Deep Copy
Shallow copy

Follow on

@Duvvuru Kishore

Understanding the difference between shallow and deep copies is crucial for effective data manipulation in JavaScript. Here’s a straightforward guide to help you grasp these concepts:

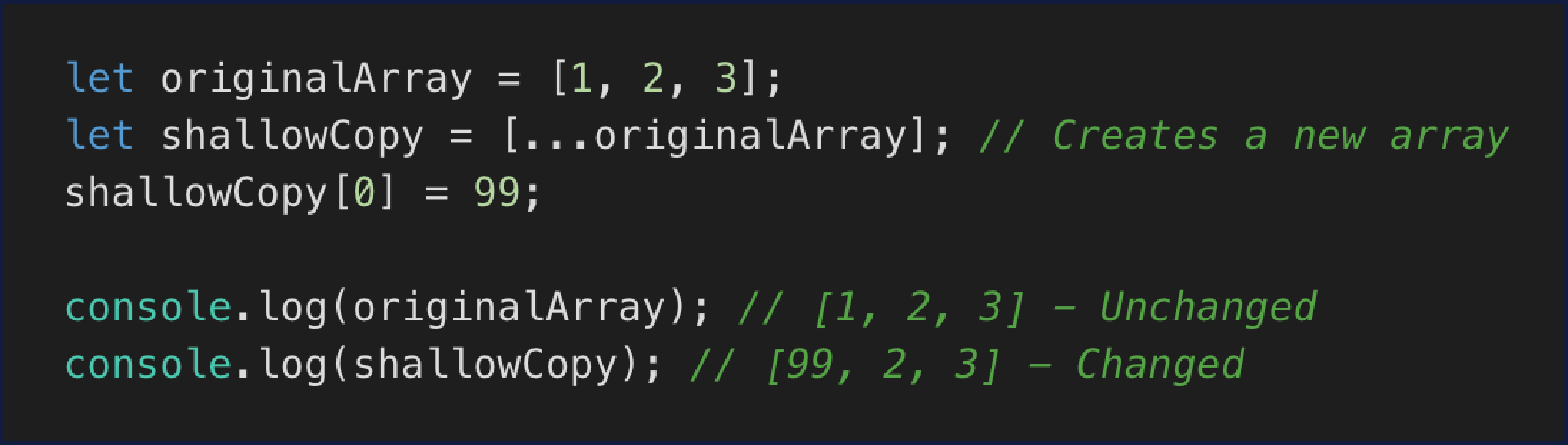
Shallow Copy:

Definition: A shallow copy duplicates the immediate values of an object or array.

However, if the original contains nested objects or arrays, the references to these nested elements are copied, not the objects themselves. This means changes to nested objects in the copied version will affect the original.

Use Case: Use shallow copies for simple structures where you don't have nested objects or arrays.

Example 1: Array of Primitives



```javascript
let originalArray = [1, 2, 3];
let shallowCopy = [...originalArray]; // Creates a new array
shallowCopy[0] = 99;

console.log(originalArray); // [1, 2, 3] – Unchanged
console.log(shallowCopy); // [99, 2, 3] – Changed
```

Explanation:

The shallow copy of an array with primitive values creates a completely new array. Changes to shallowCopy don't affect originalArray.

Methods for shallow copy:

Object.assign(), spread syntax {...}, Array.from()

Example 2: Array of Objects

```javascript
const original = [{ name: "Alice" }, { name: "Bob" }];
const shallowCopy = [...original]; // Copies references to the objects
shallowCopy[0].name = "Charlie";

console.log(original[0].name); // "Charlie"
```

Explanation:

Here, original is an array of objects. The shallow copy shallowCopy contains references to the same objects as original. Changing a property in the shallow copy affects the original object because the nested objects are shared.

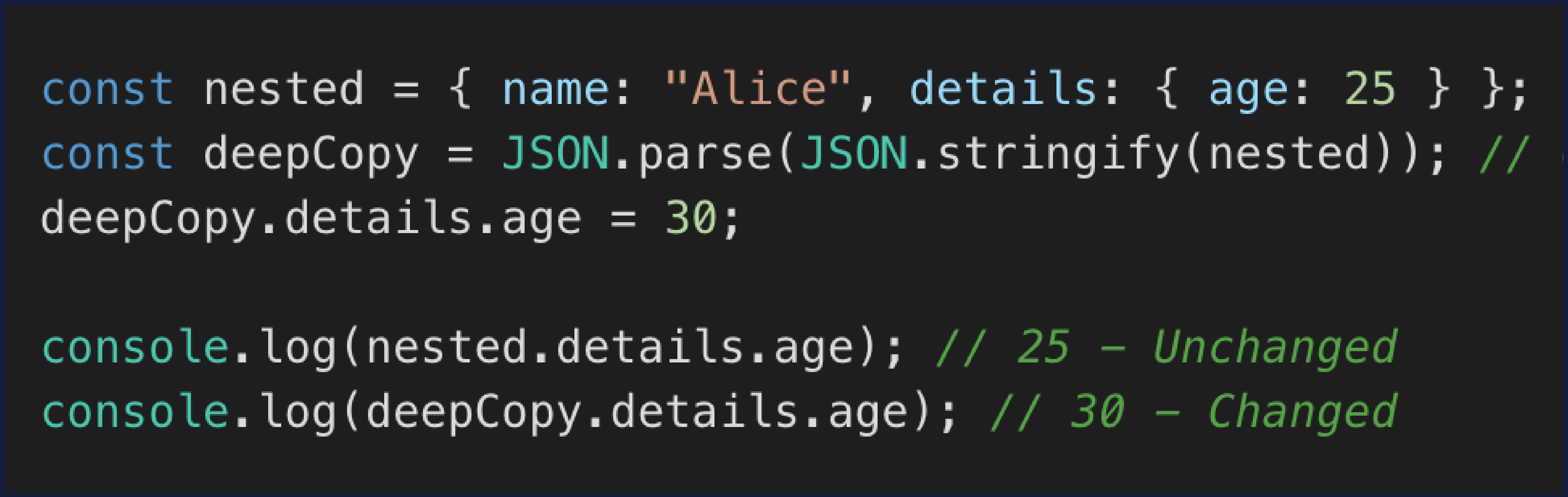
Deep Copy

Definition: A deep copy duplicates all levels of an object or array, creating entirely independent copies of all nested objects and arrays. Changes in the deep copy do not affect the original.

Use Case:

Use deep copies for complex structures with nested objects or arrays, where you want to avoid accidental changes to the original data.

Example:



```javascript
const nested = { name: "Alice", details: { age: 25 } };
const deepCopy = JSON.parse(JSON.stringify(nested)); //
deepCopy.details.age = 30;

console.log(nested.details.age); // 25 - Unchanged
console.log(deepCopy.details.age); // 30 - Changed
```

Explanation:

The JSON.parse(JSON.stringify(nested)) method creates a deep copy of the nested object. **Changes to the deepCopy do not affect the nested object**, as they are completely independent.

Methods: JSON.parse(JSON.stringify()), structuredClone(), Lodash's _.cloneDeep().

Key Points

Shallow Copies:

Efficient for simple, non-nested structures. Be cautious with nested objects since changes will reflect in both the original and copied versions.

Deep Copies:

Necessary for complex, nested structures to ensure independence between the original and the copy. However, they may be more computationally intensive.

Understanding the difference between these two types of copies is crucial for effective data manipulation and avoiding unintended side effects in your JavaScript applications.

Follow on in

@Duvvuru Kishore