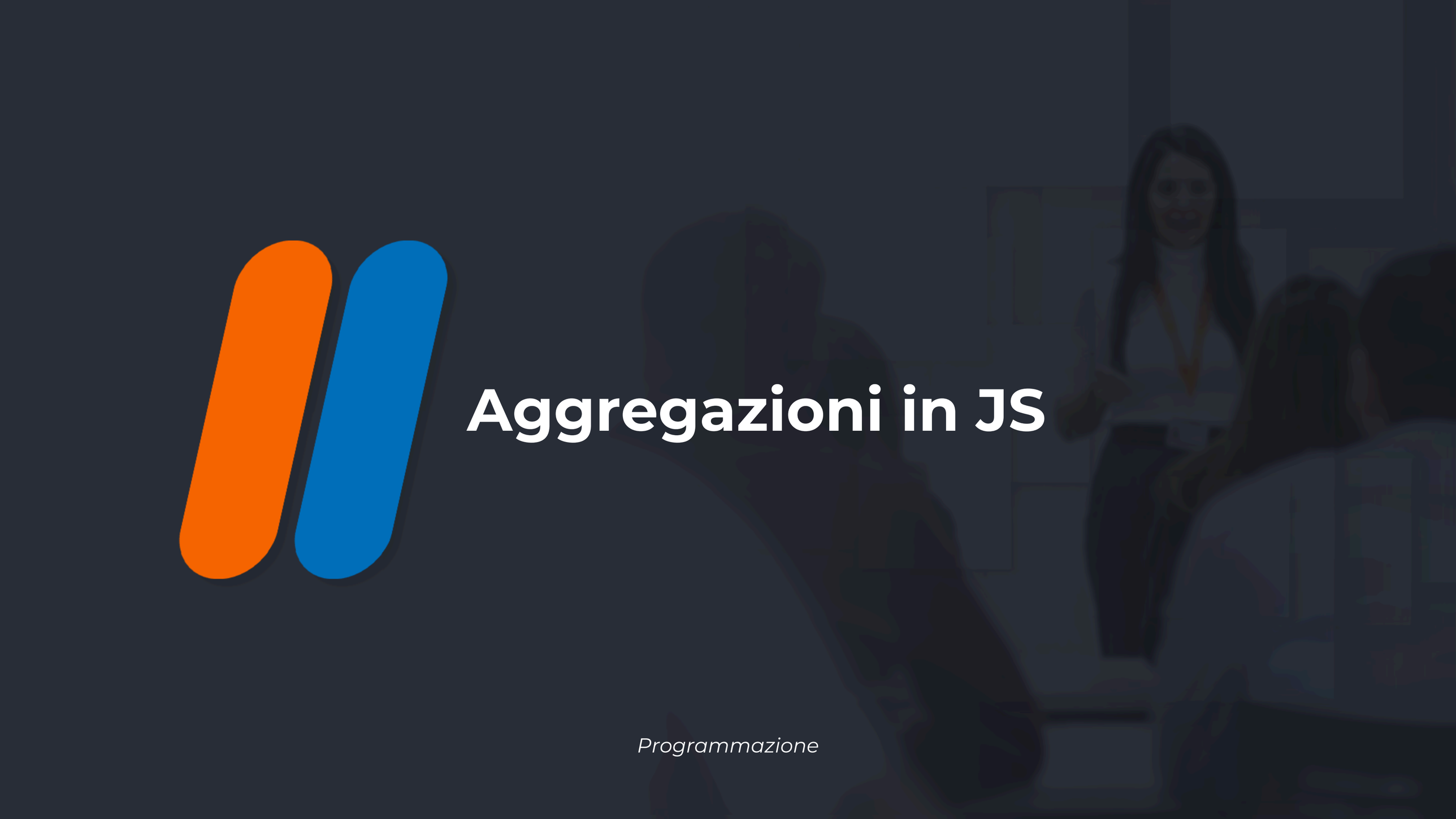


Aggregazioni in JS

Programmazione

Introduzione alle aggregazioni in JavaScript

Le aggregazioni in JavaScript si riferiscono a tutte quelle operazioni che permettono di combinare, ridurre o riassumere insiemi di dati, in genere contenuti in array o strutture simili.

Queste operazioni sono fondamentali nella manipolazione dei dati, poiché consentono di passare da un insieme di valori a un singolo risultato o a una nuova struttura.

In modo analogo a ciò che avviene nei linguaggi di programmazione funzionale, JavaScript mette a disposizione metodi integrati che rendono più leggibili, efficienti e dichiarativi i processi di calcolo o trasformazione.

Introduzione alle aggregazioni in JavaScript

L’obiettivo delle aggregazioni è quindi trasformare o sintetizzare dati complessi in informazioni utili.

Ad esempio, si possono sommare tutti i valori di un array, filtrare solo quelli che rispettano certe condizioni, o creare nuovi array basati su trasformazioni specifiche.

Queste operazioni sono alla base della programmazione moderna in JS, specialmente nello sviluppo frontend (ad esempio nella gestione di dati provenienti da API) e backend (come nel trattamento di dataset o oggetti JSON).

Array

Struttura ordinata di elementi indicizzati numericamente.

Permette di memorizzare più valori (anche di tipo diverso) e di accedervi tramite l'indice.

È la collezione più usata in JS e supporta metodi come map(), filter(), reduce(), ecc.

Esempio:

```
1. let numeri = [10, 20, 30];
2. console.log(numeri[1]); // 20
```

Object

Struttura di dati basata su coppie chiave–valore, ideale per rappresentare entità con attributi.

Le chiavi sono stringhe o simboli, e i valori possono essere di qualsiasi tipo (numeri, stringhe, array, funzioni, ecc.).

Esempio:

```
1. let persona = { nome: "Luca", età: 25 };
2. console.log(persona.nome); // "Luca"
```

Set

Collezione non ordinata e senza duplicati.

Utile quando serve garantire l’unicità dei valori (ad esempio per eliminare duplicati da un array).

Esempio:

```
1. let numeri = new Set([1, 2, 2, 3]);
2. console.log(numeri); // Set(3) {1, 2, 3}
```

Map

Collezione che memorizza coppie chiave–valore, come un Object, ma con vantaggi:

le chiavi possono essere di qualsiasi tipo (non solo stringhe) e mantiene l’ordine di inserimento.

Esempio:

```
1. let mappa = new Map();
2. mappa.set("nome", "Anna");
3. mappa.set("età", 30);
4. console.log(mappa.get("nome")); // "Anna"
```

Principali tecniche e metodi di aggregazione in JavaScript

- **forEach() →**

Permette di iterare su ogni elemento di un array ed eseguire un'azione specifica. Non restituisce un nuovo array, ma è utile per applicare effetti collaterali come stampe o aggiornamenti.

Esempio: stampare tutti gli elementi di un array.

- **map() →**

Crea un nuovo array applicando una funzione a ciascun elemento dell'array originale. È utile per trasformare i dati, ad esempio convertendo numeri in stringhe o moltiplicando valori.

Principali tecniche e metodi di aggregazione in JavaScript

- **filter() →**

Restituisce un nuovo array contenente solo gli elementi che soddisfano una condizione logica. Serve per selezionare un sottoinsieme di dati in base a criteri.

- **reduce() →**

Combina tutti gli elementi di un array in un singolo valore, come una somma, un prodotto o un oggetto riassuntivo. È il metodo di aggregazione per eccellenza.

Principali tecniche e metodi di aggregazione in JavaScript

- **filter() →**

Restituisce un nuovo array contenente solo gli elementi che soddisfano una condizione logica. Serve per selezionare un sottoinsieme di dati in base a criteri.

- **reduce() →**

Combina tutti gli elementi di un array in un singolo valore, come una somma, un prodotto o un oggetto riassuntivo. È il metodo di aggregazione per eccellenza.

```
let numeri = [1, 2, 3, 4, 5];

// forEach esegue una funzione su ogni elemento dell'array

numeri.forEach(function(numero) {
  console.log("Numero attuale:", numero);

});
```

Spiegazione:

- **Il metodo forEach() serve per eseguire un’azione su ogni elemento di un array.**
- **Non restituisce nulla (il valore di ritorno è undefined), ma è utile per operazioni come stampe, conteggi, o modifiche esterne.**
- **In questo esempio stampa uno per uno tutti i numeri dell’array.**

```
let numeri = [1, 2, 3, 4, 5];

// map crea un nuovo array moltiplicando ogni elemento per 2

let doppi = numeri.map(function(numero) {
  return numero * 2;
});

console.log(doppi); // [2, 4, 6, 8, 10]
```

Spiegazione:

- **map() crea un nuovo array trasformando ciascun elemento in base alla funzione fornita.**
- **Non modifica l'array originale ma restituisce un nuovo insieme di dati trasformati.**
- **È utile per conversioni o modifiche in massa, come cambiare unità di misura o moltiplicare valori.**

```
let numeri = [1, 2, 3, 4, 5, 6];

// filter restituisce solo i numeri pari

let pari = numeri.filter(function(numero) {
  return numero % 2 === 0;
});

console.log(pari); // [2, 4, 6]
```

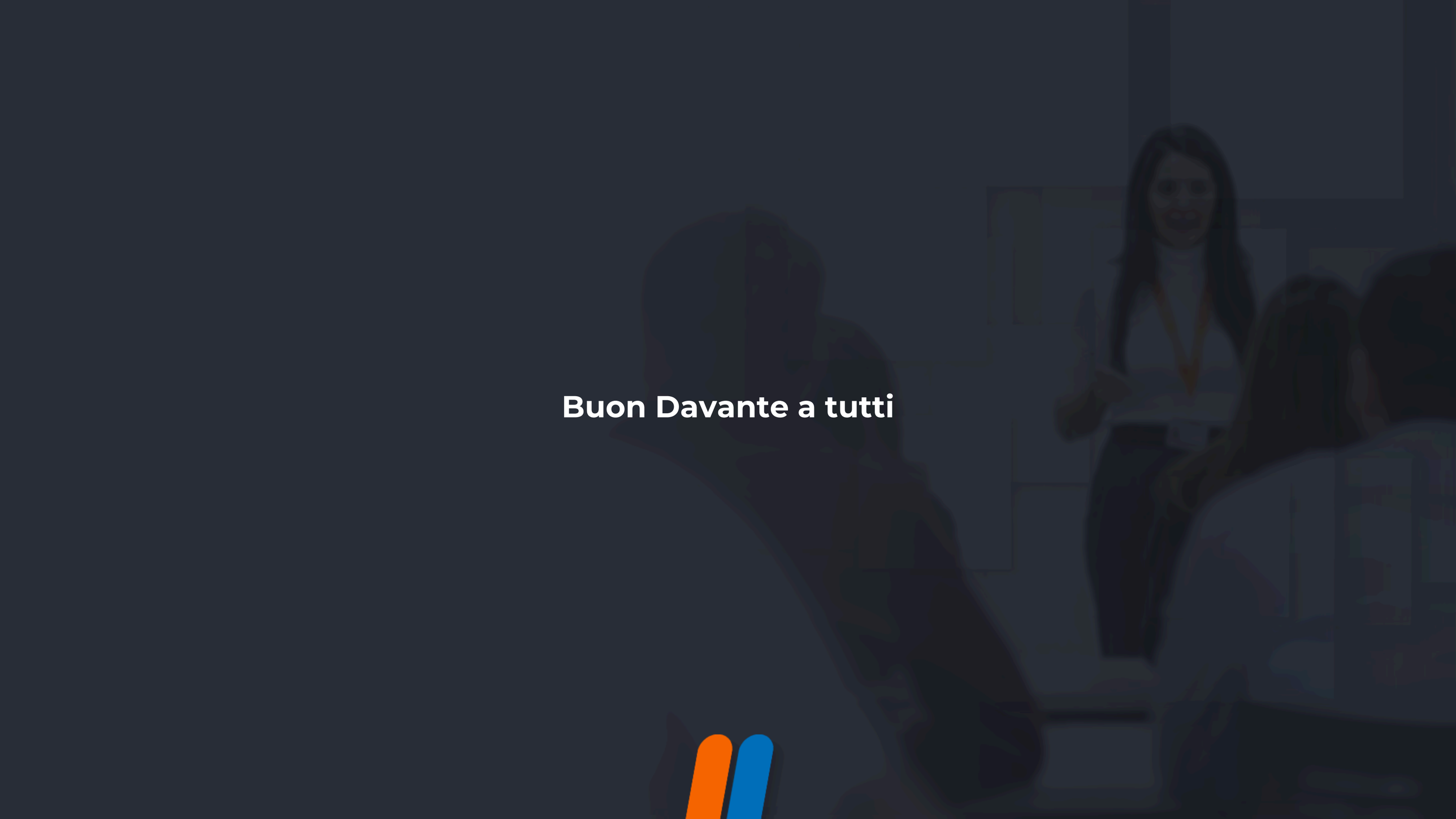
Spiegazione:

- **filter() genera un nuovo array contenente solo gli elementi che rispettano la condizione indicata.**
- **Qui, la funzione controlla se ogni numero è pari (numero % 2 === 0), e restituisce solo quelli che lo sono.**
- **È ideale per selezionare dati specifici da una collezione.**

```
let numeri = [10, 20, 30, 40];

// reduce combina tutti gli elementi in un unico valore

let somma = numeri.reduce(function(totale, numero) {
  return totale + numero;
}, 0);

console.log(somma); // 100
```

Spiegazione:

- **reduce() riduce un array a un singolo valore, partendo da un valore iniziale (in questo caso 0).**
- **La funzione prende due parametri:**
  - **totale → l'accumulatore che conserva i risultati parziali**
  - **numero → l'elemento corrente dell'array**
- **Qui serve per sommare tutti i numeri dell'array.**


Buon Davante a tutti