What is object de-structuring in JavaScript and how can you use it to assign default values?

Object destructuring in JavaScript is a concise way to extract values from objects and assign them to variables. It allows you to "unpack" properties from an object into individual variables in a clean and readable way. You can also assign **default values** in case the object does not have the property you're trying to extract.

```
Syntax

const { property1, property2 } = object;

// Assigning without default properties

const person = {
    name: "John",
    age: 30,
};

// Destructuring
const { name, age } = person;

console.log(name);  // Output: John
console.log(age);   // Output: 30

// Assigning with default properties

const person = {
    name: "John",
    // age is not defined in the object
};

// Destructuring with default values
const { name, age = 25 } = person;

console.log(name);  // Output: John
console.log(age);   // Output: 25 (default value)
```

What is the rest operator and how is it used in function parameters?

The **rest operator** (...) gathers all the remaining function arguments into an array.

It allows functions to handle a **variable number of arguments** easily.

The rest operator must be the **last** parameter in the function signature, and it can work with other named parameters.

This feature makes functions much more flexible and avoids the need to explicitly handle each argument one by one.

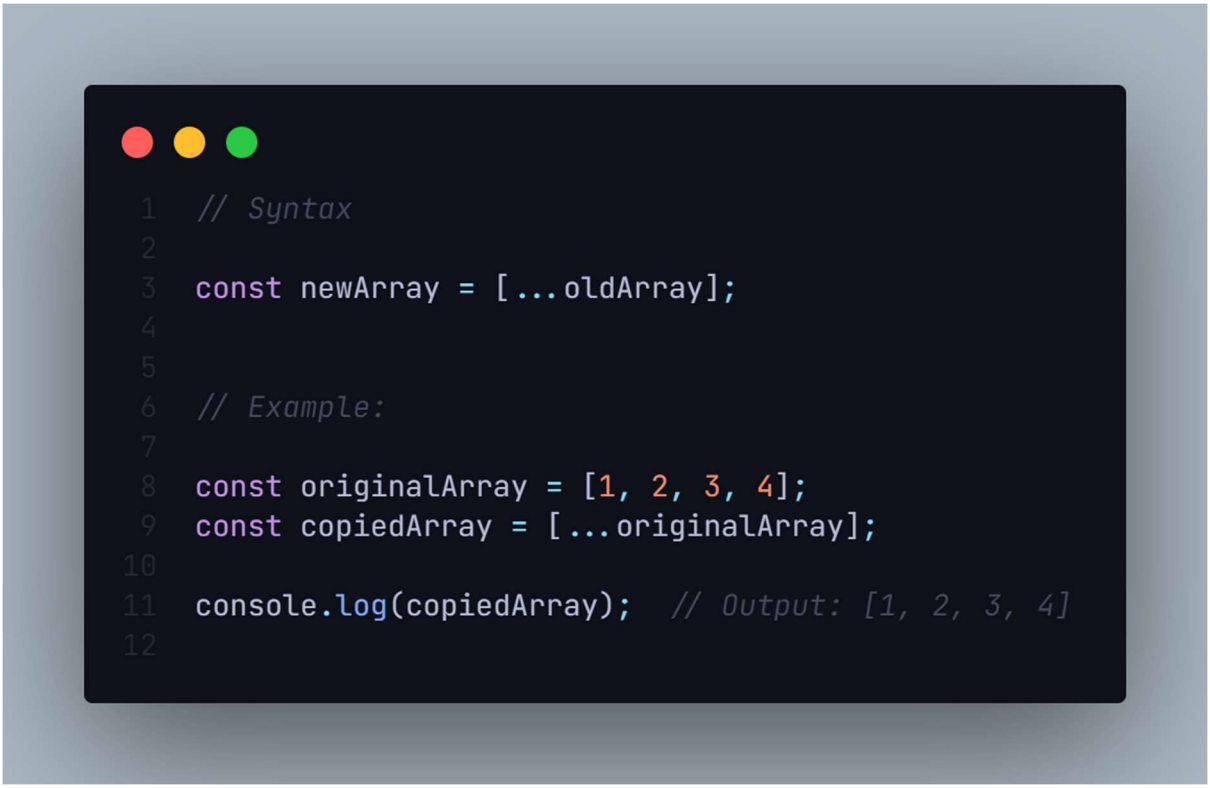
```
// Syntax

function myFunction(...args) {
  // args is an array of all the extra arguments
}

// example

function greet(greeting, ...names) {
    return `${greeting}, ${names.join(" and ")}!`;
}

console.log(greet("Hello", "Alice", "Bob", "Charlie"));
// Output: Hello, Alice and Bob and Charlie!
```

greeting takes the first argument ("Hello").

names collects the rest of the arguments (["Alice", "Bob", "Charlie"]) into an array.

Describe how the spread operator can be used to copy an array.

The **spread operator** (...) in JavaScript is a powerful tool for copying arrays and objects. When used with arrays, the spread operator **expands** the elements of an array into individual values, making it very easy to create shallow copies of arrays.



```
// Syntax

const newArray = [...oldArray];


// Example:

const originalArray = [1, 2, 3, 4];
const copiedArray = [...originalArray];

console.log(copiedArray);  // Output: [1, 2, 3, 4]

```

...originalArray spreads the elements of originalArray into individual values.

These values are then placed into a new array (copiedArray), effectively creating a **shallow copy** of originalArray.

The **spread operator** (...) is a simple and efficient way to **copy arrays**.

It creates a **shallow copy**, meaning it only copies the top-level elements of the array, not deeply nested objects.

It's a modern, clean alternative to methods like slice() or concat() for array copying.

What are template strings and how do they differ from regular strings in JavaScript?

Template strings are more **powerful**, **readable**, and **flexible** than regular strings in JavaScript. They make it easier to embed variables and expressions, handle multiline text, and avoid cumbersome string concatenation.

```
// Example

// Regular String
const name = "John";
const age = 30;
const message1 = "Hello, " + name + ". You are " + age + " years old.";
console.log(message1);

// Template String
const message2 = `Hello, ${name}. You are ${age} years old.`;
console.log(message2);
```

FEATURE	REGULAR STRING	TEMPLETE STRING
SYNTAX	'string' or "string"	`string`
VARIABLE INTERPOLATION	Uses string concatenation (+)	Embeds variables and expressions with ${}
MULTILINE STRINGS	Uses escape sequences or concatenation	Multiline directly with line breaks
EMBEDDING EXPRESSIONS	Not directly possible	Easily embed expressions using $ { }