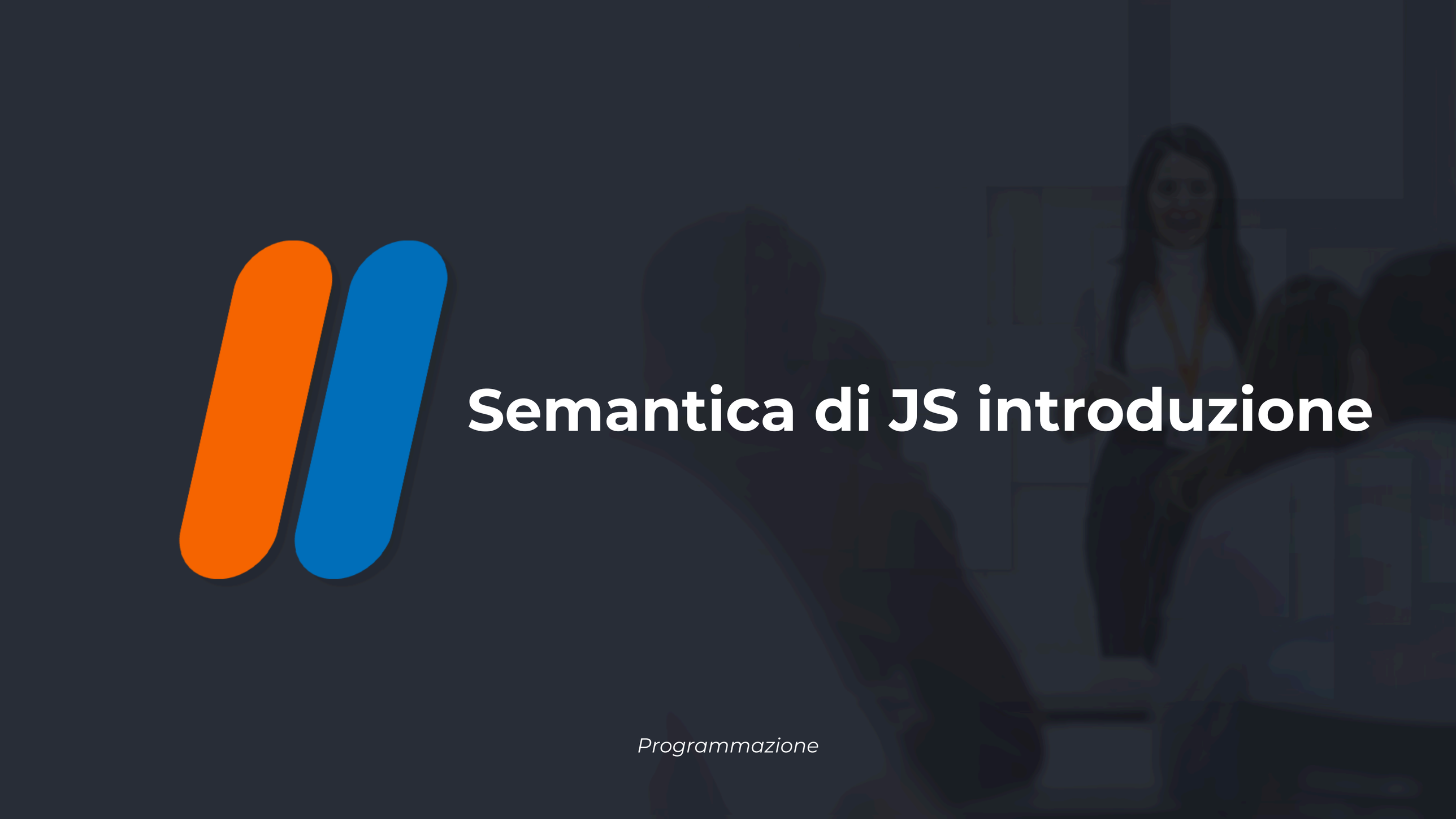


Semantica di JS introduzione

Programmazione

Introduzione alla Semantica di JavaScript

La semantica in un linguaggio di programmazione descrive il significato delle istruzioni e il comportamento del programma durante l'esecuzione.

In JavaScript, linguaggio dinamico e multi-paradigma, la semantica è influenzata da elementi come l'ordine di valutazione, l'ambiente lessicale, e la gestione dinamica dei tipi.

Comprendere la semantica è essenziale per evitare comportamenti inattesi e scrivere codice robusto e manutenibile.

Uno degli aspetti più rilevanti della semantica di JavaScript è la scoping chain, ovvero come e dove le variabili vengono risolte, assieme al contesto di esecuzione (this) che può cambiare dinamicamente. Inoltre, JavaScript ha una semantica basata su event loop, il che implica una gestione asincrona delle operazioni e una distinzione tra stack sincrono e coda di callback asincroni.

In JavaScript, le variabili non sono vincolate a un tipo specifico al momento della dichiarazione: possono contenere valori di qualsiasi tipo e cambiarlo durante l'esecuzione.

Questo comportamento è possibile grazie alla tipizzazione dinamica, che consente di assegnare un numero, poi una stringa, poi un oggetto alla stessa variabile senza errori.

Tuttavia, per gestire i valori in modo efficace, è importante conoscere e distinguere i diversi tipi di dato che JavaScript mette a disposizione.

I tipi in JavaScript si dividono in due categorie principali: tipi primitivi (valori immutabili e copiati per valore) e tipi complessi (come oggetti e array, copiati per riferimento).

Ogni tipo ha un comportamento semantico specifico che influenza come vengono memorizzati e manipolati i dati nel programma, comprendere questi concetti è fondamentale per evitare bug legati alla mutabilità, al confronto tra valori e all'uso della memoria.

In JavaScript, una variabile è un contenitore simbolico usato per memorizzare e gestire dati durante l'esecuzione di un programma.

Le variabili si dichiarano con le parole chiave var, let o const, che determinano il comportamento dello scope (visibilità) e della mutabilità del valore. var è la forma più antica e ha scope di funzione, mentre let e const, introdotti con ES6, hanno scope di blocco e offrono una gestione più prevedibile e sicura del codice, la differenza tra let e const riguarda la riassegnazione: let permette di cambiare il valore della variabile, const no (anche se gli oggetti referenziati da una const possono essere modificati internamente).

La flessibilità di JavaScript permette alle variabili di contenere valori di qualsiasi tipo, e di cambiarlo nel corso del tempo, tuttavia, questa libertà comporta anche dei rischi, come errori di tipo o comportamenti inattesi, motivo per cui è importante dichiarare le variabili in modo esplicito e usarle con coerenza.

- Tipi primitivi:

- String
- Number
- Boolean
- Undefined
- Null
- Symbol
- BigInt

- Tipi complessi (oggetti):

- Object
- Array
- Function

Esempio:

```
1. let nome = "Luca";    // tipo String
2. const eta = 25;       // tipo Number, non può essere riassegnata
3. var attivo = true;    // tipo Boolean
```

In questo esempio:

- **nome può essere modificato in seguito (es. nome = "Marco"),**
- **eta è costante e non può essere riassegnata,**
- **attivo è visibile nell'intera funzione in cui è dichiarata, anche se definita dentro un blocco.**

Tipizzazione in JavaScript

JavaScript è un linguaggio debolmente e dinamicamente tipizzato. Questo significa che:

- **I tipi delle variabili possono cambiare durante l'esecuzione.**
- **Le conversioni di tipo possono avvenire implicitamente (coercizione).**

Esempio:

```
1. let a = 10;
2. a = "dieci"; // Nessun errore, ora a è una stringa
3.
4. console.log("5" * 2); // 10 -> "5" viene convertito in numero
```

Questa flessibilità è utile per la rapidità nello sviluppo, ma può causare errori difficili da individuare.

Per rendere la tipizzazione più sicura, si usa spesso TypeScript, un superset di JavaScript con tipizzazione statica.

Caratteristiche Semantiche di JavaScript

- **Hoisting**

Le dichiarazioni di variabili (var) e funzioni vengono "spostate" in alto nel contesto di esecuzione.

1. **console.log(a); // undefined**
2. **var a = 5;**

La variabile a viene dichiarata ma inizializzata solo dopo, quindi all’inizio è undefined.

Caratteristiche Semantiche di JavaScript

- **Scope lessicale**

Lo scope delle variabili è determinato dalla posizione nel codice, non dall'esecuzione.

```
1. function outer() {
2.   let x = 10;
3.   function inner() {
4.     console.log(x); // 10
5.   }
6.   inner();
7. }
8. outer();
```

inner() ha accesso alla variabile x definita in outer() grazie allo scope lessicale.

Caratteristiche Semantiche di JavaScript

- **Chiusure (Closures)**

Una funzione può "ricordare" le variabili del contesto in cui è stata creata.

```
function creaContatore() {
  let count = 0;
  return function() {
    count++;
    return count;
  };
}

const contatore = creaContatore();
contatore(); // 1
contatore(); // 2
```

La funzione interna mantiene l’accesso a count anche dopo che creaContatore è terminata.

Caratteristiche Semantiche di JavaScript

- **Contesto dinamico (this)**

Il valore di this dipende da come viene chiamata una funzione.

```
1. const obj = {
2.   nome: "Mario",
3.   saluta: function() {
4.     console.log("Ciao da", this.nome);
5.   }
6. };
7. obj.saluta(); // Ciao da Mario
```

Quando saluta è chiamato come metodo dell’oggetto, this fa riferimento all’oggetto stesso.

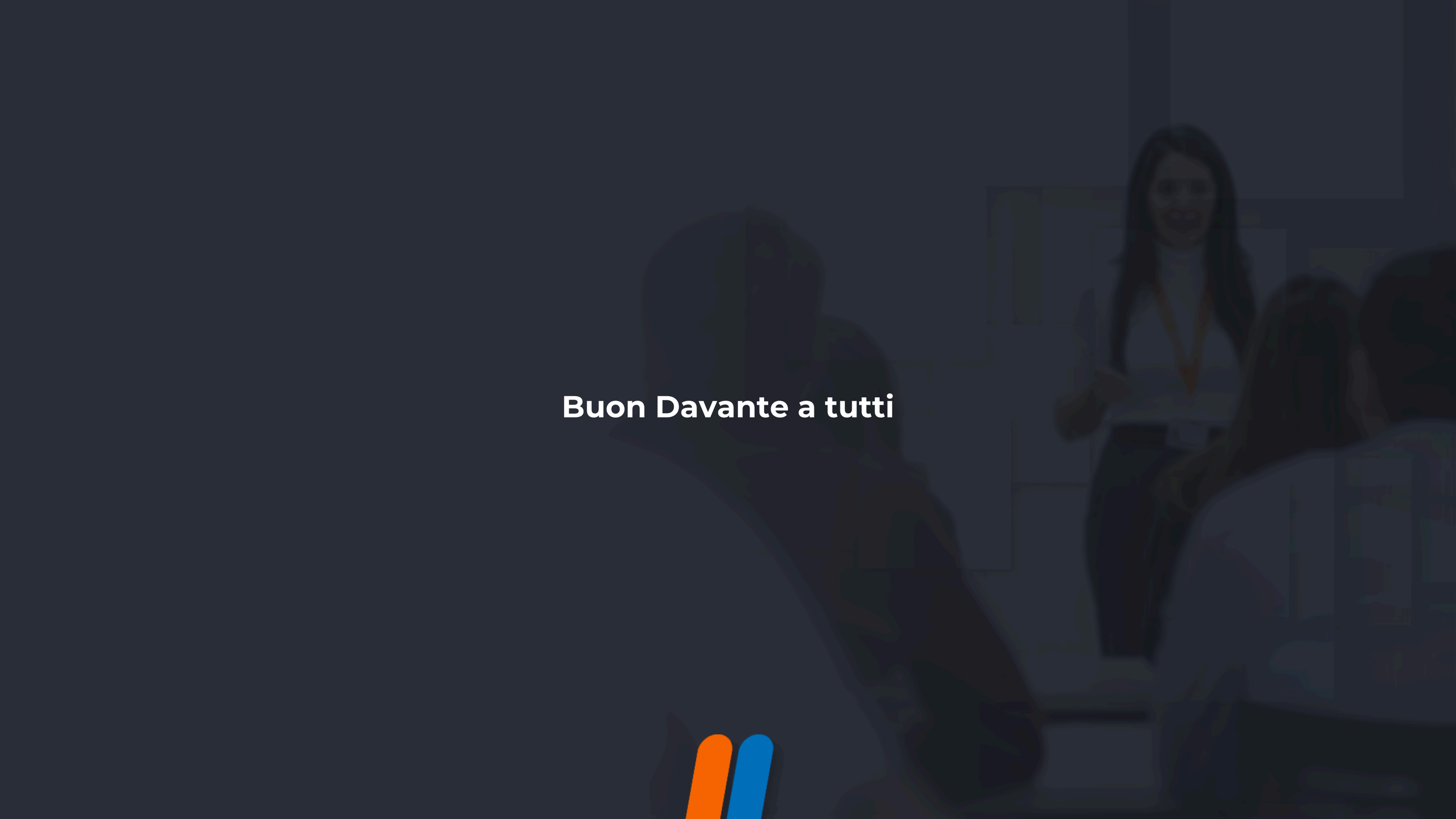
Caratteristiche Semantiche di JavaScript

- **Coercizione implicita**

JavaScript converte automaticamente tipi diversi quando necessario.

1. **console.log("5" + 1); // "51"**
2. **console.log("5" - 1); // 4**

+ concatena stringhe, - forza la conversione in numero.


Buon Davante a tutti