# Objects in JavaScript

### What are Objects in JavaScript

In Javascript, objects are made by other objects. An object is a collection of key-value or name-value pairs. Each key-value pair is called a property.

A property’s value can be a function, an array, an object itself or any primitive data type i.e. integer, string, etc. in which case the property are known as a method, an array and an object property.

Let’s consider a simple object:

var person = {

firstName: "Bablu",

lastName: "Ahmed",

age: 30,

fullName: function(){

return this.firstName + " " + this.lastName

}

}

Here firstName, lastName, and fullName with value are properties of the same object i.e. person. **firstName: "Bablu"** is a Property and *firstName* is **Property name** & “Bablu” is **Property value**. As we can see the property value is a string value, hence the property is called string property in the same manner, the fullName is called method property.

Every person will have these properties but their values may be different i.e. firstName, lastName keys may have different values for different person.

console.log(person);

#### Properties of the object can be accessed using the following two notations:

1. Dot notation

2. Square bracket notation

### Dot notation

person.firstName; //Output: Bablu

person.fullName(); //Output: Bablu Ahmed

New properties can be added using the dot notation as shown below:

person.age = 27

person.getAge = function(){

return this.age;

}

Now console person and we will find new properties age and getAge() added to the person object as shown below:

Console.log(person);

### Square bracket notation

person["firstName"]; //Output: Bablu

person ["fullName"](); //Output: Bablu Ahmed

New properties can be added using the square bracket notation as shown below:

person ["weight"] = 65

person.getWeight = function(){

return this.weight;

}

Properties can also be accessed using a variable just assign the property name to the variable as shown below:

var firstNameProperty = "firstName";

console.log(person[firstNameProperty]) // Output: Bablu

**Note**: Above method of using variable to access property names cannot be accessed properties of the object using dot notation

Console.log(person.firstNameProperty) //Output: undefined

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid Javascript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed and added to the object property using the square bracket notation

person["date of birth"] = "Nov 10";

person[12] = 12;

person.12 = 12; //gives error

console.log(person);

console.log(person.12); //Gives error

console.log(person[12]); //Output: 12

**Different ways of creating a JavaScript object**

**Using object literal**

person object created above is an example of creating an object using object literal

var person = {

firstName: "Bablu",

lastName: "Ahmed",

age: 30,

fullName: function(){

return this.firstName + " " + this.lastName

}

}

**Using new Object()**

var person = new Object()

console.log(person); // Creates an empty object

We can added as many properties as we want usign either the dot notation or the square bracket notation

person.firstName = "Bablu";

person.lastName = "Ahmed";

person.age = 30;

person.fullName = function(){

return this.firstName + " " + this.lastName;

}

console.log(person)

Method 1 and 2 do exactly the same thing. There is no need to use new Object().

For simplicity, readability and execution speed, use the first one that is object literal.

We can also create empty object as follows:

var person = {};

person.firstName = "Bablu";

person.lastName = "Ahmed";

person.age = 30;

person.fullName = function(){

return this.firstName + " " + this.lastName;

}

Now check the object

console.log(person)

Objects are addressed by reference, not by value the following statement will not create a copy of person:

var human = person;

Now, human and person are the same and person object can be modified by human

**Object constructor function in Javascript**

Object Type (Blueprint) can be created using the constructor function following two steps:

1. Define an object type constructor function by convention, name of the constructor function starts with a capital letter

2. Create an instance (an object) from the Object Type using **new** keyword

To define an object type (blueprint), create a constructor function by specifying its name and properties.

Now we need a "blueprint" for creating many objects of the same type.

Lets create a constructor function

function Person(firstName, lastName){

this.firstName = firstName,

this.lastName = lastName,

this.fullName = function(){

return this.firstName + " " + this.lastName;

}

}

Now you can create many objects as you want using this constructor or blueprint.

var objMe = new Person("Bablu", "Ahmed");

console.log(obj);

console.log(objMe.fullName); //Bablu Ahmed

var objFather = new Person('Fazlu','Ahmed');

console.log(objFather.firstName); //Fazlu

Hence, the constructor function is called Object Constructor Type

Limitation of Constructor Function

Now, we know about various ways of creating objects in JavaScript. One of the ways to create objects in JavaScript is using the Constructor function. Consider the construction function below:

function Person(firstName, lastName){

this.firstName = firstName,

this.lastName = lastName,

this.fullName = function(){

return this.firstName + " " + this.lastName;

}

}

Let's create objects person1 and person2 using the Person constructor function.

var person1 = new Person("Bablu", "Ahmed");

var person2 = new Person("Fazlul", "Haque");

during execution of above code, JavaScript engine creates properties firstName, lastName and method fullName for instances person1 and person2 as shown below:

console.log(person1);

**Output**: Person {firstName: "Bablu", lastName: "Ahmed", fullName: ƒ}

console.log(person2);

**Output**: Person {firstName: "Fazlul", lastName: "Haque", fullName: ƒ}

i.e. every object created using the constructor function will have it's own copy of properties and methods. It doesn’t make sense to have two instances of function fullName that do the same thing. Storing separte instances of function for each ojects results into wastage of memory.

Delete a property from an object

To delete a property from an object we can use the ‘delete’ operator. You cannot delete properties that were inherited.

**delete** operator returns true if the delete was successful.

delete person.firstName; // return true

console.log(person);

Let's see what happens if we try to call fullName method which uses both the firstName and lastName property of person object.

console.log(person.fullName());// undefined Ahmed

Output is **undefined** because we were trying to access firstName property of person object which does not exist.

But remember the delete is much more slower. So we can do alternative as follows.

**object[key] = undefined**

For example:

var obj = {

field: 1

};

obj.field = undefined;