Code mutation issue

```
const user = [
  { name: "Vipul", age: 31 },
  { name: "Arun", age: 28 },
  { name: "Mohan", age: 17}
 ];


 const newUser = user?.map((item) => {
   if(item?.age >= 18) {
    return item.isAdult = true;
   } else {
     return item;
   }
 })

 console.log("user:", user);
 console.log("newUser:", newUser);
```

Let's break this down **clearly and slowly** so you understand exactly what is happening.

✅ Your Code

js

```js
const user = [
  { name: "Vipul", age: 31 },
  { name: "Arun", age: 28 },
  { name: "Mohan", age: 17}
];

const newUser = user?.map((item) => {
  if(item?.age >= 18) {
    return item.isAdult = true;
  } else {
    return item;
  }
})

console.log("user:", user);
console.log("newUser:", newUser);
```

❌ Problem 1: You are MUTATING the original array

`item.isAdult = true` adds a new key on the original object inside `user`.

So after running this:

`user` becomes:

```js
[
  { name: "Vipul", age: 31, isAdult: true },
  { name: "Arun", age: 28, isAdult: true },
  { name: "Mohan", age: 17 }
]
```

❌ Problem 2: Your map returns WRONG VALUES

Look at this part:

```js
return item.isAdult = true;
```

🔍 What does assignment return in JS?

Assignment (=) returns the **assigned value**, not the object.

So **`item.isAdult = true`** returns:

```arduino
true
```

Therefore your **`newUser`** array becomes:

```js
[ true, true, { name: "Mohan", age: 17 } ]
```

✔️ Two elements are **`true`**
✔️ One element is an object

This is **not what you want**.

⭐ OUTPUT

👉 `user` (mutated):

```yaml
[
  { name: 'Vipul', age: 31, isAdult: true },
  { name: 'Arun', age: 28, isAdult: true },
  { name: 'Mohan', age: 17 }
]
```

👉 `newUser`:

```yaml
[ true, true, { name: 'Mohan', age: 17 } ]
```

✅ Correct Way (No mutation + Proper Return)

If you want a new array with new values:

```js
const newUser = user.map(item => {
  return {
    ...item,
    isAdult: item.age >= 18
  };
});
```

Output:

```js
[
  { name: "Vipul", age: 31, isAdult: true },
  { name: "Arun", age: 28, isAdult: true },
  { name: "Mohan", age: 17, isAdult: false }
]
```

✅ Correct Way (Mutate but don’t return wrong values)

If you WANT to mutate (not recommended):

```js
const newUser = user.map(item => {
  item.isAdult = item.age >= 18;
  return item;
});
```

Output:

`newUser` and **`user`** will be same.

🧠 Final Summary

Concept	What happened?
Mutation	✔️ Your original array got modified
Return value	❌ You returned **`true`**, not the object
Best practice	❗ Never mutate inside map; always return a new object

---

If you want, I can explain with diagrams also.

What does assignment return in JS?

Assignment (=) returns the assigned value, not the object.

explain this unable to undersatnd this part

Let’s explain this in **the simplest, most beginner-friendly way**, with **very clear examples** so you fully understand.

🌟 Key Idea

When you do assignment (`=`) in JavaScript:

🔥 It returns the value you assigned — NOT the object.

💡 Example 1 — Easiest Example

Code:

```js
let x;
let y = (x = 10);

console.log("x =", x);
console.log("y =", y);
```

Output:

```ini
x = 10
y = 10
```

Why?

- `(x = 10)` **assigns** 10 to `x`
- And the **result of the assignment expression** is **also 10**
- So `y` receives the value `10`

🧠 Why doesn’t it return the variable/object?

Because in JavaScript:

✔️ Assignment is an expression

✔️ An expression always evaluates to a VALUE

✔️ The value returned is the one you assigned

So:

```
js

item.isAdult = true
```

returns:

```
arduino

true
```

Not the updated object.

🌟 Example 2 — Your exact case

Code:

```
js

const user = { name: "Vipul" };

const result = (user.isAdult = true);

console.log("result:", result);
console.log("user:", user);
```

Output:

```
yaml

result: true
user: { name: "Vipul", isAdult: true }
```

Explanation:

- **`user.isAdult = true`**
  → adds **`isAdult: true`** inside the object
  → **but still RETURNS `true`**, not the object.

🌟 Example 3 — Map example

```js
const arr = [{ age: 20 }, { age: 17 }];
const result = arr.map(item => item.isAdult = true);
console.log(result);
```

Output:

```arduino
[ true, true ]
```

Because:

- first item → `item.isAdult = true` → returns `true`
- second item → `item.isAdult = true` → returns `true`

But the original array becomes:

```yaml
[{ age: 20, isAdult: true }, { age: 17, isAdult: true }]
```

🌟 Visual Diagram (Very Simple)

```vbnet
item.isAdult = true
       |
       | performs assignment → adds isAdult property
       |
returns the value TRUE   (not the object)
```

✔️ Final Summary (Super Simple)

When you write:

```ini
a = b
```

Three things happen:

1️⃣ Assign `b` to `a`
2️⃣ Update the variable/object
3️⃣ **Return the value `b`**

It never returns `a`.

If you want, I can also explain with internal JS steps or memory diagrams — just tell me!

> and difference between this isAdult: true vs isAdult = true

Let's explain this **very clearly** because this is one of the most confusing things for beginners.

✅ Difference Between:

1️⃣ `isAdult: true`

2️⃣ `isAdult = true`

They look similar...
But they are **completely different things** in JavaScript.

🟦 1️⃣ `isAdult: true` — Used INSIDE an OBJECT

This is **object property creation**.

Example:

```js
const user = {
  name: "Vipul",
  isAdult: true
};
```

Here:

- `isAdult` is a **key/property name**
- `true` is the **value**
- It does NOT execute any code
- It does NOT return anything
- It is only used inside an object `{ ... }`

✔️ Pure object property
✔️ No mutation outside
✔️ No assignment operator is used
✔️ No return value

🟥 2️⃣ `isAdult = true` — Assignment Expression

This is **assignment**, not object creation.

It **executes at runtime** and:

✔️ Assigns value

✔️ Mutates the object

✔️ And RETURNS the value assigned

Example:

```
js

item.isAdult = true;
```

What happens?

1. Adds/Updates the property **`isAdult`**
2. Mutates the original object
3. Expression returns **`true`**

🧠 Side-by-Side Comparison

Code	Meaning	Used in	Mutates?	Returns something?
`isAdult: true`	Create key-value pair	Object literal **`{}`**	❌ No	❌ No
`isAdult = true`	Assign value	Anywhere in JS	✔️ Yes (mutates)	✔️ Yes → returns **`true`**

🧪 Example to make it 100% clear

👉 Using `isAdult: true`

```
js

const user = { name: "Vipul", isAdult: true };
```

Output:

```
yaml

{ name: "Vipul", isAdult: true }
```

Nothing is mutated. No value returned.

👉 Using `isAdult = true`

```
js

const user = { name: "Vipul" };

const result = (user.isAdult = true);

console.log(result); // ?
console.log(user);   // ?
```

Output:

```
pgsql

result → true         (assignment returns the assigned value)
user → { name: "Vipul", isAdult: true }
```

🎯 SUPER SIMPLE SUMMARY

`isAdult: true`

➡️ Only used inside `{ }`
➡️ Just defines a property
➡️ Does NOT run code
➡️ Does NOT return a value

`isAdult = true`

➡️ Assignment statement
➡️ Mutates object
➡️ **Returns the assigned value** (`true`)
➡️ Can be used anywhere

💥 Final Example in Map to show difference

❌ Wrong (uses assignment)

```
js

const result = users.map(u => u.isAdult = true);
console.log(result); // [true, true, true]
```

✔️ Right (uses object literal)

```
js

const result = users.map(u => ({ ...u, isAdult: true }));
console.log(result);
// returns actual objects
```

If you want, I can create a **diagram** that shows how JS treats both internally.