Session 18

Mohamed Emary

May 24, 2024

1 Object

Object in JS is a collection of key-value pairs. Keys are also called properties and values can be any data type.

Objects are used to store multiple values in a single variable, these values are related to each other.

```
var student = {
    name: "Mohamed",
    age: 25,
    id: 12345,
    courses: ["Math", "Physics", "Chemistry"],
};
```

Objects are non-primitive data types, and when you try `console.log(typeof student)` it will return `object`.

Being a *non-primitive* data type means that the object can store **multiple values**, and these values can be of **different data types**.

Printing the object will show all the properties and their values.

```
console.log(student);
// Output:
// {
//   name: 'Mohamed',
//   age: 25,
//   id: 12345,
//   courses: [ 'Math', 'Physics', 'Chemistry' ]
// }
```

To access the properties of an object, you can use the `.` operator.

Note:

When you see the dot operator with any variable, it means that the variable is an object.

```
console.log(student.name); // Mohamed
console.log(student.age); // 25
```

When you try to access a property that doesn't exist, it will return `undefined`.

```
console.log(student.city); // undefined
student.city = "Cairo";    // Set a new property
console.log(student.city); // Get the new property
```

Now when you try to print the object, you will see the new property.

```
console.log(student);
// Output:
// {
//   name: 'Mohamed',
//   age: 25,
//   id: 12345,
//   courses: [ 'Math', 'Physics', 'Chemistry' ],
//   city: 'Cairo'
// }
```

You can put an object inside another object.

```
var student = {
    name: "Mohamed",
    age: 25,
    id: 12345,
    courses: ["Math", "Physics", "Chemistry"],
    address: {
        city: "Cairo",
        street: "Tahrir",
    },
};
```

To access the nested object, you can use the dot operator.

```
console.log(student.address.city);   // Cairo
console.log(student.address.street); // Tahrir
```

You can also put functions inside an object.

```
var student = {
    name: "Mohamed",
    age: 25,
    id: 12345,
    courses: ["Math", "Physics", "Chemistry"],
    address: {
        city: "Cairo",
        street: "Tahrir",
    },
    sayHello: function() {
        console.log("Hello, I'm a student");
    },
};
```

To call the function, you can use the dot operator.

If you don't use the parentheses `()`, it will return the function itself.

```
student.sayHello;
// Output:
// ƒ () {
//   console.log("Hello, I'm a student");
// }
```

To call the function, you should use the parentheses `()`.

```
student.sayHello(); // Hello, I'm a student
```

Using the `sayHello` function inside a `console.log` will print but also return `undefined` because the function doesn't return anything.

```
console.log(student.sayHello());
// Output:
// Hello, I'm a student
// undefined
```

But if you use a function like this:

```
var student = {
    name: "Mohamed",
    age: 25,
    id: 12345,
    courses: ["Math", "Physics", "Chemistry"],
    address: {
        city: "Cairo",
        street: "Tahrir",
    },
    sayHello: function() {
        return `Hello, I'm ${student.name}`;
    },
};
```

It will return the string without `undefined`.

```
console.log(student.sayHello());
// Output:
// Hello, I'm Mohamed
```

Note:

When you put a function inside an object, it's called a **method**.

2 JS Built-in Objects

JavaScript has many built-in objects like `window`, `document`, `console`.

2.1 `window`

`window` is a super global object that contains all global variables, functions, and objects.

Example functions in the `window` object are `alert`, `prompt`.

```
window.alert("Hello, World!");
window.prompt("What's your name?");
```

Since `window` is the super global object, you can access its properties and methods without using the `window` keyword.

This will also work:

```
alert("Hello, World!");
prompt("What's your name?");
```

2.2 `document`

`document` is another object that we can use to manipulate the HTML document.

```
document.getElementById("id");
```

Note:

Try this code:
`document.getElementById("id").innerHTML = "Hello, World!";`

Why this code works? isn't `getElementById` a method not an object?
That is because the method `getElementById` returns an object so using the dot operator `.` allows you to access the properties of the object that the method returns.

2.3 `console`

`console` is also an object that has a method `log` that we use to print messages.

```
console.log("Hello, World!");
```

2.4 `Math`

`math` is an object that has many properties and methods to perform mathematical operations.

```
console.log(Math.PI); // 3.141592653589793
console.log(Math.round(4.7)); // 5
console.log(Math.floor(4.7)); // 4
console.log(Math.ceil(4.4)); // 5
console.log(Math.pow(2, 3)); // 8
console.log(Math.sqrt(64)); // 8
console.log(Math.abs(-4.7)); // 4.7
console.log(Math.min(0, 150, 30, 20, -8, -200)); // -200
console.log(Math.max(0, 150, 30, 20, -8, -200)); // 150
console.log(Math.random()); // Random number between 0 and 1
```

Mathematical expressions of the above methods:

- `Math.PI` = π
- `Math.round(4.7)` ≈ 5
- `Math.floor(4.7)` = $\lfloor 4.7 \rfloor = 4$
- `Math.ceil(4.4)` = $\lceil 4.4 \rceil = 5$

- `Math.pow(2, 3)` = $2^3 = 8$
- `Math.sqrt(64)` = $\sqrt{64} = 8$
- `Math.abs(-4.7)` = $|-4.7| = 4.7$

```
console.log(Math.round(Math.random() * 10)); // Random number between 0 and
↪  10
```

Example to show a random number with a button click:

In HTML body:

```
  <div>
      <p id="rand"></p>
      <button id="btn">Get a random Number</button>
  </div>
  <script src="./script.js"></script>
```

In `script.js`:

```
var btn = document.getElementById("btn");
btn.onclick = getRand;
function getRand() {
  var p = document.getElementById("rand");
  p.innerHTML = Math.round(Math.random() * 10);
}
```

3 Array

Array is a collection of elements. Elements can be of any data type.

Array is a special type of object:

```
var nums = [1, 2, 3, 4, 5];
console.log(typeof nums); // object
```

And since objects are non-primitive data types, arrays can store multiple values of different data types.

Array syntax is a pair of square brackets `[]` with elements separated by commas.

```
var fruits = ["Apple", "Banana", "Orange"];
var mix = [
    [1, 2, 3],
    'Apple',
    25,
    true,
    function() { return "Hello, World!"; },
    { name: 'Mohamed', age: 25 }
];

console.log(mix[0][1]); // 2
console.log(mix[1]);    // Apple
console.log(mix[4]);    // [Function]
console.log(mix[4]());  // Hello, World!
console.log(mix[5].name); // Mohamed
```

Suppose you have multiple products and each of these products is an object, and you want to store all these products in one variable, you can use an array.

```
var products = [
    { name: "Apple", price: 10 },
    { name: "Banana", price: 5 },
    { name: "Orange", price: 7 },
];
```

Each item in the array has an index starting from 0 **(not 1)** and you can access the elements using that index.

```
var fruits = ["Apple", "Banana", "Orange"];
console.log(fruits[0]); // Apple
console.log(fruits[2]); // Orange
console.log(fruits);    // ["Apple", "Banana", "Orange"]
```

`fruits[0]` is read as "fruits of 0".

To print each item in the array, you can use a loop.

```
var fruits = ["Apple", "Banana", "Orange"];
for (var i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}
```

`fruits.length` returns the number of elements in the array, and since the last index is `fruits.length - 1`, the loop should run from `0` to `fruits.length - 1` so we use `i < fruits.length` instead of `i <= fruits.length` in the loop condition.

4 Object vs Array

	Object	Array
Type	Object & Non-primitive	Object & Non-primitive
Syntax	`{ prop1: val1, prop2: val2, ... }`	`[ elem1, elem2, ... ]`
Element Access	`object.prop`	`array[index]`
Use Case	Store different properties of an element	Store multiple elements
Index	Key	Number
Example	`{ name: 'Mohamed', age: 25 }`	`['Apple', 'Banana']`

See the execrise at the end of video 7: `object vs array & exercise`

5 Functional Programming in JS

JavaScript applies functional programming concepts like:

1. You can assign a function to a variable.

```
var x = function() {
    return "Hello, World!";
};
```

2. Functions can be properties of objects.

```
var obj = {
    sayHello: function() { // method
        return "Hello there";
    },
};
```

3. Functions can be returned from another function.

```
function twoNumAvg(sum) {
    return sum / 2;
}

function getAvg(a, b) {
    var sum = a + b;
    return twoNumAvg(sum);
}

console.log(getAvg(10, 20)); // 15
```

4. Functions can be passed as arguments to other functions.

```
function twoNumAvg(sum) {
    return sum / 2;
}

function sum(a, b) {
    return a + b;
}

console.log(twoNumAvg(sum(10, 20))); // 15
```

6 Summary

Objects:

- Object is a collection of key-value pairs.
- Objects are used to store multiple values in a single variable.
- Objects are non-primitive data types.
- Objects can store multiple values of different data types.

Arrays:

- Array is a collection of elements.
- Array is a special type of object.
- Arrays can store multiple values of different data types.

Built-in Objects:

- `window` is a super global object.
- `document` is an object that we can use to manipulate the HTML document.
- `console` is an object that has a method `log` to print messages.
- `Math` is an object that has many properties and methods to perform mathematical operations.

Functional Programming:

- JavaScript applies functional programming concepts.
    - Functions can be assigned to a variable.
    - Functions can be properties of objects.
    - Functions can be returned from another function.
    - Functions can be passed as arguments to other functions.