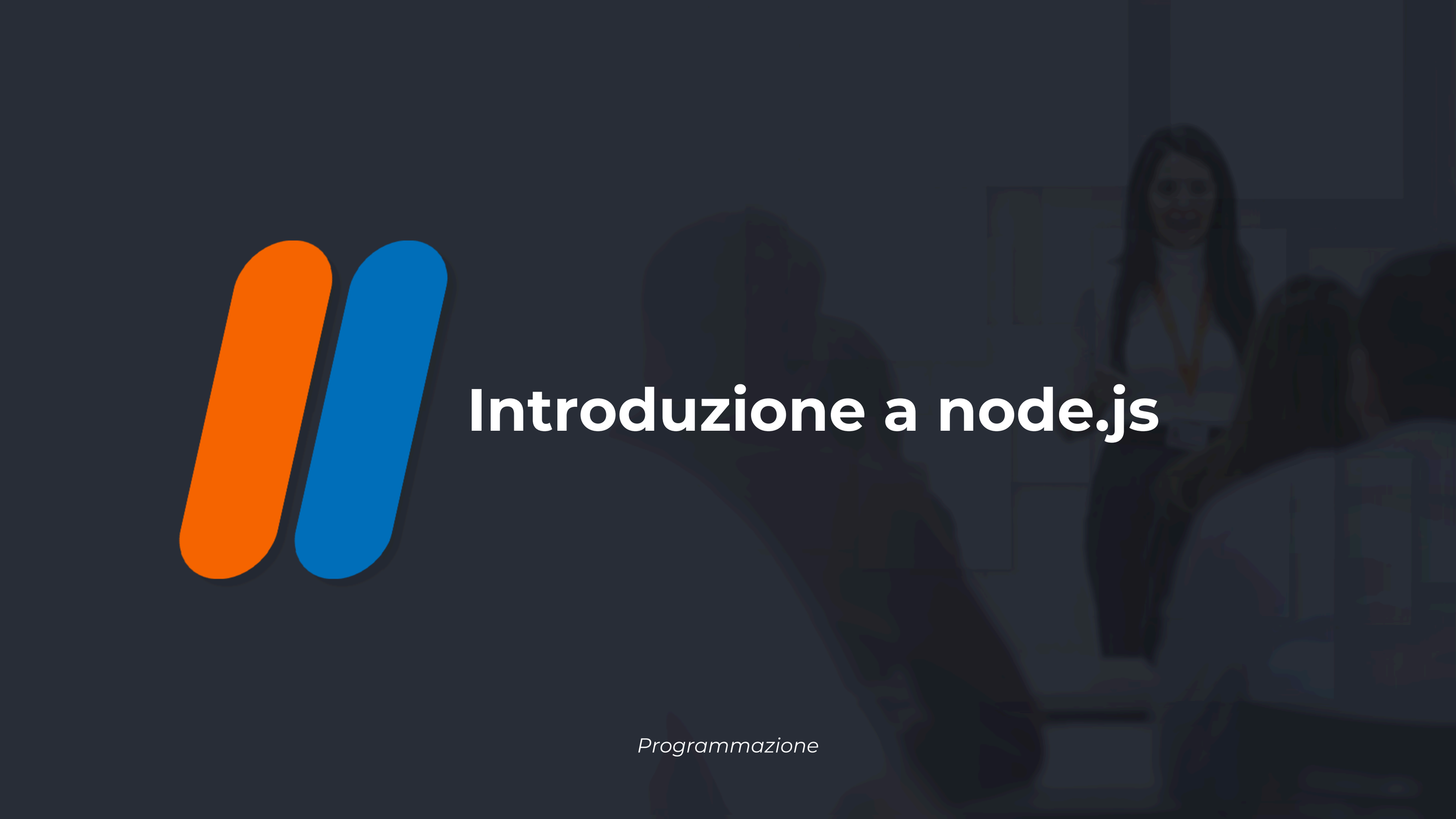


Introduzione a node.js

Programmazione

Node.js è un runtime JavaScript lato server costruito sul motore V8 di Chrome.

Permette di eseguire codice JavaScript fuori dal browser, tipicamente su un server, rendendo possibile creare applicazioni web complete solo con JavaScript.

Caratteristiche principali

- **Event-driven: utilizza un modello asincrono basato su eventi, ideale per applicazioni che gestiscono molte richieste simultanee.**
- **Single-threaded, non-blocking: esegue tutto su un solo thread ma con I/O non bloccante, permettendo performance elevate con bassi consumi.**
- **Cross-platform: funziona su Windows, macOS e Linux.**
- **NPM (Node Package Manager): include un vastissimo ecosistema di moduli e pacchetti open source pronti all’uso.**

Perché usarlo

- Per creare API REST, app real-time (chat, giochi), server web, automazioni, microservizi.
- Ideale per sviluppatori frontend JavaScript che vogliono anche lavorare lato backend (full stack).

Il ciclo di vita di un'app Node.js

1. **Avvio: l’app viene avviata con node nomefile.js.**
2. **Registrazione eventi/callback: Node registra funzioni da eseguire su eventi (es. richiesta HTTP).**
3. **Event loop: gestisce continuamente eventi e callback.**
4. **Termine: l'app termina solo se non ci sono più eventi da gestire.**

Il modulo require

Node.js utilizza require() per importare moduli:

```
1.const fs = require('fs');
2.// modulo built-in per lavorare con file
```

Esistono 3 tipi di moduli:

- **Built-in (come fs, http, path)**
- **Terze parti (scaricabili da npm, es. express)**
- **Personalizzati (scritti dall’utente)**

Node.js adotta un sistema modulare per organizzare e riutilizzare il codice.

Ogni file .js è considerato un modulo autonomo e può esportare funzioni, oggetti o variabili da riutilizzare in altri file tramite il comando require().

Questo approccio rende le applicazioni più scalabili, leggibili e manutenibili.

Moduli Built-in (nativi)

Descrizione: Sono moduli già inclusi nel core di Node.js, non richiedono installazione.

Esempio:

```
1. const fs = require('fs');
2. fs.writeFileSync('test.txt', 'Ciao da Node!');
```

Spiegazione:

- **require('fs') importa il modulo fs (file system).**
- **writeFileSync() crea o sovrascrive un file scrivendo il testo indicato.**
- **Questo è sincrono: blocca l'esecuzione finché l'operazione non è finita.**

Altri moduli built-in: http, path, os, events.

Il modulo http

Con http, possiamo creare un semplice server:

```
1. const http = require('http');
2.
3. http.createServer((req, res) => {
4.   res.end('Hello World');
5. }).listen(3000);
```

Moduli di terze parti (npm)

Descrizione: Sono moduli sviluppati dalla community e pubblicati su npmjs.com.

Vanno installati con npm install.

- *npm install moment*

1. *const moment = require('moment');*
2. *console.log(moment().format('YYYY-MM-DD'));*

Spiegazione:

moment è una libreria per gestire date e orari.
moment() restituisce la data attuale.

format() la trasforma in formato leggibile (YYYY-MM-DD = 2025-06-19, ad esempio).

Il file package.json

È il cuore del progetto Node.js. Contiene:

- **Nome, versione, descrizione**
- **Dipendenze (dependencies)**
- **Script di esecuzione (scripts)**

Può essere creato con:

1. *npm init*

Moduli personalizzati

Descrizione: Sono file .js scritti dall'utente e importati localmente.

Esempio:

calcoli.js

```
function somma(a, b) {
  return a + b;
}
module.exports = somma;
```

main.js

```
const somma = require('./calcoli');
console.log(somma(2, 3));
```

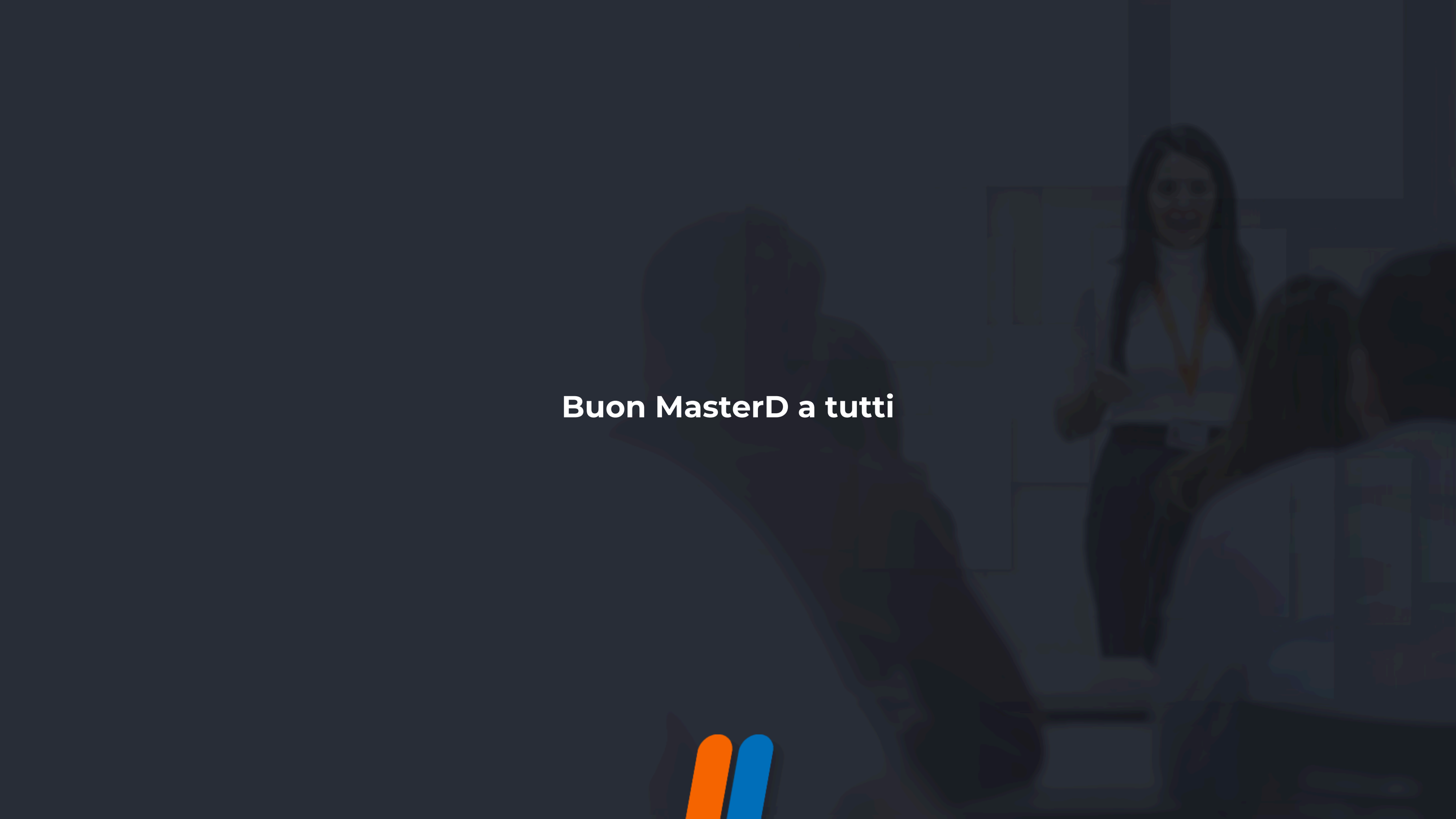
Spiegazione:

- **module.exports espone la funzione somma.**
- **require('./calcoli') importa il modulo locale (si usa il ./ per i file locali).**
- **Il risultato 5 viene stampato.**

Abbiamo visto cos’è Node.js, perché è potente per lo sviluppo lato server, e come sfrutta un sistema modulare per organizzare il codice.

Abbiamo distinto tra moduli nativi, di terze parti e personalizzati, analizzandone esempi pratici e struttura.

Node.js si presenta quindi come uno strumento leggero, efficiente e scalabile, adatto a sviluppare backend moderni, microservizi, automazioni e servizi web in real-time, con il vantaggio di usare un solo linguaggio (JavaScript) su tutta la stack.


Buon MasterD a tutti