**JavaScript Object**

## **Properties**

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

# constructor

The constructor property returns a reference to the [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) constructor function that created the instance object.

**Example**

function Tree(name) {

this.name = name

}

let theTree = new Tree('Redwood')

console.log('theTree.constructor is ' + theTree.constructor)

# \_\_proto\_\_

The \_\_proto\_\_ property of [Object.prototype](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) is an accessor property (a getter function and a setter function) that exposes the internal [[Prototype]] (either an object or [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)) of the object through which it is accessed.

**Example**

const l = console.log

const obj = {

method: function() {

l("method in obj")

}

}

const obj2 = {}

obj2.\_\_proto\_\_ = obj

obj2.method()

**Object** [**Methods**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values)

# 1. Object.values()

The Object.values() method returns an array of a given object's own enumerable property values, in the same order as that provided by a [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loop.

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values#syntax)

Object.values(obj)

**Example**

const object1 = {

a: 'somestring',

b: 42,

c: false

};

console.log(Object.values(object1));

# 2. Object.toString()

The toString() method returns a string representing the object.

**Syntax**

toString()

**Example**

function Dog(name) {

this.name = name;

}

const dog1 = new Dog('Gabby');

Dog.prototype.toString = function dogToString() {

return `${this.name}`;

}; console.log(dog1.toString());

# 3. Object.seal()

The Object.seal() method seals an object, preventing new properties from being added to it and marking all existing properties as non-configurable. Values of present properties can still

be changed as long as they are writable.

**Syntax**

Object.seal(obj)

**Example**

const object1 = {

property1: 42

};

Object.seal(object1);

object1.property1 = 33;

console.log(object1.property1);

delete object1.property;

console.log(object1.property1);

# 4. toLocaleString()

The toLocaleString() method returns a string representing the object. This method is meant to be overridden by derived objects for locale-specific purposes.

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toLocaleString#syntax)

toLocaleString()

**Example**

const date1 = new Date(Date.UTC(2012, 11, 20, 3, 0, 0));

console.log(date1.toLocaleString('ar-EG'));

const number1 = 123456.789;

console.log(number1.toLocaleString('de-DE'));

# 5. Object.keys()

The Object.keys() method returns an array of a given object's own enumerable property **names**, iterated in the same order that a normal loop would.

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys#syntax)

Object.keys(obj)

**Example**

const object1 = {

a: 'somestring',

b: 42,

c: false

};

console.log(Object.keys(object1));

# 6. Object.isFrozen()

The Object.isFrozen() determines if an object is [frozen](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze).

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isFrozen#syntax)

Object.isFrozen(obj)

**Example**

const object1 = {

property1: 42

};

console.log(Object.isFrozen(object1));

Object.freeze(object1);

console.log(Object.isFrozen(object1));

# 7. Object.isExtensible()

The Object.isExtensible() method determines if an object is extensible (whether it can have new properties added to it).

**Syntax**

Object.isExtensible(obj)

**Example**

const object1 = {};

console.log(Object.isExtensible(object1));

Object.preventExtensions(object1);

console.log(Object.isExtensible(object1));

**8. Object.assign()**

The Object.assign() method copies all enumerable own properties from one or more source objects to a target object. It returns the modified target object.

**Syntax**

Object.assign(target, ...sources)

**Example**

const target = { a: 1, b: 2 };

const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);

console.log(returnedTarget);

# 9. Object.create()

The Object.create() method creates a new object, using an existing object as the prototype of the newly created object.

**Syntax**

Object.create(proto)

Object.create(proto, propertiesObject)

**Example**

const person = {

isHuman: false,

printIntroduction: function() {

console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);

}

};

const me = Object.create(person);

me.name = 'Matthew';

me.isHuman = true;

me.printIntroduction();

# 10. Object.entries()

The Object.entries() method returns an array of a given object's own enumerable string-keyed property [key, value] pairs.

**Syntax**

Object.entries(obj)

**Example**

const object1 = {

a: 'somestring',

b: 42

};

for (const [key, value] of Object.entries(object1)) {

console.log(`${key}: ${value}`);

}

**11. Object.getOwnPropertyNames()**

The Object.getOwnPropertyNames() method returns an array of all properties (including non-enumerable properties except for those which use Symbol) found directly in a given object.

**Syntax**

Object.getOwnPropertyNames(obj)

**Example**

const object1 = {

a: 1,

b: 2,

c: 3

};

console.log(Object.getOwnPropertyNames(object1));

# 12. Object.defineProperty()

The static method Object.defineProperty() defines a new property directly on an object, or modifies an existing property on an object, and returns the object.

**Syntax**

Object.defineProperty(obj, prop, descriptor)

**Example**

const object1 = {};

Object.defineProperty(object1, 'property1', {

value: 42,

writable: false

});

object1.property1 = 77;

console.log(object1.property1);

# 13. Object.fromEntries()

The Object.fromEntries() method transforms a list of key-value pairs into an object.

**Syntax**

Object.fromEntries(iterable);

**Example**

const entries = new Map([

['foo', 'bar'],

['baz', 42]

]);

const obj = Object.fromEntries(entries);

console.log(obj);