Explain concat method

explainn concat

Here is **the clearest, beginner-level explanation** of `concat()` in JavaScript.

✅ What is `concat()`?

`concat()` is a method used on **arrays** (and strings) to **join/combine** them **without changing the original array**.

Think of it like:
👉 "Make a new array by sticking these arrays/things together."

✅ Syntax

```
js

newArray = array1.concat(array2, array3, ..., value1, value2)
```

🧠 Important: `concat()` does NOT modify the original array

It **returns a NEW array**.

✅ Simple Example

```
js

const a = [1, 2];
const b = [3, 4];

const result = a.concat(b);

console.log(result); // [1, 2, 3, 4]
console.log(a);      // [1, 2]  (original unchanged)
```

🆚 Difference between `concat()` and spread (`...`)

✔️ Both do the same thing

```js
const res1 = [1, 2].concat([3, 4]);
const res2 = [...[1, 2], ...[3, 4]];
```

Both give: `[1, 2, 3, 4]`

❗ But WHY both exist?

1️⃣ concat() existed first (old JavaScript).

2️⃣ Spread operator (...) came later (ES6) as a shorter, cleaner syntax.

🧩 *Main practical differences:*

Feature	concat()	Spread (...)
Readability	Longer	Shorter, cleaner
Copies arrays	Yes	Yes
Adds values	Yes	Yes
Deep copy	❌ No	❌ No (same problem)
Merging many arrays	Can merge multiple	Can merge multiple

Example merging 3 arrays:

```js
const res = a.concat(b, c);
```

With spread:

```js
const res = [...a, ...b, ...c];
```

Both same.

🧨 Important difference

When combining **nested arrays**, concat behaves differently:

Example:

```js
const a = [1, 2];
const b = [[3, 4]];

const r1 = a.concat(b);
const r2 = [...a, ...b];
```

```
console.log(r1); // [1, 2, [3, 4]]
console.log(r2); // [1, 2, [3, 4]]
```

Both same.

So **there is NO big difference** in modern JavaScript.

🔍 What should YOU use? (Practical advice)

Since you're a frontend dev:

👉 **Use spread `...` 99% of the time.**
Cleaner and modern.

Use **`concat()`** only when:

- You want **method chaining**:

```
js

const res = arr.concat(a).concat(b).concat(c);
```

⚠️ Does `concat()` make a deep copy?

❌ No.

Shallow copy only.

```
js

const a = [{name: "Vipul"}];
const b = a.concat();

b[0].name = "Malik";

console.log(a[0].name); // "Malik" → changed
```

Because it copies **references**, not the actual object.

🎁 Tiny summary

Method	Purpose
`concat()`	Join arrays, returns new array, old method but safe
Spread `...`	Modern, short, clean, preferred

If you want, I can also explain **difference between concat and push**, or **concat vs flat**, or give **interview questions** on concat.