Answersheet

Name: Arashad Ahamad

No. of questions attempted: 6

Submitted at: 20 Apr, 2025 6:18 PM

Assignment: Right Way to Copy Objects and Arrays | Deep Vs Shallow Copy |

Total no. of questions: 6

Total marks: 18

Result: Not reviewed yet

Sl. No.	Question	Actions
1	**How do you create a shallow copy of a portion of an array without modifying the original array in JavaScript?** 1 mark 1. By using the splice() method 2. By using the slice() method 3. By using the concat() method 4. By using the copyWithin() method	Correct answer 1 mark
2	**What are the differences between deep copy and shallow copy when copying arrays in JavaScript, and why is it important to understand these differences?** 5 marks Definition: Shallow Copy: A shallow copy creates a new array or object, but it only copies the top-level elements. If the array or object contains other nested arrays or objects, it doesn't copy them deeply; instead, it copies their references (memory addresses). Deep Copy: A deep copy creates a completely independent clone of the original array or object. It copies all levels of nested arrays or objects, so there are no shared references between the original and the copied data. Handling Nested Data: Shallow Copy: It only copies references for nested arrays or objects. So, if you change the nested data in the copied array or object, the	Marks 0

changes will reflect in the original one. Deep Copy: It fully copies nested arrays or objects. Changing the nested data in the copy will not affect the original array or object. Memory Address: Shallow Copy: Copies only the top-level structure and shares the references of nested objects or arrays, meaning both the original and copied array/object point to the same memory location for nested data. Deep Copy: Creates entirely new instances of all objects and arrays at every level, so the original and the copy don't share any memory addresses. Methods: Shallow Copy: Can be done using methods like slice(), spread operator (..., Object.assign() for objects, etc. Deep Copy: Achieved using methods like JSON.parse(JSON.stringify()), structuredClone() (modern browsers), or libraries like Lodash's cloneDeep(). Performance: Shallow Copy: More efficient as it only copies references and does not need to recursively copy nested data. Deep Copy: Slower, especially for large objects/arrays, because it requires recursively copying every level of data. Why Understanding These Differences is Important: Data Integrity: Understanding the difference helps ensure data integrity. A shallow copy might lead to unintended changes in the original data when nested objects or arrays are modified. Memory Management: A deep copy creates a fully independent clone, which can be useful when you need to work with data that should not be affected by changes to the original object/array. Performance: For performance optimization, shallow copying might be preferred when you don't need to worry about modifying nested structures, while deep copying is used when you need to protect the original data from changes in its nested objects or arrays.

3

Step 1: Create an object name ⇒ user like below.

5 marks

Marks

0

```
const user = {
  name: "Amit",
  address: {
      city: "Delhi",
      pincode: 110001
```

}

}

Step 2: Now print the user's name in the console.

Step 3: Create a variable user2 and assign user to it.

const user2 = user

Now is this a Shallow Copy or Deep Copy?

const user3 = { name: 'Amit', address: { city: 'Delhi', pincode: 110001, }, } const user4 = user3; // Reference copy (not even shallow) const user5 = { ...user3 }; // Shallow copy const user6 = structuredClone(user3); // Deep copy console.log(user3);

4

So, the answer to the question asked in the previous question is it is neither a shallow copy nor a deep copy. It is just referencing the *user* variable's address.

5 marks

Marks

0

⇒ Check the address of both variables to see if it is the same for both.

⇒ Write the code to make a Shallow Copy and a Deep Copy of the user variable.

const user3 = { name: 'Amit', address: { city: 'Delhi', pincode: 110001, }, } const user4 = user3; // Reference copy → Same memory address const user5 = { ...user3 }; // Shallow copy → New memory for outer object only const user6 = structuredClone(user3); // Deep copy → New memory for both outer and inner objects console.log(user3); // Object-level comparison console.log(user3 === user4); // true → Same memory address (reference copy) console.log(user3 === user5); // false → Different memory address (shallow copy) console.log(user3 === user6); // false → Different memory address (deep copy) // Shallow Copy: nested object (address) is still the same console.log(user3.address === user5.address); // true → Same memory address (shared nested object) // Deep Copy: nested object (address) is also new console.log(user3.address === user6.address); // false → Different memory address (not shared)

5	**Guess the output of the below code.** 1 mark *const original = {* *name: "Amit",* *address: {* *city: "Delhi",* *pincode: 110001* *}* *};* *const copy = Object.assign({}, original);* *copy.address.city = "Mumbai";* *console.log(original.address.city);* 1. Delhi 2. Mumbai 3. undefined 4. Error	**Correct answer** 1 mark
6	**Guess the output of the below code.** 1 mark *const people = ["Rahul", { name: "Sita", address: { city: "Kolkata", pincode: 700001 } }];* *const shallowCopy = people.slice();* *shallowCopy[1].address.city = "Hyderabad";* *console.log(people[1].address.city);*	**Correct answer** 1 mark

1. Kolkata
2. undefined
3. Error
4. Hyderabad