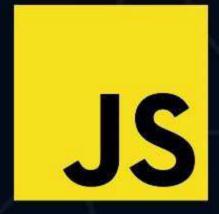
Destructuring Explained In JavaScript





Swipe

Destructuring in JavaScript is a convenient way to extract values from **arrays** or **objects** and assign them to variables.

It makes your code cleaner and more concise. Let's break it down with some simple examples:

Destructuring Arrays

Basic Array Destructuring

```
const numbers = [1, 2, 3, 4, 5];
const [first, second, third] = numbers;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(third); // Output: 3
```

This example takes an array numbers and extracts its values into variables first, second, and third.

Each variable holds one value from the array, respectively.

Skipping Values

You can skip values by leaving an empty space in the destructuring pattern

```
const numbers = [1, 2, 3, 4, 5];
const [first, , third] = numbers;
console.log(first); // Output: 1
console.log(third); // Output: 3
```

Here, we're still destructuring numbers, but we're **skipping** the second value by leaving an **empty space**.

So first gets the first value, and third gets the third value from the array.



Rest Operator (...)

It collects remaining items into a new array

```
const numbers = [1, 2, 3, 4, 5];
const [first, ...rest] = numbers;
console.log(first); // Output: 1
console.log(rest); // Output: [2, 3, 4, 5]
```

Using the **rest operator** (...), we capture the remaining values of numbers after assigning the first value to first.

The rest variable becomes an array containing all the remaining values.

Destructuring Objects

Basic Object Destructuring

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

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

In this case, we have an object **person** with properties **name** and **age**.

We extract these properties directly into variables name and age.

Changing Variable Names

You can assign a **new variable name** while destructuring.

```
const person = { name: 'John', age: 30 };
const { name: personName, age: personAge } = person;
console.log(personName); // Output: John
console.log(personAge); // Output: 30
```

By providing new variable names (personName and personAge), we're destructuring the name and age properties of person but assigning them to different variable names.

Default Values

You can provide default values in case the property is not present.

```
const person = { name: 'John', age: 30 };
const { name = 'Anonymous', job = 'Unemployed' } =
person;
console.log(name); // Output: John
console.log(job); // Output: Unemployed
```

If a property doesn't exist in the object (**job** in this case), it will default to the provided value ('**Unemployed**'). So, even though job doesn't exist in person, it gets assigned the default value 'Unemployed'.

Nested Object Destructuring

Destructuring can also be done for nested objects

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

const user = {
   id: 101,
    personalInfo: {
       name: 'Alice',
       age: 25,
   }
};

const { personalInfo: { name, age } } = user;

console.log(name); // Output: Alice
console.log(age); // Output: 25
```

This example involves destructuring a **nested object (personalInfo)** within the user object.

We directly extract the **name** and **age**properties of **personalInfo** into variables
name and age.