11/07/2025

Metodi Fondamentali in JavaScript

COSA SONO I METODI

UN metodo è una funzione associata a un oggetto.

ex:

```
let persona = {
  nome: "Luca",
  saluta: function() {
    console.log("Ciao, sono " + this.nome);
  }
};

persona.saluta(); // metodo
```

saluta --> metodo di 'persona'
this.nome fa riferimento all'attributo nome dell'oggetto.

Il Metodo push

Il metodo push è un metodo mutativo che opera sugli array. Esso consente di aggiungere uno o più elementi alla fine dell'array su cui viene invocato.
L'invocazione di push comporta una modifica dello stato interno dell'array esistente, estendendone la lunghezza.

array.push(elem1, elem2, ...);

Parametri: uno o più elementi da aggiungere in coda all'array.
Ritorno: restituisce la nuova lunghezza dell'array dopo l'aggiunta.

Esempi

```
let train = [element1, element2];
let newLenght = train.push(element3, element4);
```

In questo esempio, element3 ed element4 vengono appesi in coda all'array train.
La variabile newLenght conterrà la nuova dimensione complessiva dell'array risultante.

Proprietà di Mutabilità

Essendo un metodo mutativo, push modifica direttamente l'array originale su cui è invocato.
Tale comportamento può influenzare la leggibilità e la prevedibilità del codice, specialmente in ambienti in cui si predilige la programmazione funzionale.

Principio della Località dei Dati

Il metodo push sfrutta la località d'estensione degli array,
ovvero si limita ad appendere nuovi elementi esclusivamente in coda.
Questo comportamento favorisce un accesso più efficiente in termini di memoria e prestazioni,
in quanto non richiede spostamenti degli elementi già presenti nell'array.

Il Metodo concat

Descrizione Generale

Il metodo concat è un metodo non mutativo che serve a creare un nuovo array,
risultante dalla concatenazione dell'array chiamante con uno o più argomenti passati.
A differenza di push, non modifica l'array originale, ma restituisce un nuovo array indipendente.

array1.concat(array2, arg1, ...);

Parametri:
Se l'argomento è un array, i suoi elementi vengono espansi ed inseriti nel nuovo array.
Se l'argomento è un valore, viene inserito come singolo elemento.
Ritorno: un nuovo array che rappresenta la concatenazione.

Esempio:

```
let convoy1 = ['A', 'B'];
let convoy2 = ['C', 'D'];
let newTrain = convoy1.concat(convoy2, 'E');
```

In questo caso, newTrain conterrà l'array ['A', 'B', 'C', 'D', 'E'],
risultante dalla concatenazione degli elementi di convoy1, convoy2 e del valore singolo 'E'.

Proprietà di Immutabilità

Essendo non mutativo, concat non altera il contenuto né la struttura degli array originali.
Questo approccio è particolarmente utile per evitare effetti collaterali non desiderati e si inserisce nel paradigma della programmazione funzionale.

Purezza delle Funzioni

Il metodo concat si avvicina al paradigma funzionale, poiché non altera dati esterni.
Tale purezza facilita il reasoning e il testing, in quanto il comportamento della funzione dipende esclusivamente dagli input forniti e non dallo stato esterno.

Mutabilità vs. Immutabilità

Metodo mutativo: modifica lo stato interno dell'array su cui opera (es. push).

Metodo non mutativo: restituisce un nuovo array, lasciando invariati gli array originali (es. concat).

Considerazioni

La distinzione tra mutabilità e immutabilità è centrale in molte discipline della programmazione.
L'uso di metodi mutativi può risultare più performante in certi contesti, ma rende più difficile il tracciamento dello stato.
Al contrario, i metodi non mutativi favoriscono la scrittura di codice più sicuro, prevedibile e testabile.

Il Metodo includes

Descrizione Generale

includes è un metodo che permette di verificare se un array contiene un certo valore.
Si tratta di una funzionalità estremamente comoda e leggibile, particolarmente utile nei controlli condizionali.

array.includes(valoreCercato)

Parametri: il valore da cercare all'interno dell'array.

Ritorno: true se il valore è presente, false in caso contrario.
Esempio
let unique = [1, 2, 3, 5, 4];
unique.includes(3); // true
In questo esempio, l'array unique include il valore 3, pertanto il metodo restituisce true.
Funzionamento Interno
Il metodo esegue una ricerca sequenziale attraverso l'array,
confrontando ciascun elemento con il valore cercato. Il confronto avviene utilizzando l'operatore di uguaglianza (==).
La ricerca si arresta non appena viene trovata una corrispondenza.

Metodo indexOf

Descrizione Generale
Il metodo indexOf consente di cercare un elemento all'interno di un array e restituisce l'indice della prima occorrenza trovata.
Se l'elemento non è presente, restituisce -1.

array.indexOf(elementoDaCercare)

Parametri: l'elemento da cercare.
Ritorno: indice della prima occorrenza oppure -1 se non trovato.
Considerazioni Tecniche
È un metodo case sensitive, ovvero distingue tra maiuscole e minuscole.
Funziona con numeri, stringhe e altri tipi di dati comparabili.

map()

map() è una funzione che `applica un' altra funzione a ogni elemento` di un array e restituisce un nuovo array contenente i risultati.
Non muta l'array originale

- **Restituisce un nuovo array**
- **Applica una funzione a ciascun elemento**, passando tre parametri:

1. il valore corrente,
2. l'indice corrente,
3. l'array stesso.

Ex:
let numbers = [1, 2, 3];
let doubled = numbers.map(n => n * 2);
// Risultato: [2, 4, 6]

In questo esempio:

- La funzione n => n * 2 è **pura**: restituisce sempre il doppio del numero fornito, senza effetti collaterali.
- Il metodo map() **non altera** l'array originale numbers.
- Il risultato doubled è un nuovo array, derivato applicando una trasformazione a ogni elemento.

Il metodo map() `non garantisce automaticamente` **la purezza:** `dipende dalla funzione` **che gli viene passata**

Filter()

Restituisce un nuovo array contenente **solo gli elementi che soddisfano una condizione** (funzione di test).

Proprietà:

- **Non mutativo**
- Funziona perfettamente con **funzioni pure**
- Restituisce un sottoinsieme dell'array originale

ex:

let nums = [1, 2, 3, 4, 5];

let even = nums.filter(n => n % 2 === 0); // [2, 4]

reduce()

Riduce l'array a un singolo valore, applicando una funzione accumulatrice su ciascun elemento, da sinistra a destra.

Proprietà:

- Non mutativo
- Estremamente potente: può simulare map, filter, find, sum, groupBy
- La funzione passata deve essere pura per garantire prevedibilità

ex:

let nums = [1, 2, 3, 4];

let sum = nums.reduce((acc, n) => acc + n, 0); // 10

Il Metodo toUpperCase()

Uno dei metodi fondamentali delle stringhe è toUpperCase(), che converte tutti i caratteri di una stringa in lettere maiuscole. Questo metodo non modifica la stringa originale, ma restituisce una nuova stringa trasformata.

Esempio pratico:

```
let myString = "I'm really hungry for some";
console.log(myString);
let myUpperString = myString.toUpperCase();
console.log(myUpperString);
```

In questo esempio:

- Si crea una variabile myString a cui viene assegnata la stringa "I'm really hungry for some".
- Viene utilizzato console.log() per stampare la stringa originale.
- Si crea una seconda variabile myUpperString, a cui viene assegnata la versione maiuscola della stringa originale.
- Infine, anche questa stringa trasformata viene stampata nel console log.

Metodo .substring(startIndex, endIndex)

Definizione:

Il metodo .substring() restituisce una sottostringa compresa tra due indici all'interno della stringa originale.

string.substring(startIndex, endIndex)

startIndex (obbligatorio): la posizione **iniziale** da cui partire per l'estrazione.

- endIndex (facoltativo): la posizione **non inclusa** di arrivo. Se omesso, la sottostringa prosegue fino alla fine della stringa.

Metodo .search(regexp | string)

Definizione:
Restituisce l'indice della prima occorrenza di una sottostringa o di un'espressione regolare all'interno della stringa su cui viene chiamato. Se non trova corrispondenze, restituisce -1.

Sintassi:

string.search(regexp | string)

Parametri:

• regexp | string (obbligatorio): la sottostringa o l'espressione regolare da cercare nella stringa.

```
let str = "I'm really hungry";

let index = str.search("really");

console.log(index); // Output: 5
```

metodo .split()

Il metodo . `split()` è un metodo dell'oggetto **String**. Consente di suddividere una stringa in più parti, restituendo un **array** in cui ogni elemento è una sottostringa separata da un delimitatore.

Sintassi:

```
string.split(separator)
```

separator (obbligatorio): una stringa o espressione regolare che definisce il punto di separazione tra le sottostringhe.

Il metodo **non modifica la stringa originale**, ma restituisce un nuovo array.

- Se il separatore non viene trovato nella stringa, l'intero contenuto viene restituito come unico elemento dell'array.
- Se il separatore è una stringa vuota `""`, la stringa viene divisa **in caratteri singoli**.

Il metodo . `reverse()`

Il metodo . `reverse()` è un metodo dell'oggetto **Array**. Permette di **invertire l'ordine** degli elementi presenti nell'array.

Sintassi:

```
array.reverse()
```

Restituisce **lo stesso array** su cui è stato chiamato, ma con gli elementi **in ordine inverso**.

```
let array = ["a", "b", "c"];
```

array.reverse();

console.log(array); // Output: ["c", "b", "a"]

Il metodo . join()

Il metodo . `join()` è un metodo dell'oggetto **Array**. Consente di unire tutti gli elementi di un array in una **singola stringa**, separando ogni elemento con un delimitatore specificato.

Sintassi:

```
array.join(separator)
```

separator (facoltativo): una stringa da inserire tra ciascun elemento dell'array.
Se omesso, il separatore predefinito è la **virgola** (,).

let array = ["a", "b", "c"];

console.log(array.join("-")); // Output: "a-b-c"