JavaScript Objects

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

- `string`
- `number`
- `boolean`
- `null`
- `undefined`

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

What is Object :

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword `new`.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using `Object.create()`.

Object Literal :

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

eg:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Objects</h2>

<p>Creating a JavaScript Object:</p>


<p id="demo"></p>
```

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

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years
old.";
</script>

</body>
</html>
```

Note:

Spaces and line breaks are not important. An object definition can span multiple lines:

Eg:2

This example creates an empty JavaScript object, and then adds 4 properties:

```
<!DOCTYPE html>
<html>
```

```
<body>

<h2>JavaScript Objects</h2>
<p>Creating a JavaScript Object:</p>

<p id="demo"></p>

<script>
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years
old.";
</script>
```

```
</body>

</html>
```

Using the JavaScript Keyword new

The following example create a new JavaScript object using `new Object()`, and then adds 4 properties:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Objects</h2>

<p>Creating a JavaScript Object:</p>


<p id="demo"></p>


<script>

const person = new Object();

person.firstName = "John";
```

person.lastName = "Doe";

person.age = 50;

person.eyeColor = "blue";

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>

JavaScript for...in Loop

The JavaScript `for...in` statement loops through the properties of an object.

Syntax

```
for (let variable in object) {
  // code to be executed
}
```

eg:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Looping object property values:</p>

<p id="demo"></p>

<script>
const person = {
  fname:"John",
  lname:"Doe",
  age:25
};

let txt = "";
```

```
for (let x in person) {
  txt += person[x] + " ";
}

document.getElementById("demo").innerHTML =
txt;
</script>

</body>
</html>
```

Note:

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

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Deleting object properties.</p>
```

```
<p id="demo"></p>

<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};

delete person.age;

document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years
old.";
</script>
```

</body>

</html>

Nested Objects

Values in an object can be another object:

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

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

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

<p>Access nested objects:</p>

```
<p id="demo"></p>

<script>
const myObj = {
  name: "John",
  age: 30,
  cars: {
  car1: "Ford",
  car2: "BMW",
  car3: "Fiat"
  }
}
document.getElementById("demo").innerHTML =
myObj.cars.car2;
</script>

</body>
</html>
```

JavaScript Object Methods

Eg:

```
<!DOCTYPE html>
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>
<p>In this example, <b>this</b> refers to the
<b>person</b> object.</p>
<p>Because <b>fullName</b> is a method of the
person object.</p>

<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML =
person.fullName();
</script>

</body>
</html>
```

What is **this**?

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.
In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

Adding a Method to an Object

Adding a new method to an object is easy:

Eg:

<!DOCTYPE html>

<html>

<body>

```
<h2>JavaScript Objects</h2>
<p id="demo"></p>

<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
};
person.name = function() {
  return this.firstName + " " + this.lastName;
};

document.getElementById("demo").innerHTML =
"My father is " + person.name();
</script>
```

</body>

</html>

Note:

Some common 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()

Using Object.values()

Any JavaScript object can be converted to an array using `Object.values()`:

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

const myArray = Object.values(person);
```

`myArray` is now a JavaScript array, ready to be displayed:

eg:

<!DOCTYPE html>

<html>

<body>

```
<h2>JavaScript Objects</h2>
<p>Object.values() converts an object to an
array.</p>

<p id="demo"></p>


<script>
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

document.getElementById("demo").innerHTML =
Object.values(person);
</script>
```

</body>

</html>

Using JSON.stringify()

Any JavaScript object can be stringified (converted to a string) with the JavaScript function `JSON.stringify()`:

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

let myString = JSON.stringify(person);
```

`myString` is now a JavaScript string, ready to be displayed:

The result will be a string following the JSON notation:

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

Stringify Dates

`JSON.stringify` converts dates into strings:

Eg:

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Objects</h2>

<p>JSON.stringify will convert dates into strings:</p>
```

```
<p id="demo"></p>

<script>
var person = {
  name: "John",
  today: new Date()
};

document.getElementById("demo").innerHTML = JSON.stringify(person);
</script>

</body>
</html>
```

By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();

Here, **new keyword** is used to create object.

Eg:

```
<html>

<body>

<script>

var emp=new Object();

emp.id=101;
emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

</body>

</html>
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

```
<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);


document.write(e.id+" "+e.name+" "+e.salary);

</script>

</body>

</html>
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

Eg:

```
<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;


this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}
}
e=new emp(103,"puli",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+"
"+e.salary);
</script>
</body>
</html>
```