

Ways to Create Objects in JavaScript

👉 Object constructor

👉 Object's create method

👉 Object literal syntax

👉 Function constructor

👉 Function constructor with prototype

👉 ES6 Class syntax

👉 Singleton pattern



save it, like and share

swipe →



Object Constructor

The simplest way to create an empty object is using the Object constructor. Currently this approach is not recommended.

```
var object = new Object();
```

swipe —→



Object's Create Method

The create method of Object creates a new object by passing the prototype object as a parameter



```
var object = Object.create(null);
```

swipe —→



Object Literal Syntax

The object literal syntax (or object initializer), is a comma-separated set of name-value pairs wrapped in curly braces.

```
● ● ●  
  
var object = {  
    name: "Sunil",  
    age: 25  
};  
  
//Note:  
Object literal property values can be  
of any data type, including array,  
function, and nested object.
```

Note: This is the easiest way to create an object

swipe —→



Function Constructor

Create any function and apply the new operator to create object instances.

```
● ● ●  
function Person(name) {  
  this.name = name;  
  this.age = 25;  
}  
var object = new Person("Sunil");
```

swipe —→



Function Constructor With Prototype

This is similar to function constructor but it uses prototype for their properties and methods.

```
function Person() {}  
Person.prototype.name = "Sunil";  
var object = new Person();
```

This is equivalent to an instance created with an object create method with a function prototype and then call that function with an instance and parameters as arguments.

```
function func() {};  
new func(x, y, z);
```

OR

swipe —→



...continued



```
// Create a new instance using function prototype.  
var newInstance = Object.create(func.prototype)  
  
// Call the function  
var result = func.call(newInstance, x, y, z),  
  
// If the result is a non-null object then use it  
// otherwise just use the new instance.  
console.log(result && typeof result === 'object'  
? result : newInstance);
```

swipe —→



ES6 Class Syntax

ES6 introduces class feature to create the objects

```
● ● ●  
  
class Person {  
  constructor(name) {  
    this.name = name;  
  }  
}  
  
var object = new Person("Sunil");
```

swipe —→



Singleton pattern

A Singleton is an object which can only be instantiated one time. Repeated calls to its constructor return the same instance and this way one can ensure that they don't accidentally create multiple instances.

```
var object = new (function () {
  this.name = "Sunil";
})();
```

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