JAVA SCRIPT DOM

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

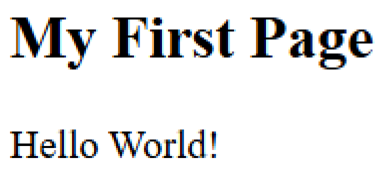
The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).



My First Page

Hello World!

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

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

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

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

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

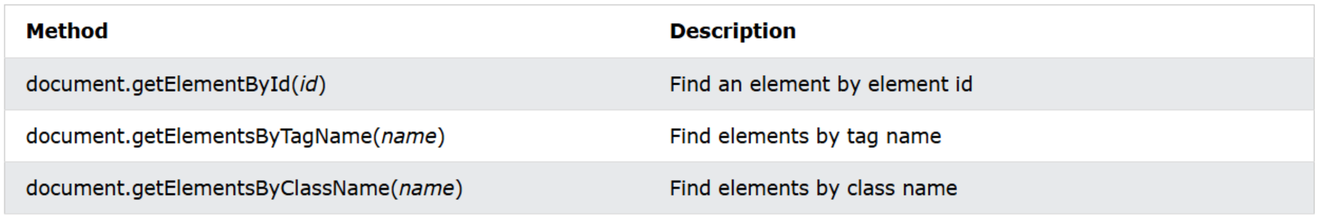
The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

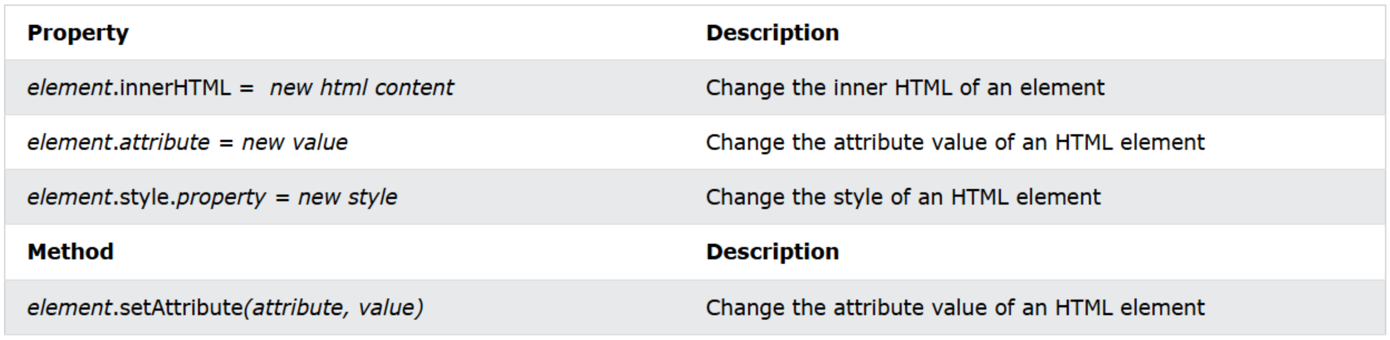
Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements



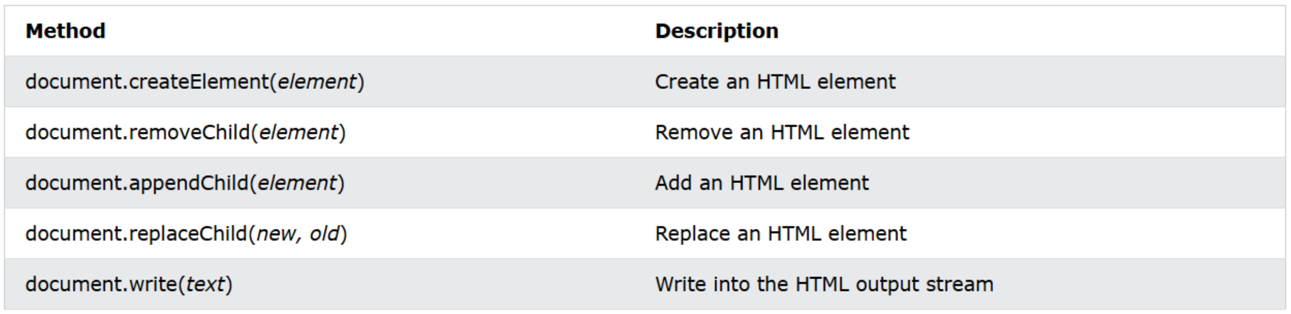
Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements



Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding and Deleting Elements



Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

JavaScript HTML DOM Elements

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>

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

<script>
const element = document.getElementById("intro");

document.getElementById("demo").innerHTML =
"The text from the intro paragraph is: " + element.innerHTML;
```

```
</script>

</body>
</html>
```

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b>
method.</p>

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

<script>
const element = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = 'The text in first paragraph
(index 0) is: ' + element[0].innerHTML;

</script>

</body>
```

</html>

JavaScript HTML DOM - Changing HTML

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing HTML Content

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

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a `<p>` element:

Example

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`

- A JavaScript changes the content (`innerHTML`) of that element to "New text!"

This example changes the content of an `<h1>` element:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an `<h1>` element with `id="id01"`
- We use the HTML DOM to get the element with `id="id01"`
- A JavaScript changes the content (`innerHTML`) of that element to "New Heading"

Dynamic HTML content

JavaScript can create dynamic HTML content:

Date : Tue Dec 03 2024 18:30:19 GMT+0530 (India Standard Time)

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.getElementById("demo").innerHTML = "Date : " + Date(); </script>
```

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

document.write()

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>

<html>

<body>

<p>Bla bla bla</p>

<script>

document.write(Date());

</script>

<p>Bla bla bla</p>

</body>

</html>
```

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

JavaScript HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

The following example changes the style of a `<p>` element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

You will learn more about events in the next chapter of this tutorial.

This example changes the style of the HTML element with `id="id1"`, when the user clicks a button:

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

JavaScript HTML DOM Animation

A Basic Web Page

To demonstrate how to create HTML animations with JavaScript, we will use a simple web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript Animation</h1>

<div id="animation">My animation will go here</div>

</body>
</html>
```

Create an Animation Container

All animations should be relative to a container element.

Example

```
<div id ="container">
  <div id ="animate">My animation will go here</div>
</div>
```

Style the Elements

The container element should be created with style = "position: relative".

The animation element should be created with style = "position: absolute".

Example

```
<!Doctype html>
<html>
<style>
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
</style>
<body>
<h2>My First JavaScript Animation</h2>
```

<div id="container">

<div id="animate"></div>

</div>

</body>

</html>

Animation Code

JavaScript animations are done by programming gradual changes in an element's style.

The changes are called by a timer. When the timer interval is small, the animation looks continuous.

The basic code is:

```
id = setInterval(frame, 5);

function frame() {
  if (/* test for finished */) {
    clearInterval(id);
  } else {
    /* code to change the element style */
  }
}
```

Create the Full Animation Using JavaScript

<!DOCTYPE html>

<html>

<style>

#container {

  width: 400px;

  height: 400px;

  position: relative;

  background: yellow;

```
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: red;
}
</style>
<body>
<p><button onclick="myMove()">Click Me</button></p>
<div id ="container">
  <div id ="animate"></div>
</div>
<script>
function myMove() {
  let id = null;
  const elem = document.getElementById("animate");
  let pos = 0;
  clearInterval(id);
  id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
    }
  }
}
</script>
</body>
</html>
```

JavaScript HTML DOM Events

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

`onclick=`*`JavaScript`*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
</body>
```

</html>

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

In the example above, a function named `displayDate` will be executed when the button is clicked.

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script
```

In the example above, a function named `displayDate` is assigned to an HTML element with the `id="myBtn"`.

The function will be executed when the button is clicked.

The onload and onunload Events

The `onload` and `onunload` events are triggered when the user enters or leaves the page.

The `onload` event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onmouseover and onmouseout Events

The `onmouseover` and `onmouseout` events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript HTML Events</h1>

<h2>The onmouseover Attribute</h2>


<div onmouseover="mOver(this)" onmouseout="mOut(this)"

style="background-color:#D94A38;width:120px;height:20px;padding:40px;">

Mouse Over Me</div>

<script>

function mOver(obj) {

  obj.innerHTML = "Thank You"

}

function mOut(obj) {

  obj.innerHTML = "Mouse Over Me"

}

</script>

</body>

</html>
```

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript HTML Events</h1>

<h2>The onmousedown Attribute</h2>


<div onmousedown="mDown(this)"
onmouseup="mUp(this)"

style="background-
color:#D94A38;width:90px;height:20px;padding:40px;">

Click Me</div>


<script>

function mDown(obj) {

  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Release Me";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
  obj.innerHTML="Thank You";
}
</script>
</body>
</html>
```

JavaScript HTML DOM EventListener

The addEventListener() method

Example

Add an event listener that fires when a user clicks a button:

```
document.getElementById("myBtn").addEventListener("
click", displayDate);
```

The `addEventListener()` method attaches an event handler to the specified element.

The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

Syntax

```
element.addEventListener(event, function, useCapture);
```

The first parameter is the type of the event (like "`click`" or "`mousedown`")

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", function(){ alert("Hello World!"); })
```

You can also refer to an external "named" function:

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

You can also refer to an external "named" function:

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {

  alert ("Hello World!");

}
```