JavaScript Object Properties

- An Object is an Unordered Collection of Properties.
- Properties are the most important part of JavaScript objects. Properties can be changed, added, deleted, and some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
// objectName.property
let age = person.age;
```

or

```
//objectName["property"]
let age = person["age"];
```

or

```
//objectName[expression]
let age = person[x];
```

Examples

```
person.firstname + " is " + person.age + " years old.";
```

```
person["firstname"] + " is " + person["age"] + " years old.";
```

```
let x = "firstname";
let y = "age";
person[x] + " is " + person[y] + " years old.";
```

Adding New Properties

Add new properties to an existing object by simply giving it a value:

```
person.nationality = "English";
```

Deleting Properties

The `delete` keyword deletes a property from an object:

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
delete person.age;
```

or delete person["age"];

Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person["age"];
```

Note: The `delete` keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again.

Nested Objects

Property values in an object can be other objects:

Example

```
myObj = {
  name:"John",
  age:30,
  myCars: {
    car1:"Ford",
    car2:"BMW",
    car3:"Fiat"
  }
}
```

You can access nested objects using the dot notation or the bracket notation:

Examples

```
myObj.myCars.car2;

myObj.myCars["car2"];

myObj["myCars"]["car2"];


let p1 = "myCars";
let p2 = "car2";
myObj[p1][p2];
```

JavaScript Object Methods

Object methods are actions that can be performed on objects.

A method is a **function definition** stored as a **property value**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
```

In the example above, `this` refers to the **person object**:
this.firstName means the **firstName** property of **person**.
this.lastName means the **lastName** property of **person**.

Accessing Object Methods

You access an object method with the following syntax:

```
objectName.methodName()
```

If you invoke the **fullName property** with (), it will execute as a **function**:

Example

```
name = person.fullName();
```

If you access the **fullName property** without (), it will return the **function definition**:

Example

```
name = person.fullName;
```

Adding a Method to an Object

Adding a new method to an object is easy:

Example

```
person.name = function () {
  return this.firstName + " " + this.lastName;
};
```

Using JavaScript Methods

This example uses the JavaScript `toUpperCase()` method to convert a text to uppercase:

Example

```
person.name = function () {
  return (this.firstName + " " + this.lastName).toUpperCase();
};
```

JavaScript Display Objects

Displaying a JavaScript object will output **[object Object]**.

Example

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};
```

```
document.getElementById("demo").innerHTML = person;
```

Some solutions to display JavaScript objects are:

- Displaying the Object Properties by name
- Displaying the Object Properties in a Loop
- Displaying the Object using Object.values()
- Displaying the Object using JSON.stringify()

Displaying Object Properties

The properties of an object can be displayed as a string:

Example

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Display Properties
document.getElementById("demo").innerHTML =
person.name + "," + person.age + "," + person.city;
```

Displaying Properties in a Loop

The properties of an object can be collected in a loop:

Example

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Build a Text
let text = "";
for (let x in person) {
  text += person[x] + " ";
};

// Display the Text
document.getElementById("demo").innerHTML = text;
```

Note: You must use **person[x]** in the loop. **person.x** will not work (Because **x** is the loop variable).

Using Object.values()

`Object.values()` creates an array from the property values:

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Create an Array
const myArray = Object.values(person);

// Display the Array
document.getElementById("demo").innerHTML = myArray;
```

Using Object.entries()

`Object.entries()` makes it simple to use objects in loops:

Example

```
const fruits = {Bananas:300, Oranges:200, Apples:500};

let text = "";
for (let [fruit, value] of Object.entries(fruits)) {
  text += fruit + ": " + value + "<br>";
}
```

Using JSON.stringify()

JavaScript objects can be converted to a string with JSON method `JSON.stringify()`.

`JSON.stringify()` is included in JavaScript and supported in all major browsers.

Note: The result will be a string written in JSON notation:

{"name":"John","age":50,"city":"New York"}

Example

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Stringify Object
let myString = JSON.stringify(person);

// Display String
document.getElementById("demo").innerHTML = myString;
```

JavaScript Object Constructors

Sometimes we need to create many objects of the same **type**. To create an **object type** we use an **object constructor function**.

It is considered good practice to name constructor functions with an upper-case first letter.

Object Type Person

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

Note: In the constructor function, `this` has no value. The value of `this` will become the new object when a new object is created.

Now we can use `new Person()` to create many new Person objects:

Example

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
const mySister = new Person("Anna", "Rally", 18, "green");
const mySelf = new Person("Johnny", "Rally", 22, "green");
```

Property Default Values

A **value** given to a property will be a **default value** for all objects created by the constructor:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.nationality = "English";
}
```

Adding a Property to an Object

Adding a property to a created object is easy:

Example

```
myFather.nationality = "English";
```

Note: The new property will be added to **myFather**. Not to any other of **Person Objects**.

Adding a Property to a Constructor

You can **NOT** add a new property to an object constructor:

Example

```
Person.nationality = "English";
```

To add a new property, you must add it to the constructor function prototype:

Example

```
Person.prototype.nationality = "English";
```

Constructor Function Methods

A constructor function can also have **methods**:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.fullName = function() {
    return this.firstName + " " + this.lastName;
  };
}
```

Adding a Method to an Object

Adding a method to a created object is easy:

Example

```
myMother.changeName = function (name) {
  this.lastName = name;
}
```

Note: The new method will be added to **myMother**. Not to any other **Person Objects**.

Adding a Method to a Constructor

You cannot add a new method to an object constructor function. Below code will produce a TypeError:

Example

```
Person.changeName = function (name) {
  this.lastName = name;
}

myMother.changeName("Doe");
```

```
TypeError: myMother.changeName is not a function
```

Adding a new method must be done to the constructor function prototype:

Example

```
Person.prototype.changeName = function (name) {
  this.lastName = name;
}
```

```
myMother.changeName("Doe");
```

Note: The changeName() function assigns the value of `name` to the person's `lastName` property, substituting `this` with `myMother`.

Built-in JavaScript Constructors

JavaScript has built-in constructors for all native objects:

```
new Object()   // A new Object object
new Array()    // A new Array object
new Map()      // A new Map object
new Set()      // A new Set object
new Date()     // A new Date object
new RegExp()   // A new RegExp object
new Function() // A new Function object
```

Note: The `Math()` object is not in the list. `Math` is a global object.
The `new` keyword cannot be used on `Math`.

Best way to create:

- Object literals `{}` instead of `new Object()`.
- Array literals `[]` instead of `new Array()`.
- Pattern literals `/()/` instead of `new RegExp()`.
- Function expressions `() {}` instead of `new Function()`.

Example

```
"";             // primitive string
0;              // primitive number
false;          // primitive boolean
{};            // object object
[];             // array object
/()/            // regexp object
function(){}; // function
```