


Day :11 JavaScript Notes.

JavaScript Objects

Real Life Objects

In real life, **objects** are things like: houses, cars, people, animals, or any other subjects.

Here is a **car object** example:

Car Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

Object Properties

A real life car has **properties** like weight and color:

`car.name = Fiat`, `car.model = 500`, `car.weight = 850kg`, `car.color = white`.

Car objects have the same **properties**, but the **values** differ from car to car.

Object Methods

A real life car has **methods** like start and stop:

`car.start()`, `car.drive()`, `car.brake()`, `car.stop()`.

Car objects have the same **methods**, but the methods are performed **at different times**.

JavaScript Variables

JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

Example

```
let car = "Fiat";
```

JavaScript Objects

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to an **object** named car:

Example

```
const car = {type:"Fiat", model:"500", color:"white"};
```

JavaScript Object Definition

How to Define a JavaScript Object

- Using an Object Literal
- Using the new Keyword
- Using an Object Constructor

1} JavaScript Object Literal

An object literal is a list of **name:value** pairs inside curly braces {}.

```
{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}
```

Note:

name:value pairs are also called **key:value pairs**.

object literals are also called **object initializers**.

Creating a JavaScript Object

These examples create a JavaScript object with 4 properties:

Examples

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

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

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

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

```
// Create an Object
const person = {};

// Add Properties
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

2} Using the new Keyword

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

Example

```
// Create an Object
const person = new Object();

// Add Properties
person.firstName = "John";
```

```
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

Note:

The examples above do exactly the same.

But, there is no need to use `new Object()`.

For readability, simplicity and execution speed, use the **object literal** method.

Object Properties

The **named values**, in JavaScript objects, are called **properties**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Objects written as name value pairs are similar to:

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

objectName["propertyName"]

Examples

```
person.lastName;
```

```
person["lastName"];
```

JavaScript Object Methods

Methods are **actions** that can be performed on objects.

Methods are **function definitions** stored as **property values**.

Property	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**.

In JavaScript, Objects are King.

If you Understand Objects, you Understand JavaScript.

Objects are containers for **Properties** and **Methods**.

Properties are named **Values**.

Methods are **Functions** stored as **Properties**.

Properties can be primitive values, functions, or even other objects.

In JavaScript, almost "everything" is an object.

Objects are objects

Maths are objects

Functions are objects

Dates are objects

Arrays are objects

Maps are objects

Sets are objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

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

3.14 is a primitive value

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

JavaScript defines 7 types of primitive data types:

- string
- number
- boolean
- null
- undefined
- symbol
- bigint

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

```
const x = person;
```

The object x is **not a copy** of person. The object x **is** person.

The object x and the object person share the same memory address.

Any changes to x will also change person:

Example

```
//Create an Object
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}
```

```
// Try to create a copy
```

```
const x = person;
```

```
// This will change age in person:
```

```
x.age = 10;
```

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

You can add new properties to an existing object by simply giving it a value:

Example

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

Deleting Properties

The delete keyword deletes a property from an object:

Example

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

```
delete person.age;
```

```
or 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

How to Display JavaScript 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()

1} 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;
```

2} 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).

3} 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;
```

4} 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>";
}
```

5} 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;
```

3} JavaScript Object Constructors

Object Constructor Functions

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: 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 **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.

This 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");
```