**JavaScript Constructor Function**

In this tutorial, you will learn about JavaScript constructor function with the help of examples.

In JavaScript, a constructor function is used to create objects. For example,

// constructor function

function Person () {

this.name = 'John',

this.age = 23

}

// create an object

const person = new Person();

[Run Code](https://www.programiz.com/javascript/online-compiler)

In the above example, function Person() is an object constructor function.

To create an object from a constructor function, we use the new keyword.

**Note**: It is considered a good practice to capitalize the first letter of your constructor function.

**Create Multiple Objects with Constructor Function**

In JavaScript, you can create multiple objects from a constructor function. For example,

// constructor function

function Person () {

this.name = 'John',

this.age = 23,

this.greet = function () {

console.log('hello');

}

}

// create objects

const person1 = new Person();

const person2 = new Person();

// access properties

console.log(person1.name); // John

console.log(person2.name); // John

[Run Code](https://www.programiz.com/javascript/online-compiler)

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

**JavaScript this Keyword**

In JavaScript, when this keyword is used in a constructor function, this refers to the object when the object is created. For example,

// constructor function

function Person () {

this.name = 'John',

}

// create object

const person1 = new Person();

// access properties

console.log(person1.name); // John

[Run Code](https://www.programiz.com/javascript/online-compiler)

Hence, when an object accesses the properties, it can directly access the property as person1.name.

**JavaScript Constructor Function Parameters**

You can also create a constructor function with parameters. For example,

// constructor function

function Person (person\_name, person\_age, person\_gender) {

// assigning parameter values to the calling object

this.name = person\_name,

this.age = person\_age,

this.gender = person\_gender,

this.greet = function () {

return ('Hi' + ' ' + this.name);

}

}

// creating objects

const person1 = new Person('John', 23, 'male');

const person2 = new Person('Sam', 25, 'female');

// accessing properties

console.log(person1.name); // "John"

console.log(person2.name); // "Sam"

[Run Code](https://www.programiz.com/javascript/online-compiler)

In the above example, we have passed arguments to the constructor function during the creation of the object.

const person1 = new Person('John', 23, 'male');

const person2 = new Person('Sam', 25, 'male');

This allows each object to have different properties. As shown above,

console.log(person1.name); gives John

console.log(person2.name); gives Sam

**Create Objects: Constructor Function Vs Object Literal**

* Object Literal is generally used to create a single object. The constructor function is useful if you want to create multiple objects. For example,

// using object literal

let person = {

name: 'Sam'

}

// using constructor function

function Person () {

this.name = 'Sam'

}

let person1 = new Person();

let person2 = new Person();

* Each object created from the constructor function is unique. You can have the same properties as the constructor function or add a new property to one particular object. For example,

// using constructor function

function Person () {

this.name = 'Sam'

}

let person1 = new Person();

let person2 = new Person();

// adding new property to person1

person1.age = 20;

Now this age property is unique to person1 object and is not available to person2 object.

However, if an object is created with an object literal, and if a variable is defined with that object value, any changes in variable value will change the original object. For example,

// using object lateral

let person = {

name: 'Sam'

}

console.log(person.name); // Sam

let student = person;

// changes the property of an object

student.name = 'John';

// changes the origins object property

console.log(person.name); // John

[Run Code](https://www.programiz.com/javascript/online-compiler)

When an object is created with an object literal, any object variable derived from that object will act as a clone of the original object. Hence, any change you make in one object will also reflect in the other object.

**Adding Properties And Methods in an Object**

You can add properties or methods in an object like this:

// constructor function

function Person () {

this.name = 'John',

this.age = 23

}

// creating objects

let person1 = new Person();

let person2 = new Person();

// adding property to person1 object

person1.gender = 'male';

// adding method to person1 object

person1.greet = function () {

console.log('hello');

}

person1.greet(); // hello

// Error code

// person2 doesn't have greet() method

person2.greet();

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

hello

Uncaught TypeError: person2.greet is not a function

In the above example, a new property gender and a new method greet() is added to the person1 object.

However, this new property and method is only added to person1. You cannot access gender or greet() from person2. Hence the program gives error when we try to access person2.greet();

**JavaScript Object Prototype**

You can also add properties and methods to a constructor function using a **prototype**. For example,

// constructor function

function Person () {

this.name = 'John',

this.age = 23

}

// creating objects

let person1 = new Person();

let person2 = new Person();

// adding new property to constructor function

Person.prototype.gender = 'Male';

console.log(person1.gender); // Male

console.log(person2.gender); // Male

[Run Code](https://www.programiz.com/javascript/online-compiler)

To learn more about prototypes, visit [JavaScript Prototype](https://www.programiz.com/javascript/prototype).

**JavaScript Built-in Constructors**

JavaScript also has built-in constructors. Some of them are:

let a = new Object(); // A new Object object

let b = new String(); // A new String object

let c = new Number(); // A new Number object

let d = new Boolean(); // A new Boolean object

In JavaScript, strings can be created as objects by:

const name = new String ('John');

console.log(name); // "John"

[Run Code](https://www.programiz.com/javascript/online-compiler)

In JavaScript, numbers can be created as objects by:

const number = new Number (57);

console.log(number); // 57

[Run Code](https://www.programiz.com/javascript/online-compiler)

In JavaScript, booleans can be created as objects by:

const count = new Boolean(true);

console.log(count); // true

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Note**: It is recommended to use primitive data types and create them in a normal way, such as const name = 'John';, const number = 57; and const count = true;

You should not declare strings, numbers, and boolean values as objects because they slow down the program.

**Note**: In JavaScript, the keyword class was introduced in ES6 (ES2015) that also allows us to create objects. Classes are similar to constructor functions in JavaScript. To learn more, visit [JavaScript Classes](https://www.programiz.com/javascript/ES6#class).