In **Angular + RxJS**, when working with a `BehaviorSubject` that holds an **object**, the best practice is **not** to mutate the object directly and then call `.next()` with the same reference.

Instead, you should **work on an immutable copy** and emit that new object.

Why?

- **Change detection**:
  Angular's change detection (especially with `OnPush`) relies on **reference changes**.
  If you mutate the same object and call `.next()` with it, subscribers may not detect the change because the reference is the same.
- **Predictability**:
  Immutable updates make debugging easier — you always know that a new emission means a new state.
- **Avoid side effects**:
  Direct mutation can cause unexpected behavior if other parts of the app still hold a reference to the old object.

Best Practice Example

```
import { BehaviorSubject } from 'rxjs';

interface User {
  id: number;
  name: string;
  age: number;
}

class UserService {
  private userSubject = new BehaviorSubject<User>({ id: 1, name:
'Alice', age: 25 });
  user$ = this.userSubject.asObservable();

  updateUserName(newName: string) {
    const currentUser = this.userSubject.value;
    // Create a new object (immutable update)
    const updatedUser = { ...currentUser, name: newName };
```

```
    this.userSubject.next(updatedUser);
  }

  incrementAge() {
    const currentUser = this.userSubject.value;
    this.userSubject.next({ ...currentUser, age: currentUser.age + 1
});
  }
}
```

Bad Practice (Mutating in place)

```
updateUserName(newName: string) {
  const currentUser = this.userSubject.value;
  currentUser.name = newName; // ❌ Mutating the same object
  this.userSubject.next(currentUser); // Reference is the same
}
```

- This can cause **no UI update** in `OnPush` components.
- Other subscribers holding the same reference will see changes **before** `.next()` is even called.

Extra Tip

If your state is complex, consider:

- `immer` for immutable updates without deep cloning manually.
- **State management libraries** like NgRx, Akita, or NGXS for predictable state handling.

✅ **Rule of thumb**:
Always emit a new object/array reference from a `BehaviorSubject` when updating state.

If you want, I can show you a **deeply nested object update pattern** for `BehaviorSubject` that avoids full deep cloning but still keeps immutability.
Do you want me to prepare that?