

# javascript

# Object Destructuring

7 usages you must know

1

Use destructuring  
to retrieve values  
from an object

2

Use destructuring to  
retrieve values from  
a nested object

3

Define a new  
variable with object  
destructuring

4

How to use JavaScript  
object destructuring  
aliases?

5

Handle dynamic name  
property with object  
destructuring

6

Destructure objects in  
the function argument  
and return value

7

Use object destructuring in loops



Explained →





# JavaScript objects

We use JavaScript objects to store data and retrieve it later. We store data(aka information) in key-value pairs. The key-value pair is also known as the object properties.

Here is an employee object with three properties: id, name, dept as keys and 007, 'James', and 'Spy' as values.

```
● ● ●  
  
const employee = {  
  id: 007,  
  name: 'James',  
  dept: 'Spy'  
}
```

swipe —→



# Use destructuring to retrieve values from an object

The most basic usage of object destructuring is to retrieve the value of a property key from the object.



```
const person = { firstName: "John", lastName: "Doe", age: 30 };

const { firstName, lastName, age } = person;

console.log(firstName); // "John"
console.log(lastName); // "Doe"
console.log(age); // 30
```

swipe —→



# Use destructuring to retrieve values from a nested object

```
const person = {  
    name: "John",  
    age: 30,  
    address: {  
        city: "New York",  
        state: "NY",  
        country: "USA"  
    }  
};  
const {  
    name, age,  
    address: { city, state, country }  
} = person;  
  
console.log(name); // "John"  
console.log(age); // 30  
console.log(city); // "New York"  
console.log(state); // "NY"  
console.log(country); // "USA"
```

swipe →



# Define a new variable with object destructuring

There could be a situation where you are unsure if the object has a specific key while retrieving its value. Also, you may want to **create a new variable with a default value** in case the key is unavailable in the object.

```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: 'Spy'  
}  
  
const { name, age=25 } = employee;  
console.log(age); //25  
  
// Magic  
const {name, dept, message =  
` ${name} is ${dept}`} = employee;  
  
console.log(message); // James is Spy
```

swipe —→



# How to use JavaScript object destructuring aliases?

Let's assume your source code has an existing variable named dept. So if we use the same variable name in destructuring, there will be a name conflict. We use aliases in this case.

```
const { dept: department } = employee;  
console.log(department); //Spy
```

Please note, we have destructured with the alias name, not with the actual key name that is still not defined.

```
console.log(dept);  
  
// Uncaught ReferenceError: dept is not defined
```

swipe —→



# Handle dynamic name property with object destructuring

We often handle API response data as JavaScript objects. These objects may contain dynamic data such that, as a client, we may not even know the property key names in advance.

Can we write a function that returns the value of the employee object properties when we pass a key as an argument? Yes, so it means we will not hard-code the key name inside the function. It is dynamic for the function.

```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: 'Spy'  
}
```

```
function getPropertyValue(key) {  
  const { [key]: returnValue } = employee;  
  return returnValue;  
}  
  
const id = getPropertyValue('id');  
const name = getPropertyValue('name');  
  
console.log(id, name); // 7 'James'
```

swipe →



# Destructure objects in the function argument and return value

You must learn this usage if you want to explore any modern JavaScript-based frameworks/libraries like React, Vue, Svelte, Angular, etc. You can use object destructuring to pass the property values as arguments to the function.

```
const employee = {  
    id: 007,  
    name: 'James',  
    dept: 'Spy'  
}  
  
function logEmployee({name, dept}) {  
    console.log(` ${name} is ${dept}`);  
}  
  
logEmployee(employee); // James is Spy
```

Now if you are interested to retrieve the value of the age property, you can do it like,

```
const { age } = getUser();  
console.log(age); // 45
```

swipe —→



# Use object destructuring in loops

Let's think of an array of employee objects. We want to iterate through the array and want to use the property values of each of the employee object. You can use the for-of loop to loop through the employees array and then use the object destructuring assignment syntax to retrieve the details.

```
const employees= [
  {
    'name': 'Alex',
    'address': '15th Park Avenue',
    'age': 43
  },
  {
    'name': 'John',
    'address': 'USA',
    'age': 33
  },
  {
    'name': 'Ravi',
    'address': 'Bangalore',
    'age': 16
  }
];
```

```
for(let {name, age} of employees) {
  console.log(` ${name} is ${age} years old!!!`);
}
```

swipe —→

SUNIL VISHWAKARMA  
@linkinsunil

# Thats a Wrap!

If you liked it, visit my profile and checkout for other  
**short and easy explanations**

**Context API vs Redux-Toolkit**

Feature ▾	Context API ▾	Redux-Toolkit ▾
State Management	Not a full-fledged state management tool. Passes down values and update functions, but does not have built-in ability to store, get, update, and notify changes in values.	A full-fledged state management tool with built-in ability to store, get, update, and notify changes in values.
Usage	Best for passing static or infrequently updated values and moderately complex state that does not cause performance issues when passed using props.	Best for managing large-scale, complex state that requires asynchronous actions and side-effects.
Code Complexity	Minimal setup and low learning curve. However, can become complex when used with a large number of components and nested Contexts.	
Performance	Can cause unnecessary re-renders if the state passed down is not simple and can require the use of additional memoization techniques to optimize performance.	
Developer Tools	Does not come with pre-built developer tools but can be used with third-party tools like React DevTools.	
Community	Has a large and active community.	

**React**

**Virtual DOM**

**useRef()**  
referencing values in React

When you want a component to remember some information, but you don't want that information to trigger new renders, you can use a ref.

Lets See into

1. How to add a ref to component?
2. How to update a ref's value?
3. How refs are different from state?
4. When to use refs?
5. Best practices for using refs?

{ Current }

⚠ Please Like & Share for no reason

VS

TF is a Virtual DOM?

real DOM

swipe →

**JavaScript Evolution**

**ES6** ES2015

1. let and const
2. Arrow functions
3. Default parameters
4. Rest and spread operators
5. Template literals
6. Destructuring assignment
7. Classes and inheritance
8. Promises for asynchronous programming
9. Symbols for creating unique object keys
10. Iterators and generators

**ES9** ES2018

1. Object.getOwnPropertyDescriptors()
2. Spread syntax for objects
3. Promise.prototype.finally()

**ES10** ES2019

1. Array.prototype.flat()
2. Array.prototype.flatMap()
3. String.prototype.trimStart()
4. String.prototype.trimEnd()
5. Array.prototype.sort() (stable)

**ES11** ES2020

1. BigInt
2. Nullish coalescing operator (??)
3. Optional chaining operator (?)
4. Promise.allSettled()

**ES12** ES2021

1. String.prototype.replaceAll()
2. Logical assignment operators (|=, &=&, ??=)

**ES13** ES2022

1. Array.prototype.lastIndexOf()
2. Object.hasOwn()
3. at() for strings and arrays
4. Top level await()

share

**Redux Toolkit**  
Easiest Explanation Ever

React      Redux Toolkit

swipe →

like and share

Instagram icon @linkinsunil

Twitter icon @officialskv



SUNIL VISHWAKARMA  
@linkinsunil

P.S.

**Repost this if you  
think your followers  
will like it**





SUNIL VISHWAKARMA  
@linkinsunil

# Enjoyed this?

1. Follow me
2. Click the notification
3. Never miss a post