JavaScript Objects

JavaScript Objects

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

Here is an example of a JavaScript object.

```
// object
const student = {
    firstName: 'ram',
    class: 10
};
```

Here, student is an object that stores values such as strings and numbers.

JavaScript Object Declaration

The syntax to declare an object is:

```
const object_name = {
   key1: value1,
   key2: value2
}
```

Here, an object object_name is defined. Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}.

For example,

```
// object creation
const person = {
    name: 'John',
    age: 20
};
console.log(typeof person); // object
```

JavaScript Object Properties

In JavaScript, "key: value" pairs are called **properties**. For example,

```
let person = {
    name: 'John',
    age: 20
};
```

Here, name: 'John' and age: 20 are properties.

Accessing Object Properties

You can access the **value** of a property by using its **key**

1. Using dot Notation

Here's the syntax of the dot notation. :
objectName.key

For example,

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person.name); // John
```

2. Using bracket Notation

Here is the syntax of the bracket notation. :
objectName["propertyName"]

For example,

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person["name"]); // John
```

JavaScript Nested Objects

An object can also contain another object. For example,

```
// nested object
const student = {
    name: 'John',
    age: 20,
    marks: {
        science: 70,
        math: 75
    }
}
// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}

// accessing property of marks object
console.log(student.marks.science); // 70
```

In the above example, an object student contains an object value in the marks property.

JavaScript Object Methods

In JavaScript, an object can also contain a function. For example,

```
const person = {
    name: 'Sam',
    age: 30,
    // using function as a value
    greet: function() { console.log('hello') }
}

person.greet(); // hello
```

Here, a function is used as a value for the greet key. That's why we need to use person.greet() instead of person.greet to call the function inside the object.

JavaScript Methods and this Keyword

In JavaScript, objects can also contain functions. For example,

```
// object containing method
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};
```

In the above example, a person object has two keys (name and greet), which have a string value and a function value, respectively.
Hence basically, the JavaScript **method** is an object property that has a function value.

Accessing Object Methods

You can access an object method using a dot notation. The syntax is: **objectName.methodKey()**

You can access property by calling an **objectName** and a **key**. You can access a method by calling an **objectName** and a **key** for that method along with (). For example,

```
// accessing method and property
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};

// accessing property
person.name; // John

// accessing method
person.greet(); // hello
```

Here, the greet method is accessed as person.greet() instead of person.greet.

If you try to access the method with only person.greet, it will give you a function definition.

```
person.greet; // ƒ () { console.log('hello'); }
```

Adding a Method to a JavaScript Object

You can also add a method in an object. For example,

```
// creating an object
let student = { };
// adding a property
student.name = 'John';
// adding a method
student.greet = function() {
    console.log('hello');
}
// accessing a method
student.greet(); // hello
```

In the above example, an empty student object is created. Then, the name property is added. Similarly, the greet method is also added. In this way, you can add a method as well as property to an object.

JavaScript this Keyword

To access a property of an object from within a method of the same object, you need to use the this keyword. Let's consider an example.

```
const person = {
    name: 'John',
    age: 30,

    // accessing name property by using this.name
    greet: function() { console.log('The name is' + ' ' + this.name); }
};

person.greet();
```

Output : The name is John

In the above example, a person object is created. It contains properties (name and age) and a method greet.
In the method greet, while accessing a property of an object, this keyword is used.
In order to access the **properties** of an object, this keyword is used following by . and **key**.

However, the function inside of an object can access it's variable in a similar way as a normal function would. For example,

```
const person = {
    name: 'John',
    age: 30,
    greet: function() {
        let surname = 'Doe';
        console.log('The name is' + ' ' + this.name + ' ' + surname); }
};

person.greet();
```

Output: The name is John Doe

JavaScript Constructor Function

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();
```

In the above example, function Person() is an object constructor function.
To create an object from a constructor function, we use the new keyword.

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
```

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
```

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"
```

JavaScript Getter and Setter

In JavaScript, there are two kinds of object properties:

- Data properties
- Accessor properties

Data Property

Here's an example of data property that we have been using in the previous tutorials.

```
const student = {
    // data property
    firstName: 'Monica';
};
```

Accessor Property

In JavaScript, accessor properties are methods that get or set the value of an object. For that, we use these two keywords:

- get - to define a getter method to get the property value
- set - to define a setter method to set the property value

JavaScript Getter

In JavaScript, getter methods are used to access the properties of an object. For example,

```
const student = {
    // data property
    firstName: 'Monica',
    // accessor property(getter)
    get getName() {
        return this.firstName;
    }
};
// accessing data property
console.log(student.firstName); // Monica
// accessing getter methods
console.log(student.getName); // Monica
// trying to access as a method
console.log(student.getName()); // error
```

JavaScript Setter

In JavaScript, setter methods are used to change the values of an object. For example,

```
const student = {
    firstName: 'Monica',

    //accessor property(setter)
    set changeName(newName) {
        this.firstName = newName;
    }
};

console.log(student.firstName); // Monica

// change(set) object property using a setter
student.changeName = 'Sarah';

console.log(student.firstName); // Sarah
```

JavaScript Object.defineProperty()

In JavaScript, you can also use Object.defineProperty() method to add getters and setters. For example,

```
const student = {
    firstName: 'Monica'
}
// getting property
Object.defineProperty(student, "getName", {
    get : function () {
        return this.firstName;
    }
});

// setting property
Object.defineProperty(student, "changeName", {
    set : function (value) {
        this.firstName = value;
    }
});
console.log(student.firstName); // Monica

// changing the property value
student.changeName = 'Sarah';
console.log(student.firstName); // Sarah
```