Introduction to the JavaScript objects

In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property.

The key of a property can be a string and the value of a property can be any valid JavaScript value e.g., a string, a number, an array, and even a function.

When a function is a property of of an object, it's often called a **method**.

JavaScript provides you with many ways to create a new object. The most popular one is to use the object literal syntax.

The following example creates an empty object using the the object literal syntax:

```
let empty = {};
```

To create an object with properties, you use the `key:value` within the curly braces. For example, the following creates a new `person` object:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

The `person` object has two properties `firstName` and `lastName` with the corresponding values `'John'` and `'Doe'`.

When an object has multiple properties, you use a comma (`,`) to separate them like the above example.

Accessing properties

To access a property of an object, you use one of two notations: the dot notation and array-like notation.

1) The dot notation (.)

The following illustrates how to use the dot notation to access a property of an object:

```
objectName.propertyName
```

For example, to access the `firstName` property of the `person` object, you use the following expression:

```
person.firstName
```

This example creates a `person` object and shows the first name and last name to the console:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(person.firstName);
console.log(person.lastName);
```
Code language: JavaScript (javascript)

2) Array-like notation (`[]`)

The following illustrates how to access the value of an object's property via the array-like notation:

```
objectName['propertyName']
```

For example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(person['firstName']);
console.log(person['lastName']);
```

When a property name contains spaces, you need to place it inside quotes. For example, the following `address` object has the `'building no'` as a property:

```
let address = {
    'building no': 3960,
    street: 'North 1st street',
    state: 'CA',
    country: 'USA'
};
```
Code language: JavaScript (javascript)

To access the `'building no'` property, you need to use the array-like notation:

```
address['building no'];
```
Code language: CSS (css)

If you use the dot notation, you'll get an error:

```
address.'building no';
```
Code language: JavaScript (javascript)

Error:

```
SyntaxError: Unexpected string
```

Note that it's not a good practice to use spaces in the property names of an object.

Reading from a property that does not exist will result in an `undefined`. For example:

```
console.log(address.district);
```

Output:

```
undefined
```

Modifying the value of a property

To change the value of a property, you use the assignment operator (=). For example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

person.firstName = 'Jane';

console.log(person);
```
Code language: JavaScript (javascript)

Output:

```
{ firstName: 'Jane', lastName: 'Doe' }
```
Code language: CSS (css)

In this example, we changed the value of the `firstName` property of the `person` object from `'John'` to `'Jane'`.

Adding a new property to an object

Unlike objects in other programming languages such as Java and C++, you can add a property to an object after object creation.

The following statement adds the `age` property to the `person` object and assigns 25 to it:

```
person.age = 25;
```

Deleting a property of an object

To delete a property of an object, you use the `delete` operator:

```
delete objectName.propertyName;
```

The following example removes the `age` property from the `person` object:

```
delete person.age;
```

If you attempt to access the age property again, you'll get an `undefined` value.

Checking if a property exists

To check if a property exists in an object, you use the `in` operator:

```
propertyName in objectName
```

The `in` operator returns `true` if the `propertyName` exists in the `objectName`.

The following example creates an `employee` object and uses the `in` operator to check if the `ssn` and `employeeId` properties exist in the object:

```
let employee = {
    firstName: 'Peter',
    lastName: 'Doe',
    employeeId: 1
};

console.log('ssn' in employee);
console.log('employeeId' in employee);
```

Output:

```
false
true
```

Iterating over properties of an object using `for...in` loop

To iterate over all properties of an object without knowing property names, you use the `for...in` loop:

```
for(let key in object) {
    // ...
}
```

For example, the following statement creates a `website` object and iterates over its properties using the `for...in` loop:

```
let website = {
    title: 'JavaScript Tutorial',
    url: 'https://www.javascripttutorial.net',
    tags: ['es6', 'javascript', 'node.js', 'reactjs', 'react native']
};

for (const key in website) {
    console.log(key);
}
```

Output:

```
'title'
'url'
'tags'
```

Methods

Besides data, objects can have actions. The actions of objects are known as methods.

The following adds the `greet` action to the `person` object:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

person.greet = function () {
    console.log('Hello!');
}
person.greet();
```

Output:

```
Hello!
```

In this example, we used a function expression to create the function and assigned it to the `greet` property of the `person` object.

Then, we call the function via the `greet` property as `greet()`.

Besides using a function expression, you can define a function and assign it to an object like this:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

function greet() {
    console.log('Hello, World!');
}

person.greet = greet;

person.greet();
```
Code language: JavaScript (javascript)

In this example, the greet is defined as a regular function. The expression `person.greet = greet` assigns the `greet` function to the `greet` property of the `person` object.

Method shorthand

It's possible to define methods of an object using the object literal syntax as shown in the following example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    greet: function () {
        console.log('Hello, World!');
    }
};
```

ES6 provides you with the concise method syntax that allows you to make it shorter to define a method for an object:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    greet() {
        console.log('Hello, World!');
    }
};

person.greet();
```

This syntax looks much cleaner and less verbose.

The `this` value

Typically, methods need to access data stored in the object.

For example, you may want to develop a method that returns the full name of the person object by concatenating the first name and last name.

Inside the method, the `this` value references the object used to invoke the method. Therefore, you can access an object property using the `this` value as follows:

```
this.propertyName
```

The following example uses the `this` value in the `getFullName()` method:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    greet: function () {
        console.log('Hello, World!');
    },
    getFullName: function () {
        return this.firstName + ' ' + this.lastName;
    }
};
```

```
console.log(person.getFullName());
```

Output

```
'John Doe'
```

Summary

- An object is a collection of key-value pairs.
- Use the dot notation (`.`) or array-like notation (`[]`) to access a property of an object.
- The `delete` operator removes a property from an object.
- The `in` operator check if a property exists in an object.
- The `for...in` iterates over properties of an object.
- When functions are the properties of an object, they are called methods.
- Use the `this` inside the method to access the object's properties.