Code mutation explanation

Almost right, but let me clarify **precisely** — there's a subtle distinction.

Your code:

```js
const users = [
  { name: "Vipul", age: 31 },
  { name: "Arun", age: 28 }
];

const newUsers = users.map(item => item.isAdult = true);
```

Step-by-step

1️⃣ **`users.map(...)` creates a new array** → this is `newUsers`. ✅

2️⃣ Inside `map` callback:

```js
item.isAdult = true
```

- This **mutates the original object** inside `users`.
- **It does NOT create a new object.**
- So now each object in `users` has `isAdult: true`.

3️⃣ The **return value of the callback** is **the value of the assignment**, i.e., `true`.

- That's what `map` puts into `newUsers`.

✅ So after this:

```js
console.log(users);
/* [
  { name: "Vipul", age: 31, isAdult: true },
  { name: "Arun", age: 28, isAdult: true }
] */

console.log(newUsers);
/* [true, true] */
```

Key Takeaways

- `map()` → returns a **new array**.
- Objects inside the array → still **same references**. Mutating them affects the original array.

- **`item.isAdult = true`** → returns **`true`**, that's why **`newUsers`** is **`[true, true]`**.

If you want **`newUsers`** to be an array of **objects with isAdult**, without mutating **`users`**, you'd do:

```js
const newUsers = users.map(item => ({ ...item, isAdult: true }));
```

Then:

```js
console.log(users); // original unchanged
console.log(newUsers); // new array with added key
```

I can also make a **diagram showing references and new array** — it really helps visualize why mutation happens.
Do you want me to do that?