

SUNIL VISHWAKARMA  
@linkinsunil



# #JAVASCRIPT OBJECT METHODS

Everything you need to master Objects in JS

Object.**create()**

Object.**keys()**

Object.**values()**

Object.**entries()**

Object.**assign()**

Object.**freeze()**

Object.**seal()**

Object.**getPrototypeOf()**

Object.**setPrototypeOf()**



save it, like and share

swipe →

# Object.create()

The Object.create() method is used to **create a new object** and **link it to** the prototype of an **existing object**. We can create a job object instance, and extend it to a more specific object.



```
const job = {  
    position: 'cashier',  
    type: 'hourly',  
    isAvailable: true,  
    showDetails() {  
        const accepting = this.isAvailable ? 'is open' : 'is closed'  
        console.log(`The ${this.position} position is ${this.type} and ${accepting}.`)  
    },  
}  
  
// Use Object.create to pass properties  
const barista = Object.create(job)  
  
barista.position = 'barista'  
barista.showDetails()  
  
// Output  
// The barista position is hourly and is open.
```

swipe →

# Object.keys()

Object.keys() creates an array containing the keys of an object. We can create an object and print the array of keys.



```
// Initialize an object
const employees = {
    boss: 'Michael',
    secretary: 'Pam',
    sales: 'Jim',
    accountant: 'Oscar'
};

// Get the keys of the object
const keys = Object.keys(employees);
console.log(keys);

// Output
["boss", "secretary", "sales", "accountant"]
```

swipe →

# Object.values()

Object.values() creates an array containing the values of an object.



```
// Initialize an object
const session = {
  id: 1,
  time: `26-July-2018`,
  device: 'mobile',
  browser: 'Chrome'
};

// Get all values of the object
const values = Object.values(session);
console.log(values);

// Output
[1, "26-July-2018", "mobile", "Chrome"]
```

swipe →

# Object.entries()

Object.entries() creates a nested array of the key/value pairs of an object.



```
// Initialize an object
const operatingSystem = {
    name: 'Ubuntu',
    version: 18.04,
    license: 'Open Source'
};

// Get the object key/value pairs
const entries =
Object.entries(operatingSystem);
console.log(entries);

// Output
[
    ["name", "Ubuntu"]
    ["version", 18.04]
    ["license", "Open Source"]
]
```

swipe →

# Object.assign()

Object.assign() is used to copy values from one object to another. We can create two objects, and merge them with Object.assign().



```
// Initialize an object
const name = {
    firstName: 'Philip',
    lastName: 'Fry'
};

// Initialize another object
const details = {
    job: 'Delivery Boy',
    employer: 'Planet Express'
};

// Merge the objects
const character = Object.assign(name, details);
console.log(character);

// Output
{firstName: "Philip", lastName: "Fry", job: "Delivery Boy", employer: "Planet Express"}
```

swipe →

# Object.freeze()

Object.freeze() prevents modification to properties and values of an object, and prevents properties from being added or removed from an object.

```
// Initialize an object
const user = {
    username: 'AzureDiamond',
    password: 'hunter2'
};

// Freeze the object
const newUser = Object.freeze(user);

newUser.password = '*****';
newUser.active = true;
console.log(newUser);

// Output
{username: "AzureDiamond", password: "hunter2"}
```

swipe →

# Object.seal()

Object.seal() prevents new properties from being added to an object, but allows the modification of existing properties. This method is similar to Object.freeze(). Refresh your console before implementing the code below to avoid an error.



```
// Initialize an object
const user = {
    username: 'AzureDiamond',
    password: 'hunter2'
};

// Seal the object
const newUser = Object.seal(user);

newUser.password = '*****';
newUser.active = true;
console.log(newUser);

// Output
{username: "AzureDiamond", password: "*****"}
```

swipe →

# Object.getPrototypeOf()

Object.getPrototypeOf() is used to get the internal hidden [[Prototype]] of an object, also accessible through the \_\_proto\_\_ property.

In this example, we can create an array, which has access to the Array prototype.



```
const employees = ['Ron', 'April', 'Andy', 'Leslie'];

Object.getPrototypeOf(employees);

// Output
[constructor: f, concat: f, find: f, findIndex: f, pop: f, ...]
```

swipe →

# Object.setPrototypeOf()

The `Object.setPrototypeOf()` static method **sets the prototype** (i.e., the internal `[[Prototype]]` property) of a specified object to another object or null.



```
const obj = {};
const parent = { foo: 'bar' };

console.log(obj.foo);
// Expected output: undefined

Object.setPrototypeOf(obj, parent);

console.log(obj.foo);
// Expected output: "bar"
```

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      LinkedIn icon @linkinsunil      Twitter icon @officialskv

P.S.

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



# Enjoyed this?

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