JavaScript - Spread and Rest Operator, Shallow Copy vs Deep Copy

1. Spread Operator (...)

Definition

The spread operator is three dots `...` used to *expand* an array, object, or iterable into individual items. In simple English: it takes a container (like a box of items) and opens it so each item comes out one by one.

When you copy an array or object using the spread operator, it creates a **new** array or object. But this new copy is only a **shallow copy**. That means the top-level values are copied, but if there are nested objects or arrays inside, those inner parts are still connected to the original.

You can think of the spread operator as saying, “Take everything inside this array or object and place it here piece by piece.”

Real-World Usage

- Combine arrays
- Copy arrays or objects
- Add new items without changing the old data

Syntax

```
const newArray = [...oldArray];
const newObject = {...oldObject};
```

Examples

Array Merging Examples

Simple Merge

```
const a = [1, 2];
const b = [3, 4];
const merged = [...a, ...b];
```

Merge Multiple Arrays

```
const x = [10, 20];
const y = [30, 40];
```

```
const z = [50, 60];
const all = [...x, ...y, ...z];
```

Object Merging Examples

Basic Merge

```
const obj1 = {a: 1};
const obj2 = {b: 2};
const mergedObj = {...obj1, ...obj2};
```

Override Keys While Merging

```
const base = {name: "Asha", age: 20};
const update = {age: 21};
const updated = {...base, ...update};
```

Nested Array Copying (Shallow)

```
const arr = [1, [2, 3]];
const shallowCopyArr = [...arr];
shallowCopyArr[1][0] = 99;
// original[1][0] also becomes 99 (shared nested array)
```

Nested Object Copying (Shallow)

```
const obj = {a: 1, b: {c: 2}};
const shallowCopyObj = {...obj};
shallowCopyObj.b.c = 100;
// obj.b.c also becomes 100
```

Simple

```
const numbers = [1, 2, 3];
const moreNumbers = [...numbers, 4, 5];
```

Intermediate

```
const person = { name: "Asha", age: 20 };
const updatedPerson = { ...person, age: 21 };
```

Advanced

```
function logThings(a, b, c) {
  console.log(a, b, c);
}
const values = ["apple", "banana", "orange"];
logThings(...values);
```

Easy Tip

Think of `...` as *unpacking* a bag and spreading the items on the table.

Common Mistakes

- Thinking spread operator makes a deep copy (it does NOT).

Visual Model

Array: `[a, b, c]` → `...` → `a b c`

Interview Question

Q: What does the spread operator do? **A:** It expands an array or object into individual items.

Practice Problems

1. Use spread operator to combine `[1, 2]` and `[3, 4]`.
2. Copy this object: `{a: 1, b: 2}` using spread.

Solutions

1. `const result = [...[1,2], ...[3,4]];`
2. `const copy = {...{a:1, b:2}};`

---

2. Rest Operator (...)

Definition

The rest operator also uses three dots `...`, but its purpose is different. It *collects* multiple values and packs them into one array. It is mainly used inside function parameters.

In plain English: when you don't know how many values a function will receive, the rest operator gathers all remaining values and stores them safely in a single array.

It is like giving a basket to a function and saying, "Whatever extra items you get, put them into this basket."

Only **one** rest parameter is allowed in a function, and it must be the **last** parameter. (...)

Definition

Rest operator collects many values into one array. It is also three dots `...`, but used in a function.

Real-World Usage

- Functions with unknown number of arguments

Syntax

```
function sum(...nums) {}
```

Examples

Simple

```
function show(...items) {
  console.log(items);
}
show(1, 2, 3);
```

Intermediate

```
function addAll(...nums) {
  return nums.reduce((a, b) => a + b);
}
```

Advanced

```
function mix(first, second, ...rest) {
  console.log(first);
  console.log(second);
  console.log(rest);
}
mix(1, 2, 3, 4, 5);
```

Tip

Spread = *open* the bag. Rest = *collect* the items into a bag.

Common Mistake

- Using more than one rest parameter in a function (not allowed).

Interview Question

Q: What is the rest operator? **A:** It gathers multiple arguments into one array.

Practice Problems

1. Write a function that returns the largest value using rest.

Solution

```
function largest(...nums){ return Math.max(...nums); }
```

---

3. Shallow Copy vs Deep Copy

Detailed Definition

Shallow Copy

A shallow copy creates a new object or array, but only the top level is copied. If the object contains nested objects or arrays, those inner objects are **not copied**. They still point to the original.

In simple English: - The outer box is new. - The inner boxes are still shared.

So changes inside nested objects affect both the original and the copied version.

Common shallow copy methods: - `{ ...obj }` - `[ ...arr ]` - `Object.assign({}, obj)`

Deep Copy

A deep copy creates a fully independent clone of the object, including all nested objects.

In simple English: - The outer box is copied. - The inner boxes are also copied. - Even the inner-inner boxes are copied.

Changes in the deep copy never affect the original.

One simple deep copy method:

```
const deepCopy = JSON.parse(JSON.stringify(data));
```

Nested Object Example (Shallow vs Deep)

```
const original = {
  name: "Asha",
  details: {
    city: "Delhi",
    score: 90,
    address: { pin: 12345 }
  }
```

```
};

// Shallow Copy
const shallowCopy = { ...original };
shallowCopy.details.address.pin = 99999;
// original.details.address.pin also becomes 99999

// Deep Copy
const deepCopyObj = JSON.parse(JSON.stringify(original));
deepCopyObj.details.address.pin = 55555;
// original remains unchanged
```
```js
const original = {
  name: "Asha",
  details: {
    city: "Delhi",
    score: 90
  }
};

// Shallow Copy
const shallowCopy = { ...original };
shallowCopy.details.score = 100;
// original.details.score also becomes 100

// Deep Copy
const deepCopyObj = JSON.parse(JSON.stringify(original));
deepCopyObj.details.score = 120;
// original stays unchanged
```

Nested Object Comparison

JavaScript cannot compare objects directly because they compare memory locations.

Correct Way to Compare Nested Objects

```
function deepCompare(obj1, obj2) {
  const keys1 = Object.keys(obj1);
  const keys2 = Object.keys(obj2);

  if (keys1.length !== keys2.length) return false;

  for (let key of keys1) {
    const val1 = obj1[key];
    const val2 = obj2[key];

    const bothObjects = typeof val1 === "object" && val1 !== null && typeof
val2 === "object" && val2 !== null;

    if (bothObjects) {
```

```
      if (!deepCompare(val1, val2)) return false;
    } else {
      if (val1 !== val2) return false;
    }
  }

  return true;
}
```

Tip

Shallow copy = copy the box only. Deep copy = copy the box and boxes inside.

Interview Question

Q: What is a shallow copy? **A:** A copy that does not duplicate inner objects.

Practice Problem

Copy object deeply: `{a:1, b:{c:2}}`.

Solution

```
const deep = JSON.parse(JSON.stringify(obj));
```

---

4. JSON.stringify()

Definition

`JSON.stringify()` converts JavaScript data into JSON string format.

More Examples

Convert nested object

```
const data = {
  name: "Ravi",
  scores: { math: 90, science: 85 }
};
const result = JSON.stringify(data);
```

Convert array of objects

```
const students = [
  {name: "Asha", age: 20},
  {name: "Ravi", age: 22}
```

```
];
JSON.stringify(students);
```

Convert object with array

```
const obj = {
  id: 1,
  items: ["pen", "book", "pencil"]
};
JSON.stringify(obj);
```

Tip

Think of JSON.stringify() as turning your data into a text story.

Interview Question

Q: What does JSON.stringify() do? **A:** Converts JavaScript data into a JSON string.

Practice Problem

Convert `{x:10, y:20}` to JSON.

Solution

`JSON.stringify({x:10,y:20});`

---

5. JSON.parse()

Definition

`JSON.parse()` converts JSON text into a real JavaScript object.

More Examples

Parse nested JSON

```
const str = '{"user":{"name":"Ali","age":25}}';
const obj = JSON.parse(str);
```

Parse array JSON

```
const jsonArr = '[1, 2, 3, 4]';
JSON.parse(jsonArr);
```

Parse array of objects JSON

```
const jsonText = '[{"id":1,"name":"Asha"},{"id":2,"name":"Ravi"}]';
JSON.parse(jsonText);
```()

Definition

`JSON.parse()` takes a JSON text string and converts it back into a real JavaScript object.

In simple English:

- It reads the text created by `JSON.stringify()`
- It rebuilds the original data structure
- Now you can use it as a real object again

You can imagine it like taking the written text (your object written on paper) and rebuilding the real box with real items.()

Definition

Turns a JSON string back into a JavaScript object.

Example

```js
const str = '{"name":"Ali"}';
const obj = JSON.parse(str);
```

Tip

JSON.parse() turns the text story back into real data.

Interview Question

Q: What does JSON.parse() do? **A:** Converts JSON text to a JavaScript object.

Practice Problem

Parse this: `'{"a":1}'`.

Solution

`JSON.parse('{"a":1}');`