Explain indexOf method

explain indexOf

Here is the **simplest, beginner-friendly explanation of `indexOf` in JavaScript**, with examples so you never forget it again.

⭐ What is `indexOf`?

`indexOf` is a JavaScript method used to **find the position (index)** of a value inside:

- an **array**, or
- a **string**

It always returns a **number**.

⭐ How does it work?

👉 If the value is found

It returns the **index (position)** where it first appears.

👉 If the value is NOT found

It returns **-1**

Simple rule:

> **Found → index**
> **Not found → -1**

🔹 1. `indexOf` with Arrays

Example:

```
js

const arr = [10, 20, 30, 20];

console.log(arr.indexOf(20));   // 1
console.log(arr.indexOf(30));   // 2
console.log(arr.indexOf(50));   // -1 (not found)
```

📌 It always returns the **first occurrence**.

🔹 Optional second parameter (start index)

You can tell **`indexOf`** from where to start searching.

```
js

const arr = [10, 20, 30, 20];

console.log(arr.indexOf(20, 2));   // 3 (starts search from index 2)
```

🔹 2. `indexOf` with Strings

It works exactly the same.

```
js

const str = "hello world";

console.log(str.indexOf("o"));       // 4
console.log(str.indexOf("world"));   // 6
console.log(str.indexOf("z"));       // -1
```

⭐ Why is `indexOf` useful?

👉 To check if a value exists

```
js

if (arr.indexOf(20) !== -1) {
  console.log("20 exists!");
}
```

👉 To avoid duplicates

```
js

if (arr.indexOf(value) === -1) {
  arr.push(value);
}
```

🔥 Very simple way to remember `indexOf`

✔️ **indexOf = "Position of?"**
✔️ If found → gives position
✔️ If not found → -1

If you want, I can also explain:

- `lastIndexOf`
- `includes` (better than indexOf)
- difference between `indexOf` and `findIndex`
  Just tell me!

const arr = [10, 20, 30, 20];

console.log(arr.indexOf(20)); // 1
console.log(arr.indexOf(30)); // 2
console.log(arr.indexOf(50)); // -1 (not found)

but there are 20 present two times than it returned 1 for first twenty or second twenty and what is for other one?

Great question, Vipul!
Let's break it down **in the simplest way** so you understand it once and never forget.

✅ Why `indexOf(20)` returns 1 even though 20 appears twice?

Because `indexOf` **always returns the FIRST matching index**.

Your array:

```makefile
Index:   0   1   2   3
Value:  10  20  30  20
```

- The first `20` is at index **1**
- The second `20` is at index **3**

So:

```js
arr.indexOf(20)  // returns 1   (FIRST match only)
```

🟦 How to get the index of the second 20?

Use the optional **start index**.

```js
arr.indexOf(20, 2);
```

Explanation:

- `2` means → "Start searching from index 2"
- Starting from index 2, the next `20` is at index 3

So:

```
js

console.log(arr.indexOf(20, 2));  // 3
```

✔️ Final Summary

Search	What it returns
`arr.indexOf(20)`	1 (first 20)
`arr.indexOf(20, 2)`	3 (second 20)
`arr.indexOf(20, 4)`	-1 (no 20 after index 4)

If you want, I can explain:

- How to get **all indexes** of a value
- Difference between `indexOf` and `findIndex`
- Why `includes` is sometimes better

Just tell me!

yes

Great! I'll explain all three in a **very simple, beginner-friendly way**:

✅ 1. How to get ALL indexes of a value in an array

Example array:

```
js

const arr = [10, 20, 30, 20, 40, 20];
```

🔹 Method 1: Using a simple loop

(Easiest to understand)

```js
const indexes = [];

for (let i = 0; i < arr.length; i++) {
  if (arr[i] === 20) {
    indexes.push(i);
  }
}

console.log(indexes);  // [1, 3, 5]
```

🔹 Method 2: Using `map` + `filter`

```js
const indexes = arr
  .map((value, index) => (value === 20 ? index : -1))
  .filter(i => i !== -1);

console.log(indexes); // [1, 3, 5]
```

✅ 2. Difference between `indexOf` and `findIndex`

Feature	indexOf	findIndex
Checks for	Exact value match	Condition (callback)
Returns	First index	First index
Example	**`arr.indexOf(20)`**	**`arr.findIndex(v => v > 15)`**

Example:

```js
const arr = [10, 20, 30];

console.log(arr.indexOf(20));   // 1
console.log(arr.findIndex(num => num > 15)); // 1
```

➡️ **`findIndex`** is more powerful because it uses conditions.

✅ 3. Why `includes` is sometimes better than `indexOf`

✔️ `includes`

Returns **true / false**
(Just checks existence)

```js
arr.includes(20)  // true
arr.includes(50)  // false
```

✔️ `indexOf`

Returns a **number**

```
js

arr.indexOf(20)   // 1
arr.indexOf(50)   // -1
```

When to use what?

If you want to know...	Use
Does this value exist?	`includes`
Where does it exist?	`indexOf`
Find index using a condition	`findIndex`

🎯 Final Summary

✔️ `indexOf` → first index only

✔️ `indexOf(value, start)` → find next occurrences

✔️ To get ALL indexes → use loop or map+filter

✔️ `findIndex` → use condition

✔️ `includes` → true/false only

If you want, I can also create:

- 10 interview questions on **`indexOf`**, **`findIndex`**, and **`includes`**
- Real-world use cases
- Practice problems

Just tell me!