



# What are Constructors in JavaScript?



## Table of Contents

What is a Constructor in JavaScript?

What Happens When A Constructor Gets Called?

JavaScript Constructor Examples

- Using the "this" Keyword

- Create Multiple Objects

- Constructor with Parameters

- Constructor vs Object Literal

- Object Prototype

- Built-in Constructors

Track, Analyze and Manage Errors With Rollbar

[Try Rollbar Today](#)

Proactively discover and resolve errors

## What is a Constructor in JavaScript?

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the `new` keyword.

The purpose of a constructor is to create a new object and set values for any existing object properties.

### What Happens When A Constructor Gets Called?

When a constructor gets invoked in JavaScript, the following sequence of operations take place:

- A new empty object gets created.
- The `this` keyword begins to refer to the new object and it becomes the current instance object.
- The new object is then returned as the return value of the constructor.

### JavaScript Constructor Examples

Here's a few examples of constructors in JavaScript:

#### Using the "this" Keyword

When the `this` keyword is used in a constructor, it refers to the newly created object:

```
//Constructor
function User() {
    this.name = 'Bob';
}

var user = new User();
```

#### Create Multiple Objects

In JavaScript, multiple objects can be created in a constructor:

```
//Constructor
function User() {
    this.name = 'Bob';
}

var user1 = new User();
var user2 = new User();
```

In the above example, two objects are created using the same constructor.

#### Constructor with Parameters

A constructor can also have parameters:

```
//Constructor
function User (name, age) {
    this.name = name;
    this.age = age;
}

var user1 = new User('Bob', 25);
var user2 = new User('Alice', 27);
```

In the above example, arguments are passed to the constructor during object creation. This allows each object to have different property values.

### Constructor vs Object Literal

An object literal is typically used to create a single object whereas a constructor is useful for creating multiple objects:

```
//Object literal
let user = {
    name: 'Bob'
}

//Constructor
function User() {
    this.name = 'Bob';
}

var user1 = new User();
var user2 = new User();
```

Each object created using a constructor is unique. Properties can be added or removed from an object without affecting another one created using the same constructor. However, if an object is built using an object literal, any changes made to the variable that is assigned the object value will change the original object.

### Object Prototype

Properties and methods can be added to a constructor using a prototype:

```
//Constructor
function User() {
    this.name = 'Bob';
}

let user1 = new User();
```

```
let user2 = new User();

//Adding property to constructor using prototype
User.prototype.age = 25;

console.log(user1.age); // 25
console.log(user2.age); // 25
```

In the above example, two `User` objects are created using the constructor. A new property `age` is later added to the constructor using a prototype, which is shared across all instances of the `User` object.

### Built-in Constructors

JavaScript has some built-in constructors, including the following:

```
var a = new Object();

var b = new String();
var c = new String('Bob')

var d = new Number();
var e = new Number(25);

var f = new Boolean();
var g = new Boolean(true);
```

Although these constructors exist, it is recommended to use primitive data types where possible, such as:

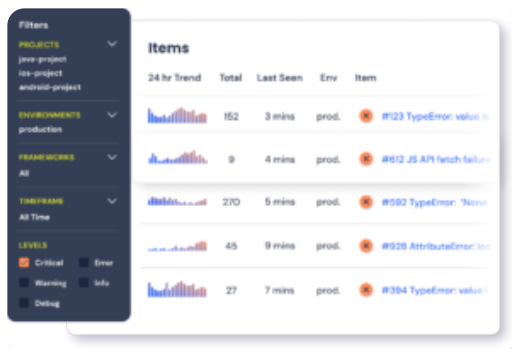
```
var a = 'Bob';

var b = 25;

var c = true;
```

Strings, numbers and booleans should not be declared as objects since they hinder performance.

### Track, Analyze and Manage Errors With Rollbar



Managing errors and exceptions in your code is challenging. It can make deploying production code an unnerving experience. Being able to track, analyze, and manage errors in real-time can help you to proceed with more confidence. Rollbar automates error monitoring and triaging, making fixing [Java errors](#) easier than ever. [Sign Up Today!](#)

JavaScript

Related Posts

[See all posts](#)





# What is Heap Space in Java?

Java Guide: What is Heap Space & Dynamic Memory Allocation?  
[Read more](#) ➔



## Headless Exception in Java

How to Handle the Headless Exception in Java

[Read more](#) →



## List of Checked & Unchecked Exceptions

Java: List of Checked & Unchecked Exceptions

[Read more](#) →

Start continuously improving your code today.

Try for free

Speak with an expert →

"Rollbar allows us to go from alerting to impact analysis and resolution in a matter of minutes. Without it we would be flying blind."

 Get Started Shape

Get the latest updates, tutorials and more,  
delivered to your inbox.

your@email.

Subscribe



## PRODUCT

Product

Pricing

Customers

Platforms

Integrations

## PLATFORMS

JavaScript

Ruby

iOS

.NET

React

Node

Django

## COMPANY

About Us

Careers

Contact Us

Changelog

 LinkedIn

 Twitter

 Github

## DOCUMENTATION

Docs Overview

Setting up Rollbar

Notifications

Deploy Tracking

Telemetry

Security & Compliance

API

PHP

Python

Java

Angular

Laravel

Rails

More...

