

Below is a clean, corrected, deeply conceptual, and exam-ready explanation of the `new` keyword in JavaScript, written in the same structured learning style you've been following.

I'll also point out internal mechanics, edge cases, and best practices (important for interviews + debugging).

◆ `new` Keyword in JavaScript

The `new` keyword in JavaScript is used to **create a new object instance from a constructor function or class**.

When a function is called with `new`, JavaScript treats that function as a **constructor** and performs a **well-defined internal process** to create and return an object.

📌 One-Line Definition (Exam Ready)

The `new` keyword creates a new object, links it to a constructor's prototype, binds `this` to the new object, executes the constructor, and returns the object.

🧠 What Happens Internally When `new` Is Used?

When you execute:

```
js  
  
new Constructor(args)
```

JavaScript performs **5 internal steps**:

✓ Step 1: Create an Empty Object

```
js  
  
{}
```

A brand-new empty object is created in memory.

✓ Step 2: Set Prototype (Inheritance)

```
js  
  
obj.__proto__ = Constructor.prototype
```

This links the new object to the constructor's prototype, enabling **prototype-based inheritance**.

📌 This is why methods defined on `Constructor.prototype` are accessible.

Step 3: Bind `this`

Inside the constructor:

```
js
```

`this` → newly created object

So assignments like:

```
js
```

```
this.name = "value";
```

attach properties to the new object.

Step 4: Execute Constructor Function

The constructor body runs using the new object as `this`.

Step 5: Return the Object

- If constructor returns **nothing or a primitive** → new object is returned
- If constructor returns a **non-null object** → that object is returned instead

 This rule is **very important**.

Syntax

```
js
```

```
new Constructor(arguments);
```

Parameters

Parameter	Description
Constructor	Function or class used as blueprint
arguments	Values to initialize properties

Return Value Rules

Constructor Return	Actual Return
Nothing	New object
Primitive	New object

Constructor Return

Actual Return

Object

Returned object

🔧 Using `new` with Function Constructor

Example

html

```
<script>
function Watch(brand, price, type) {
  this.brand = brand;
  this.price = price;
  this.type = type;
}

const titan = new Watch("Titan", 4000, "Analog");
const sonata = new Watch("Sonata", 3000, "Digital");

console.log(titan);
console.log(sonata);
</script>
```

Output

js

```
{ brand: "Titan", price: 4000, type: "Analog" }
{ brand: "Sonata", price: 3000, type: "Digital" }
```

✓ Each call creates a **separate object**

✓ Same constructor logic reused

⚠ Important Performance Note

✗ Bad Practice

js

```
function Laptop() {
  this.getDetails = function() { }
}
```

Every object gets its **own copy** of the function.

✓ Best Practice: Use Prototype

js

```
function Laptop(brand, model, cpu) {  
    this.brand = brand;  
    this.model = model;  
    this.cpu = cpu;  
}  
  
Laptop.prototype.getDetails = function () {  
    return `${this.brand}, ${this.model}, ${this.cpu}`;  
};
```

✓ One shared method

✓ Memory efficient

✓ Faster

🧱 Using `new` with ES6 Classes

Classes are **syntactic sugar** over constructor functions.

Example

html

```
<script>  
class WindowClass {  
    constructor(color, size, type) {  
        this.color = color;  
        this.size = size;  
        this.type = type;  
    }  
  
    getDetails() {  
        return `${this.color}, ${this.size}, ${this.type}`;  
    }  
}  
  
const window1 = new WindowClass("Blue", "Small", "Wooden");  
console.log(window1.getDetails());  
</script>
```

Output

text

Blue, Small, Wooden

✖ new is mandatory with classes

✖ Calling class without new throws an error

⚠️ Using new with Built-in Objects

JavaScript provides built-in constructors like:

- Number
- String
- Boolean
- Array
- Object
- Date

Example

js

```
const num = new Number(10);
const str = new String("Hello");

console.log(typeof num); // object
console.log(typeof str); // object
```

Output

text

object

object

⚠️ Important Warning (Interview Favorite)

✖ Avoid using new with primitive wrappers

js

```
new Number(10) ✖
new String("Hi") ✖
```

new Boolean(true) 

Why?

- Creates **object wrappers**
- Causes unexpected comparisons
- Breaks strict equality

Correct Way

js

```
let num = 10;  
let str = "Hello";  
let flag = true;
```

Constructor Returning Object (Edge Case)

js

```
function Test() {  
  this.a = 10;  
  return { b: 20 };  
}  
  
const obj = new Test();  
console.log(obj);
```

Output

js

```
{ b: 20 }
```

 Returned object replaces `this`

Constructor Returning Primitive

js

```
function Demo() {  
  this.x = 5;  
  return 100;  
}
```

```
const obj = new Demo();
console.log(obj);
```

Output

js

```
{ x: 5 }
```

✓ Primitive ignored

✗ Forgetting `new` (Very Common Bug)

js

```
function Car(name) {
  this.name = name;
}
```

```
const c = Car("BMW"); // ✗
```

Result

- `this` → `window` (non-strict)
- `this` → `undefined` (strict)
- Property leaks or error occurs

✓ Always use `new` with constructors

📝 Summary Table

Feature	<code>new</code> Keyword
Creates object	✓
Sets prototype	✓
Binds <code>this</code>	✓
Calls constructor	✓
Returns object	✓
Required for class	✓

Used with built-ins

 Avoid

⌚ Final Exam Takeaway

The `new` keyword creates an object instance by allocating memory, setting prototype linkage, binding `this`, executing the constructor, and returning the created object.

If you want next:

- `new` vs factory functions
- `prototype` deep dive
- Memory optimization patterns
- Interview MCQs
- `this + new` traps

Just say **next** 