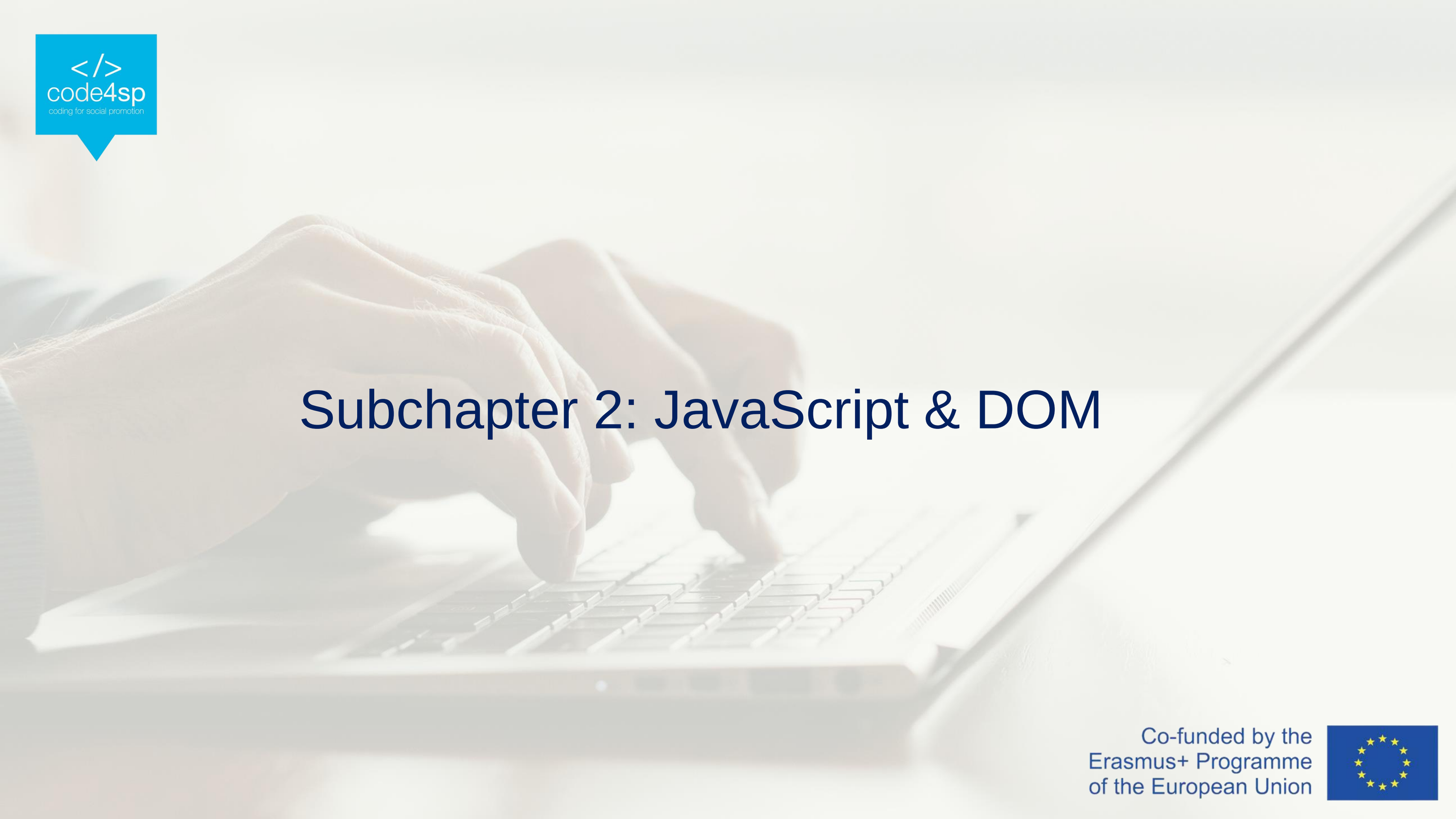
Co-funded by the
Erasmus+ Programme
of the European Union

code4sp
coding for social promotion

JavaScript Trainer Materials
Subchapter 2 – JavaScript & DOM

WP3: Code4SP Training Materials

Prepared by: CITIZENS IN POWER



Subchapter 2: JavaScript & DOM

Co-funded by the
Erasmus+ Programme
of the European Union

What is the Document Object Model (DOM)?

DOM is created by the browser when a web page is loaded in HTML or XML documents. It is used to define the logical structure of these documents and to access and alter their elements.

In this subchapter, we will focus on the HTML DOM which can be used to access and manipulate HTML documents via JavaScript.

The DOM is constructed as a hierarchical tree of objects that include all parts of an HTML document such as elements, attributes, text, etc.

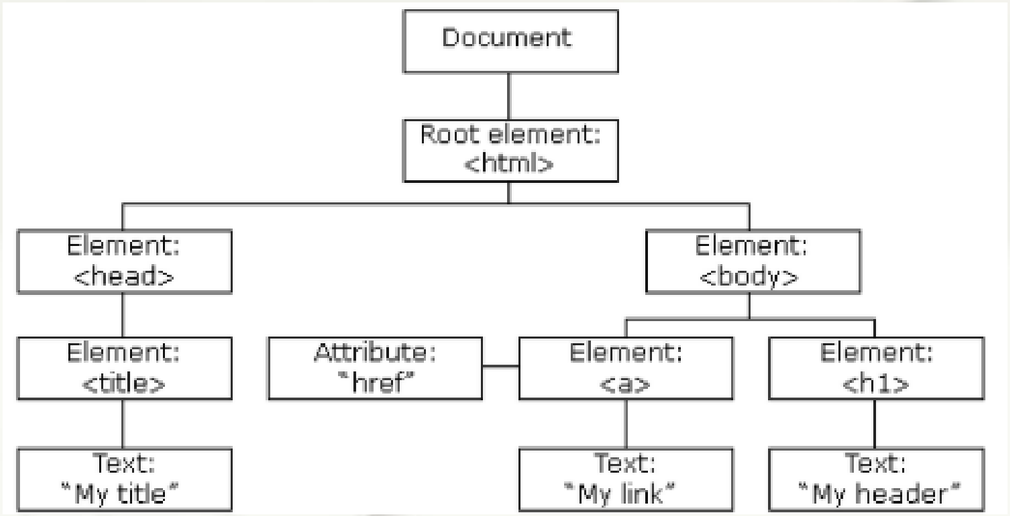


Figure 1 – HTML DOM Tree of Objects
(Source: https://www.w3schools.com/js/js_htmldom.asp)

Co-funded by the Erasmus+ Programme of the European Union

What can JavaScript do in the HTML DOM?

It can do the following:

- change all the HTML elements in the page
- change all the HTML attributes in the page
- change all the CSS styles in the page
- remove existing HTML elements and attributes
- add new HTML elements and attributes
- react to all existing HTML events in the page
- create new HTML events in the page

Co-funded by the
Erasmus+ Programme
of the European Union

Selecting DOM Elements in JavaScript

JavaScript is used to **get or modify the content or value of the HTML elements** of the web page and apply some special effects such as animations or hide.

To be able to perform any action, you need to find or select the target HTML element.

We will go through some of the most common ways of selecting elements on a page and manipulating them with JavaScript.

Co-funded by the
Erasmus+ Programme
of the European Union

Selecting the Topmost Elements

The topmost elements can be **accessed directly as document properties**.

For instance, to access the <html> element, use the document.documentElement property. For the <head> element, you can use the document.head property and for the <body> element, the document.body property.

Co-funded by the
Erasmus+ Programme
of the European Union

Selecting the Topmost Elements

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Topmost Elements</title>
</head>
<body>
    <script>
    // Display lang attribute value of html element
    alert(document.documentElement.getAttribute("lang")); // Outputs: en

    // Set background color of body element
    document.body.style.background = "yellow";

    // Display tag name of the head element's first child
    alert(document.head.firstElementChild.nodeName); // Outputs: meta
    </script>
</body>
</html>
```

Topmost Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

* It is important to note that the document.body **should not be used before** the <body> element since it will return null. The program needs to go through the <body> element first to access the document.body property.

Co-funded by the Erasmus+ Programme of the European Union

Selecting the Topmost Elements

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Document.body Demo</title>
    <script>
    alert("From HEAD: " + document.body); // Outputs: null (since <body> is not parsed yet)
    </script>
</head>
<body>
    <script>
    alert("From BODY: " + document.body); // Outputs: HTMLBodyElement
    </script>
</body>
</html>
```

Topmost Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

This example demonstrates what we saw in the beginning about the hierarchical relationships that exist between nodes. You need to be mindful that in order to access the document.body property, you will have to start from the <body> element to avoid null values.

Co-funded by the Erasmus+ Programme of the European Union

Selecting Elements by ID

If you want to find or select an HTML element, the easiest way is to select it is based on its unique ID. You can do this with the getElementById() method.

The getElementById() method is used to return the element as an object if a matching element is found. Otherwise, it will return null.

* Keep in mind that any HTML element can have an id attribute, which must be a unique value within a page. This essentially means that no two elements can have the same id.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Element by ID</title>
</head>
<body>
    <p id="mark">This is a paragraph of text.</p>
    <p>This is another paragraph of text.</p>

    <script>
    // Selecting element with id mark
    var match = document.getElementById("mark");

    // Highlighting element's background
    match.style.background = "yellow";
    </script>
</body>
</html>
```

Selecting Elements by ID Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

Co-funded by the Erasmus+ Programme of the European Union

Selecting Elements by Class Name

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Class Name</title>
</head>
<body>
    <p class="test">This is a paragraph of text.</p>
    <div class="block test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting elements with class test
    var matches = document.getElementsByClassName("test");

    // Displaying the selected elements count
    document.write("Number of selected elements: " + matches.length);

    // Applying bold style to first element in selection
    matches[0].style.fontWeight = "bold";

    // Applying italic style to last element in selection
    matches[matches.length - 1].style.fontStyle = "italic";

    // Highlighting each element's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

Selecting Elements by ID Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

If you want to select all the elements with specific class names, use the getElementsByClassName() method. It will return an array-like object of all child elements which have all the given class names.

Co-funded by the Erasmus+ Programme of the European Union

Selecting Elements by Tag Name

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Tag Name</title>
</head>
<body>
    <p>This is a paragraph of text.</p>
    <div class="test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting all paragraph elements
    var matches = document.getElementsByTagName("p");

    // Printing the number of selected paragraphs
    document.write("Number of selected elements: " + matches.length);

    // Highlighting each paragraph's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

Selecting Elements by Tag Name Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

If you want to select elements by their tag name, use the getElementsByTagName() method. This method will also return an array-like object of all child elements which have the given tag name.

Co-funded by the Erasmus+ Programme of the European Union

Selecting Elements with CSS Selectors

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements with CSS Selectors</title>
</head>
<body>
    <ul>
        <li>Bread</li>
        <li class="tick">Coffee</li>
        <li>Pineapple Cake</li>
    </ul>

    <script>
    // Selecting all li elements
    var matches = document.querySelectorAll("ul li");

    // Printing the number of selected li elements
    document.write("Number of selected elements: " + matches.length + "<hr>")

    // Printing the content of selected li elements
    for(var elem of matches) {
        document.write(elem.innerHTML + "<br>");
    }

    // Applying line through style to first li element with class tick
    matches = document.querySelectorAll("ul li.tick");
    matches[0].style.textDecoration = "line-through";
    </script>
</body>
</html>
```

Selecting Elements with CSS Selectors Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

CSS Selectors offer a very powerful and efficient way to select HTML elements in a document.

To select elements that match the specified CSS Selector, you can use the querySelectorAll() method.

This method will return a list of all the elements that match the specified selectors.

Co-funded by the Erasmus+ Programme of the European Union

Styling DOM Elements in JavaScript

You can also change the visual presentation of HTML documents in a dynamic way by using JavaScript to apply different styles to HTML elements. Almost all element styles can be set such as fonts, colours, margins, borders, background images, text alignment, width and height, position, and so on.

Here, we will go through various methods that can be used to set styles in JavaScript.

Co-funded by the
Erasmus+ Programme
of the European Union

Setting Inline Styles on Elements

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Set Inline Styles Demo</title>
</head>
<body>
    <p id="intro">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Appling styles on element
    elem.style.color = "blue";
    elem.style.fontSize = "18px";
    elem.style.fontWeight = "bold";
    </script>
</body>
</html>
```

Inline Styles on Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

The style attribute is used to apply inline styles directly to the specific HTML element. The style property is used in JavaScript to get or set the inline style of an element.

In the following example, colour and font properties will be set for an element with id="intro":

Co-funded by the Erasmus+ Programme of the European Union

Naming Conventions of CSS Properties in JavaScript

It is important to mention that many of CSS properties contain hyphens (-) in their names such as font-size, background-image, text-decoration, etc. However, in JavaScript, the hyphen is a reserved operator that signifies a minus sign. Therefore, it is not possible to write an expression in this way: elem.style.font-size.

To overcome this issue, CSS property names in JavaScript that contain one or more hyphens are converted to intercapatisalised style words. This essentially means that the hyphens are removed and the first letter after the hyphen is capitalised. For instance, the CSS property font-size become fontSize in DOM property.

Co-funded by the
Erasmus+ Programme
of the European Union

Getting Style Information from Elements

The style property is also used to get the styles applied to HTML elements.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Get Element's Style Demo</title>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting style information from element
    alert(elem.style.color);  // Outputs: red
    alert(elem.style.fontSize);  // Outputs: 20px
    alert(elem.style.fontStyle);  // Outputs nothing
    </script>
</body>
</html>
```

Style property - Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

The style property isn't the most useful when it comes to getting style information from the elements since it only returns the style rules that are set in the element's style attribute and not those that come from elsewhere such as style rules in the embedded style sheets, or external style sheets.

Co-funded by the Erasmus+ Programme of the European Union

Getting Style Information from Elements

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JS Get Computed Style Demo</title>
<style type="text/css">
    #intro {
        font-weight: bold;
        font-style: italic;
    }
</style>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting computed style information
    var styles = window.getComputedStyle(elem);

    alert(styles.getPropertyValue("color"));  // Outputs: rgb(255, 0, 0)
    alert(styles.getPropertyValue("font-size"));  // Outputs: 20px
    alert(styles.getPropertyValue("font-weight"));  // Outputs: 700
    alert(styles.getPropertyValue("font-style"));  // Outputs: italic
    </script>
</body>
</html>
```

window.getComputedStyle() - Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

If you want to get the values of all CSS properties that are used to render an element you can use the window.getComputedStyle() method, as shown in the following example:

* Keep in mind that the value 700 for the CSS property font-weight is the same as the keyword bold. The colour keyword red is the same as rgb(255,0,0), which is the rgb notation of a colour.

Co-funded by the Erasmus+ Programme of the European Union

Adding CSS Classes to Elements

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Add or Replace CSS Classes Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.className = "note";  // Add or replace all classes with note class
    elem.className += " highlight";  // Add a new class highlight
    </script>
</body>
</html>
```

Another way to get or set CSS classes to HTML elements is by using the className property. Class is a reserved word in JavaScript; thus, JavaScript uses the className property to refer to the value of the HTML class attribute.

Adding Classes to Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

Co-funded by the Erasmus+ Programme of the European Union

Adding CSS Classes to Elements

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS classList Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.classList.add("hide");  // Add a new class
    elem.classList.add("note", "highlight");  // Add multiple classes
    elem.classList.remove("hide"); // Remove a class
    elem.classList.remove("disabled", "note"); // Remove multiple classes
    elem.classList.toggle("visible"); // If class exists remove it, if not add it

    // Determine if class exist
    if(elem.classList.contains("highlight")) {
        alert("The specified class exists on the element.");
    }
    </script>
</body>
</html>
```

classList property - Adding Classes to Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

An even better way to work with CSS classes is by using the classList property to get, set or remove CSS classes easily from an element. This property is supported in all major browsers except Internet Explorer before version 10.

Co-funded by the Erasmus+ Programme of the European Union

Working with Attributes

Attributes are special words used inside the start tag of an HTML element to control the tag's behaviour or provide more information about the tag.

In this section, we will go through several methods of adding, removing or changing an HTML element's attribute.

Co-funded by the
Erasmus+ Programme
of the European Union

Getting Element's Attribute Value

To get the current value of an element’s attribute, you can use the getAttribute() method. If that particular attribute is not found on the element, it will return null.

```
<a href="https://www.google.com/" target="_blank" id="myLink">Google</a>

<script>
    // Selecting the element by ID attribute
    var link = document.getElementById("myLink");

    // Getting the attributes values
    var href = link.getAttribute("href");
    alert(href); // Outputs: https://www.google.com/

    var target = link.getAttribute("target");
    alert(target); // Outputs: _blank
</script>
```

getAttribute() method – Getting Element’s Attribute Value Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Co-funded by the Erasmus+ Programme of the European Union

Setting Attributes on Elements

If you want to set an attribute on a specified element, you can use the setAttribute() method. If the attribute already exists on the element, the value will be updated. If not, a new attribute will be added with specified name and value.

```
<button type="button" id="myBtn">Click Me</button>

<script>
    // Selecting the element
    var btn = document.getElementById("myBtn");

    // Setting new attributes
    btn.setAttribute("class", "click-btn");
    btn.setAttribute("disabled", "");
</script>
```

setAttribute() method – Setting Attributes on Elements Example
(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Setting Attributes on Elements

If you want to update or change the value of an existing attribute on an element, you can also use the setAttribute() method.

Let's see an example that will update the value of the existing href attribute of an anchor (<a>) element:

```
<a href="#" id="myLink">Tutorial Republic</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Changing the href attribute value
    link.setAttribute("href", "https://www.tutorialrepublic.com");
</script>
```

setAttribute() method – Setting Attributes on Elements Example
(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Removing Attributes from Elements

To remove an attribute from a specific element, you can use the removeAttribute() method.

Remember the href attribute that we changed from the anchor element; we are now going to remove it in the following example:

```
<a href="https://www.google.com/" id="myLink">Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Removing the href attribute
    link.removeAttribute("href");
</script>
```

removeAttribute() method – Removing Attributes from Elements Example
(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Manipulating DOM Elements in JavaScript

So far, we have learnt how to select and style HTML DOM elements.

Now, we will learn how to add or remove DOM elements in a dynamic way, how to get their contents and many more.

Co-funded by the
Erasmus+ Programme
of the European Union

Adding New Elements to DOM

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.appendChild(newDiv, currentDiv);
</script>
```

The document.createElement() method is used to create a new element in an HTML document. It creates a new element; however, it does not add it to the DOM.

A separate step is needed to add it to the DOM. In the example we just saw, the appendChild() is used to add the new element at the end of any other children under the specified parent node.

Adding New Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

Co-funded by the Erasmus+ Programme of the European Union

Adding New Elements to DOM

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.insertBefore(newDiv, currentDiv);
</script>
```

Adding New Elements Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

You also have the option to **add the new element before any other children**, as shown in the example here.

Co-funded by the Erasmus+ Programme of the European Union

Getting or Setting HTML Contents to DOM

If you want to get or set the contents of HTML elements, you can use the innerHTML property. This property is used to set or get the HTML markup inside the element, which contains content between its opening and closing tags.

As you can from the example, new elements are inserted quite easily into the DOM with the innerHTML property. But this property replaces all the existing content of an element.

```
1  <div id="main">
2      <h1 id="title">Hello World!</h1>
3      <p id="hint">This is a simple paragraph.</p>
4  </div>
5
6  <script>
7  // Getting inner HTML conents
8  var contents = document.getElementById("main").innerHTML;
9  alert(contents); // Outputs inner html contents
10
11 // Setting inner HTML contents
12 var mainDiv = document.getElementById("main");
13 mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
14 </script>
```

Getting or Setting HTML Contents to DOM Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Getting or Setting HTML Contents to DOM

```
<!-- beforebegin -->
<div id="main">
    <!-- afterbegin -->
    <h1 id="title">Hello World!</h1>
    <!-- beforeend -->
</div>
<!-- afterend -->

<script>
// Selecting target element
var mainDiv = document.getElementById("main");

// Inserting HTML just before the element itself, as a previous sibling
mainDiv.insertAdjacentHTML('beforebegin', '<p>This is paragraph one.</p>');

// Inserting HTML just inside the element, before its first child
mainDiv.insertAdjacentHTML('afterbegin', '<p>This is paragraph two.</p>');

// Inserting HTML just inside the element, after its last child
mainDiv.insertAdjacentHTML('beforeend', '<p>This is paragraph three.</p>');

// Inserting HTML just after the element itself, as a next sibling
mainDiv.insertAdjacentHTML('afterend', '<p>This is paragraph four.</p>');
</script>
```

Getting or Setting HTML Contents to DOM Example
(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

If you do not want to replace the existing contents of an element, you can use the insertAdjacentHTML() method.

This method takes two parameters: the HTML to be inserted and its position.

The position must be one of the following: "beforebegin", "afterbegin", "beforeend", and "afterend". It is also significant to note that this method is supported in all major browsers.

Co-funded by the Erasmus+ Programme of the European Union

Removing Existing Elements from DOM

To remove a child node from the DOM, you can use the removeChild() method. This method will also return the removed node.

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var childElem = document.getElementById("hint");
parentElem.removeChild(childElem);
</script>
```

Removing Existing Elements from DOM Example
(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

Co-funded by the Erasmus+ Programme of the European Union

Removing Existing Elements from DOM

You can also remove the child element without knowing the parent element. You can find the child element and use the parentNode property to find its parent. It will return the parent of the given node in the DOM tree.

```
1  <div id="main">
2      <h1 id="title">Hello World!</h1>
3      <p id="hint">This is a simple paragraph.</p>
4  </div>
5
6  <script>
7  var childElem = document.getElementById("hint");
8  childElem.parentNode.removeChild(childElem);
9  </script>
```

Removing Existing Elements from DOM Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

Co-funded by the Erasmus+ Programme of the European Union

Replacing Existing Elements in DOM

```html
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var oldPara = document.getElementById("hint");

// Creating new elememt
var newPara = document.createElement("p");
var newContent = document.createTextNode("This is a new paragraph.");
newPara.appendChild(newContent);

// Replacing old paragraph with newly created paragraph
parentElem.replaceChild(newPara, oldPara);
</script>
```

Replacing Existing Elements in DOM Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

You also have the option of replacing an element in HTML DOM with another by using the replaceChild() method.

This method takes on two parameters: the node to be inserted and the node to be replaced.

The syntax is used is as follows:

parentNode.replaceChild(newChild, oldChild);

Co-funded by the Erasmus+ Programme of the European Union

Navigating Between DOM Nodes

By now, you should have a better idea of how to select individual elements on a web page. There are many occasions where you would need to access child, parent or ancestor element. We have talked about nodes in the beginning of this subchapter and now we will see how we can access the different types of nodes.

DOM nodes have several properties and methods that let you navigate or traverse through the tree DOM structure and make necessary changes quite easily.

Co-funded by the
Erasmus+ Programme
of the European Union

Accessing the Child Nodes

The firstChild and lastChild properties allow you to access the first and last direct child node of a node respectively. If a node does not have any child element, it will return null.

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");
console.log(main.firstChild.nodeName); // Prints: #text

var hint = document.getElementById("hint");
console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

Accessing Child Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

* Please note that the nodeName is a read-only property, which returns the name of the current node as a string. For example, it will return the tag name of an element node, #text for text node, #comment for comment node, #document for document node, and so on.

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Child Nodes

```
1  <div id="main">
2      <h1 id="title">My Heading</h1>
3      <p id="hint"><span>This is some text.</span></p>
4  </div>
5
6  <script>
7  var main = document.getElementById("main");
8  console.log(main.firstChild.nodeName); // Prints: #text
9
10 var hint = document.getElementById("hint");
11 console.log(hint.firstChild.nodeName); // Prints: SPAN
12 </script>
```

Accessing Child Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

In the example that we just saw, the nodeName of the first child node of the main DIV elements returned #text instead of H1.

This happens because **white space**, i.e., spaces, tabs, newlines, and so on, **are considered valid characters and they become part of the DOM tree in the form of #text nodes.**

Then, the tag that contains a newline before the <h1> will create #text node.

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Child Nodes

```
1  <div id="main">
2      <h1 id="title">My Heading</h1>
3      <p id="hint"><span>This is some text.</span></p>
4  </div>
5
6  <script>
7  var main = document.getElementById("main");
8  alert(main.firstElementChild.nodeName); // Outputs: H1
9  main.firstElementChild.style.color = "red";
10
11 var hint = document.getElementById("hint");
12 alert(hint.firstElementChild.nodeName); // Outputs: SPAN
13 hint.firstElementChild.style.color = "blue";
14 </script>
```

Accessing Child Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

In order to prevent this issue with the firstChild and lastChild returning #text or #comment nodes, you can use the firstElementChild and lastElementChild properties as an alternative.

These properties will return only the first and last element of the node respectively. However, this will not work in Internet Explores prior to Version 9.

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Child Nodes

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.childNodes;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

Accessing Child Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

To access all child nodes of a given element, you can also use the childNodes property.

Keep in mind that the first child node is assigned index 0.

Here, the childNodes returns all child nodes, including non-element nodes like text and comment nodes.

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Child Nodes

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.children;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

If you want to get a collection of only elements, you should use the children property instead.

Accessing Child Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Parent Nodes

To access the parent node of a specific node in the DOM tree, you can use the parentNode property.

* Note that the parentNode property will always return null values for document nodes because they do not have parents.

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
alert(document.documentElement.parentNode.nodeName); // Outputs: #document
alert(document.parentNode); // Outputs: null
</script>
```

Accessing Parent Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

It is good to know that the **topmost DOM tree nodes can be accessed directly as document properties,** such as the <html> element, which can be accessed with document.documentElement property.

Co-funded by the Erasmus+ Programme of the European Union

Accessing the Parent Nodes

There is also an option to **get only element nodes** with the parentElement, as shown in the example below:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
hint.parentNode.style.backgroundColor = "yellow";
</script>
```

Accessing Parent Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Accessing the Sibling Nodes

To **access the previous and next node** in the DOM tree, you can use the previousSibling and nextSibling properties respectively.

```
1  <div id="main">
2      <h1 id="title">My Heading</h1>
3      <p id="hint"><span>This is some text.</span></p><hr>
4  </div>
5
6  <script>
7  var title = document.getElementById("title");
8  alert(title.previousSibling.nodeName); // Outputs: #text
9
10 var hint = document.getElementById("hint");
11 alert(hint.nextSibling.nodeName); // Outputs: HR
12 </script>
```

Accessing Sibling Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

Co-funded by the
Erasmus+ Programme
of the European Union

Accessing the Sibling Nodes

To skip any whitespace text nodes, you can use the previousElementSibling and nextElementSibling as alternatives to **get the previous and next sibling elements**. If no such sibling is found, these properties will return null values.

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.previousElementSibling.nodeName); // Outputs: H1
alert(hint.previousElementSibling.textContent); // Outputs: My Heading

var title = document.getElementById("title");
alert(title.nextElementSibling.nodeName); // Outputs: P
alert(title.nextElementSibling.textContent); // Outputs: This is some text.
</script>
```

Accessing Sibling Nodes Example

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

The textContent property used here signifies the **text content of a node and all of its descendants.**

Co-funded by the Erasmus+ Programme of the European Union

Types of DOM Nodes

The DOM tree is comprised of different types of nodes that includes elements, text, comments and many more.

Every node has a nodeType property that can help you understand how you can access and manipulate said node.

Constant	Value	Description
`ELEMENT_NODE`	1	An element node such as `<p>` or `<img>`.
`TEXT_NODE`	3	The actual text of element.
`COMMENT_NODE`	8	A comment node i.e. `<!-- some comment -->`
`DOCUMENT_NODE`	9	A document node i.e. the parent of `<html>` element.
`DOCUMENT_TYPE_NODE`	10	A document type node e.g. `<!DOCTYPE html>` for HTML5 documents.

Table of Most Common Types of DOM Nodes

(Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

Co-funded by the Erasmus+ Programme of the European Union

Let’s practice

You have learnt a lot of new things by now, so it is time to put what we have learnt into practice!

To do this, follow either link:

- https://www.w3resource.com/javascript-exercises/javascript-dom-exercises.php
- https://www.tutorialrepublic.com/javascript-examples.php

Co-funded by the
Erasmus+ Programme
of the European Union

THANK YOU!

NEXT CHAPTER: JavaScript & BOM

Co-funded by the
Erasmus+ Programme
of the European Union