Unit-4: JavaScript Objects :

4.1 Creating object :

(By object literal, By creating instance of Object,

By using an object constructor)

4.2 Date object :

4.2.1 Date constructor: Date(), Date(milliseconds), Date(dateString), Date(year, month, day, hours,minutes, seconds, milliseconds)

4.2.2 Date Methods: getDate(), getDay(),getMonth(), getHours(),setDate, setMonth(),setDay(), toString()

4.3 Document Object Model (DOM):

4.3.1 DOM concepts

4.3.2 DOM properties

4.3.3 DOM methods :

write(), writeln(),getElementById(),getElementsByName()

4.1 Creating object:

The **this** Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the **person object** that "owns"
the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

With JavaScript, you can define and create your own 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().

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

```
<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>
```

JavaScript Objects

Creating a JavaScript Object:

John is 50 years old.

By creating instance of Object

```
<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 Object Constructors

Example

```
<html>
<body>

<h2>JavaScript Object Constructors</h2>

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

<script>
```

```
// Constructor function for Person objects
function Person(first, last, age, eye) {
this.firstName = first;
this.lastName = last;
this.age = age;
this.eyeColor = eye;
}

// Create a Person object
Const myFather = new Person("John", "Doe", 50, "blue");

// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ".";
</script>

</body>
</html>
```

JavaScript Object Constructors

My father is 50.

4.2 Date object :

4.2.1 Date constructor: Date(), Date(milliseconds), Date(dateString), Date(year, month, day, hours,minutes, seconds, milliseconds)

4.2.2 Date Methods: getDate(), getDay(),getMonth(), getHours(),setDate, setMonth(),setDay(), toString()

4.2.1 Date constructor: Date (), Date (milliseconds), Date (date String), Date(year, month, day, hours, minutes, seconds, milliseconds)

Creating Date Objects

Date objects are created with the new Date() constructor.

There are **4 ways** to create a new date object:

new Date()
new Date(*year*, *month*, *day*, *hours*, *minutes*, *seconds*, *milliseconds*)
new Date(*milliseconds*)
new Date(*date string*)

new Date()

new Date() creates a new date object with the **current date and time**:

```
<html>
<body>

<h2>JavaScript new Date()</h2>
<p>Using new Date(), creates a new date object with the current date and time:</p>

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

<script>
const d = new Date();
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

OUTPUT:
Tue Aug 03 2021 11:28:48 GMT+0530 (India Standard Time)

New Date (*year, month,day, hours, minutes, seconds, milliseconds*)

```
<html>

<body>

<h2>JavaScript new Date()</h2>

<p>Using new Date(7 numbers), creates a new date object with the specified date and time:</p>

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

<script>

const d = new Date(2018, 11, 24, 10, 33, 30, 0);

document.getElementById("demo").innerHTML = d;

</script>

</body>

</html>
```

Note: JavaScript counts months from 0 to 11.

January is 0. December is 11.

6 numbers specify year, month, day, hour, minute, second:

Example

```
const d = new Date(2018, 11, 24, 10, 33, 30);
```

new Date(*dateString*)

new Date(dateString) creates a new date object from a **date string**:

Example

```
<html>
<body>
<h2>JavaScript new Date()</h2>
<p>A Date object can be created with a specified date and time:</p>
<p id="demo"></p>
<script>
const d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

OUTPUT:

JavaScript new Date()

A Date object can be created with a specified date and time:

Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)

new Date(*milliseconds*)

new Date(*milliseconds*) creates a new date object as **zero time plus milliseconds**:

Example

```
<html>
<body>
```

```
<p id="demo"></p>
<script>
const d = new Date(0);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

OUTPUT:

JavaScript new Date()

Using new Date(milliseconds), creates a new date object as January 1, 1970, 00:00:00 Universal Time (UTC) plus the milliseconds:

Thu Jan 01 1970 05:30:00 GMT+0530 (India Standard Time)

4.2.2 Date Methods: getDate(), getDay(),getMonth(), getHours(),setDate, setMonth(),setDay(), toString()

getDate(): Get the day as a number (1-31)

Example

```
<html>
<body>
<h2>JavaScript getDate()</h2>
<p>The getDate() method returns the day of a date as a number (1-31):</p>
<p id="demo"></p>
<script>
```

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getDate();
</script>
</body>
</html>
```

getDay(): Get the weekday as a number (0-6).

Example

```
<html>
<body>
<h2>JavaScript getDay()</h2>
<p>The getDay() method returns the weekday as a number:</p>
<p id="demo"></p>
<script>
const d = new Date();
document.getElementById("demo").innerHTML = d.getDay();
</script>
</body>
</html>
```

getMonth (): Get the **month** as a number (0-11)

Example

```
<html>
<body>
<h2>JavaScript getMonth()</h2>
```

```
<p>The getMonth() method returns the month of a date as a number from 0 to 11.</p>

<p>To get the correct month, you must add 1:</p>

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

<script>

const d = new Date();

document.getElementById("demo").innerHTML = d.getMonth() + 1;

</script>

</body>

</html>
```

getHours (): Get the **hour** (0-23)

Example

```
<html>
<body>
<h2>JavaScript getHours()</h2>
<p>The getHours() method returns the hours of a date as a number (0-23):</p>
<p id="demo"></p>
<script>
const d = new Date();
document.getElementById("demo").innerHTML = d.getHours();
</script>
</body>
</html>
```

setDate()

The setDate() method sets the day of a date object (1-31):

Example

```
<html>
<body>
<h2>JavaScript setDate()</h2>
<p>The setDate() method sets the day of a date object:</p>
<p id="demo"></p>
<script>
const d = new Date();
d.setDate(15);
document.getElementById("demo").innerHTML = d;
</script>
</body>
</html>
```

setMonth ()

The setMonth() method sets the month of a date object (0-11):

Example

```
<html>
<body>
<h2>JavaScript setMonth()</h2>
<p>The setMonth() method sets the month of a date object.</p>
<p>Note that months count from 0. December is month 11:</p>
<p id="demo"></p>
<script>
const d = new Date();
```

```
d.setMonth(11);
document.getElementById("demo").innerHTML = d;
</script>
</body>
</html>
```

toString ()

Definition and Usage

The toString() method returns the value of a string.

Syntax

string.toString()

```
<html>
<body>
<h2>JavaScript Strings</h2>
<p>The toString() method returns the value of a string.</p>
<p id="demo"></p>
<script>
let str = "Hello World!";
document.getElementById("demo").innerHTML = str.toString();
</script>
</body>
</html>
```

4.3 Document Object Model (DOM):

4.3.1 DOM concepts

4.3.2 DOM properties

4.3.3 DOM methods :

write(), writeln(),getElementById(),getElementsByName()

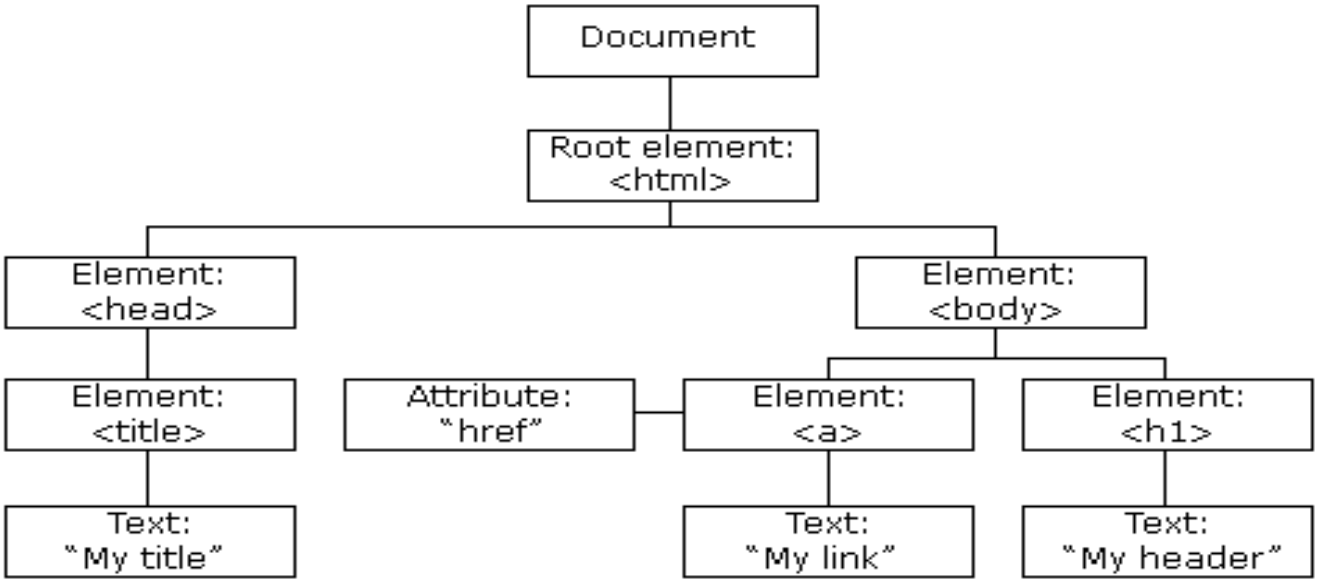
4.3.1 DOM concepts

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

4.3.2 DOM properties

These are some typical DOM properties:

- x.innerHTML - the inner text value of x (a HTML element)
- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

Note: In the list above, x is a node object (HTML element).

innerHTML

The easiest way to get or modify the content of an element is by using the innerHTML property.

innerHTML is not a part of the W3C DOM specification. However, it is supported by all major browsers.

The innerHTML property is useful for returning or replacing the content of HTML elements (including <html> and <body>), it can also be used to view the source of a page that has been dynamically modified.

Example

The JavaScript code to get the text from a <p> element with the id "intro" in a HTML document:

```
txt=document.getElementById("intro").innerHTML
```

After the execution of the statement, txt will hold the value "Hello World!"

Explained:

- **document** - the current HTML document
- **getElementById("intro")** - the <p> element with the id "intro"
- **innerHTML** - the inner text of the HTML element

In the example above, getElementById is a method, while innerHTML is a property.

```
<html>
<body>
<p id="intro">Hello World!</p>

<script type="text/javascript">
txt=document.getElementById("intro").innerHTML;
document.write("The text from the intro paragraph: " + txt);
</script>

</body>
</html>
```

childNodes and nodeValue

The W3C DOM specified way of getting to the content of an element works like this:

The JavaScript code to get the text from a <p> element with the id "intro" in a HTML document:

txt=document.getElementById("intro").childNodes[0].nodeValue

After the execution of the statement, txt will hold the value "Hello World!"

Explained:

- **document** - the current HTML document
- **getElementById("intro")** - the <p> element with the id "intro"
- **childNodes[0]** - the first child of the <p> element (the text node)
- **nodeValue** - the value of the node (the text itself)

In the example above, getElementById is a method, while childNodes and nodeValue are properties.

```
<html>
<body>
<p id="intro">Hello World!</p>
<p id="main">This is an example for the HTML DOM tutorial.</p>
<script type="text/javascript">
txt=document.getElementById("intro").childNodes[0].nodeValue;
document.write("The text from the intro paragraph: " + txt);
</script>
</body>
</html>
```

4.3.3 DOM methods :

write(), writeln(),getElementById(),getElementsByName()

1. **write()**

Definition and Usage

The write() method writes HTML expressions or JavaScript code to a document.

The write() method is mostly used for testing: If it is used after an HTML document is fully loaded, it will delete all existing HTML.

Note: When this method is not used for testing, it is often used to write some text to an output stream opened by the document.open() method. See "More Examples" below.

Tip: The document.writeln() method is similar to write(), only it adds a newline character after each statement.

Syntax

document.write(*exp1, exp2, exp3, ...*)

Parameter Values

Parameter	Description
exp1, exp2, exp3, ...	Optional. What to write to the output stream. Multiple arguments can and they will be appended to the document in order of occurrence

```
<html>
<body>

<script>
document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");
</script>

</body>
</html>
```

2. writeln()

Definition and Usage

The writeln() method is identical to the document.write() method, with the addition of writing a newline character after each statement.

Syntax

document.writeln(*exp1*, *exp2*, *exp3*, ...)

Parameter Values

Parameter	Description
exp1, *exp2*, *exp3*, ...	Optional. What to write to the output stream. Multiple arguments can be they will be appended to the document in order of occurrence

Example:

```
<html>
<body>

<p>Note that write() does NOT add a new line after each statement:</p>

<pre>
<script>
document.write("Hello World!");
document.write("Have a nice day!");
</script>
</pre>
```

```
<p>Note that writeln() add a new line after each statement:</p>

<pre>
<script>
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</pre>

</body>
</html>
```

Output:

Note that write() does NOT add a new line after each statement:

Hello World!Have a nice day!

Note that writeln() add a new line after each statement:

Hello World!
Have a nice day!

3. getElementById()

Definition and Usage

The getElementById() method returns the element that has the ID attribute with the specified value.

This method is one of the most common methods in the HTML DOM, and is used almost every time you want to manipulate, or get info from, an element on your document.

Returns *null* if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code

Syntax

document.getElementById(*elementID*)

Parameter Values

Parameter	**Type**	**Description**
elementID	String	Required. The ID attribute's value of the element you want to get

```
<html>
<body>
<p id="demo">Click the button to change the color of this paragraph.</p>
<button onclick="myFunction()">Try it</button>
<script>
Function myFunction() {
var x = document.getElementById("demo");
x.style.color = "red";
}
```

```
</script>
</body>
</html>
```

4. getElementsByName()

Definition and Usage

The getElementsByName() method returns a collection of all elements in the document with the specified name (the **value** of the name attribute), as an HTMLCollection object.

The HTMLCollection object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

Tip: You can use the length property of the HTMLCollection object to determine the number of elements with the specified name, then you can loop through all elements and extract the info you want.

Syntax

document.getElementsByName(*name*)

Parameter Values

Parameter	Type	Description
name	String	Required. The name attribute value of the element you want to access/manipulate

```
<html>
<body>
Cats:
<input name="animal" type="checkbox" value="Cats">
```

```
Dogs:
<input name="animal" type="checkbox" value="Dogs">
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
Function myFunction() {
var x = document.getElementsByName("animal").length;
document.getElementById ("demo").innerHTML = x;
}
</script>
</body>
</html>
```