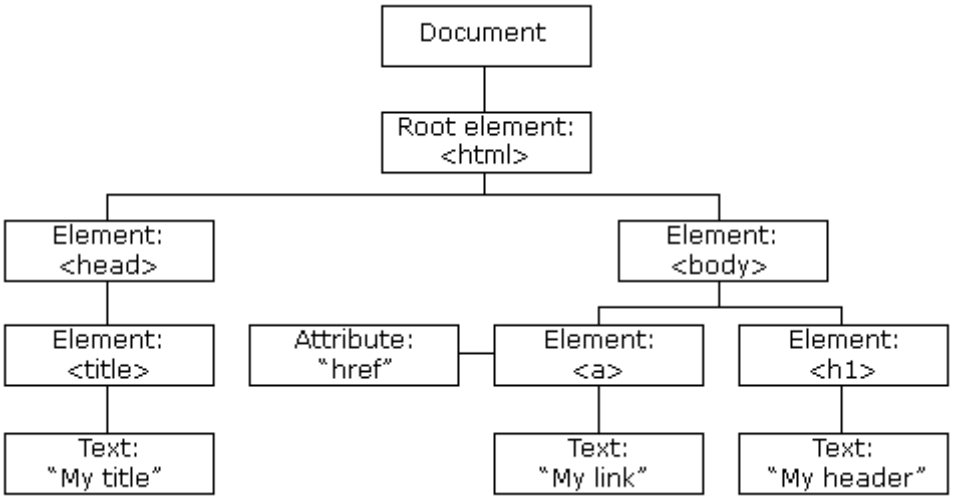
LAB 5: INTRODUCTION TO DOM

What is the HTML DOM (Document Object Model)?

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.** With the HTML DOM, JavaScript can access and change all the elements of an HTML document. 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**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes

- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements). Where HTML DOM properties are **values** (of HTML Elements) that you can set or change. As we have seen in the LAB3, the following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<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 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

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

Example 1: Find elements by tag name

```
<!DOCTYPE html>

<html>

<body>

<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>

<p>This example demonstrates the <b>getElementsByTagName</b>
method.</p>

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

<script>

var x =
document.getElementsByTagName("p");document.getElementById("demo").inne
rHTML =

'The text in first paragraph (index 0) is: ' + x[0].innerHTML;
</script> </body> </html>
```

Example 2: Find elements by Class name

```
<!DOCTYPE html>
<html>
<body>
<h2>Finding HTML Elements by Class Name</h2>
<p>Hello World!</p>
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the
<b>getElementsByClassName</b> method.</p>
<p id="demo"></p>
<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
</body>
</html>
```

Example 3: Find elements by CSS Selectors

```
<!DOCTYPE html>
<html>
<body>
<h2>Finding HTML Elements by Query Selector</h2>
<p>Hello World!</p>
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b>
method.</p>
```

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

<script>

var x = document.querySelectorAll("p.intro");

document.getElementById("demo").innerHTML =

'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;

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

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:

```
<!DOCTYPE html>

<html>

<body>

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

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

<script>

document.getElementById("p2").style.color = "blue";

document.getElementById("p2").style.fontFamily = "Arial";

document.getElementById("p2").style.fontSize = "larger";

</script>

<p>The paragraph above was changed by a script.</p>

</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>
```

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>
```

In this example, a function is called from the event handler:

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="changeText(this)">Click on this text!</h1>
<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

HTML Event Attributes

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

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

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

</body>
</html>
```

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:

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

<p>Click "Try it" to execute the displayDate() function.</p>

<button id="myBtn">Try it</button>

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

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

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

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>

<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>
```

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

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- `parentNode`
- `childNodes[nodenumber]`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`

Child Nodes and Node Values

A common error in DOM processing is to expect an element node to contain text.

Example:

```
<title id="demo">DOM Tutorial</title>
```

The element node `<title>` (in the example above) does **not** contain text.

It contains a **text node** with the value "DOM Tutorial".

The value of the text node can be accessed by the node's `innerHTML` property:

```
var myTitle = document.getElementById("demo").innerHTML;
```

Accessing the innerHTML property is the same as accessing the `nodeValue` of the first child:

```
var myTitle = document.getElementById("demo").firstChild.nodeValue;
```

Accessing the first child can also be done like this:

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

nnerHTML

In this tutorial we use the innerHTML property to retrieve the content of an HTML element.

However, learning the other methods above is useful for understanding the tree structure and the navigation of the DOM.

DOM Root Nodes

There are two special properties that allow access to the full document:

- `document.body` - The body of the document
- `document.documentElement` - The full document

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.body</b> property.</p>
</div>

<script>
alert(document.body.innerHTML);
</script>

</body>
</html>
```

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates
the <b>document.documentElement</b> property.</p>
</div>

<script>
alert(document.documentElement.innerHTML);
</script>

</body>
</html>
```

The nodeName Property

The `nodeName` property specifies the name of a node.

- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<!DOCTYPE html>
<html>
<body>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
</body>
</html>
```

This code creates a new `<p>` element:

```
var para = document.createElement("p");
```

To add text to the `<p>` element, you must create a text node first. This code creates a text node:

```
var node = document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the `<p>` element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
var element = document.getElementById("div1");
```

This code appends the new element to the existing element:

```
element.appendChild(para);
```

Creating new HTML Elements - insertBefore()

The `appendChild()` method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the `insertBefore()` method:

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
</body>
</html>
```

Removing Existing HTML Elements

To remove an HTML element, use the `remove()` method:

Example

```
<!DOCTYPE html>
<html>
<body>
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<button onclick="myFunction()">Remove Element</button>
<script>
function myFunction() {
  var elmnt = document.getElementById("p1");
  elmnt.remove();
}
</script>
</body>
</html>
```

The HTML document contains a `<div>` element with two child nodes (two `<p>` elements):

```
<div>
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

Find the element you want to remove:

```
var element = document.getElementById("p1");
```

Then execute the remove() method on that element:

```
element.remove();
```

ExO1:
Create in an empty html page, 3 buttons and an empty div tag.
Clicking on the 1st button should display an input window in which you will enter a last name. Clicking this button displays the following information:

Hello name!
You are a student in Bachelor2 at CYTech
Your browser is: ……..navig
Your operating system is:……. SE

Each click must add a student in a block with a black border, a gray background, written in green with constant words in bold and variables in italics.
The 2nd button allows you to delete each entry, one by one. Finally, the 3rd button

resets the vacuum zone.

EXO2: Record store

The purpose of this exercise is to manage the shopping cart of an online record sales site, which is present as follows:

- A "source" block, on the left, which contains the discs that can be purchased, each being represented by an image.
- A "destination" block on the right, which contains the discs selected for purchase.
- A "cart" block, centered at the top, which indicates the total amount to be paid.

Each time you click on an image in the source block, this will have the effect of send it to the destination block (accompanied by a text giving the title of the disc) and increase the total amount in the shopping cart block. Conversely, each time we click on an image of the destination block, this will have the effect of returning it to the source block, delete the accompanying text, and decrease the total sum of the shopping cart block.