**Document Object Model**

The Document Object Model, or DOM for short, is a platform and language independent model to represent the HTML or XML documents. It defines the logical structure of the documents and the way in which they can be accessed and manipulated by an application program.

In the DOM, all parts of the document, such as elements, attributes, text, etc. are organized in a hierarchical tree-like structure; similar to a family tree in real life that consists of parents and children. In DOM terminology these individual parts of the document are known as nodes.

The Document Object Model that represents HTML document is referred to as HTML DOM. Similarly, the DOM that represents the XML document is referred to as XML DOM.

<!DOCTYPE html>

<html>

<head>

    <title>My Page</title>

</head>

<body>

    <h1>Mobile OS</h1>

    <ul>

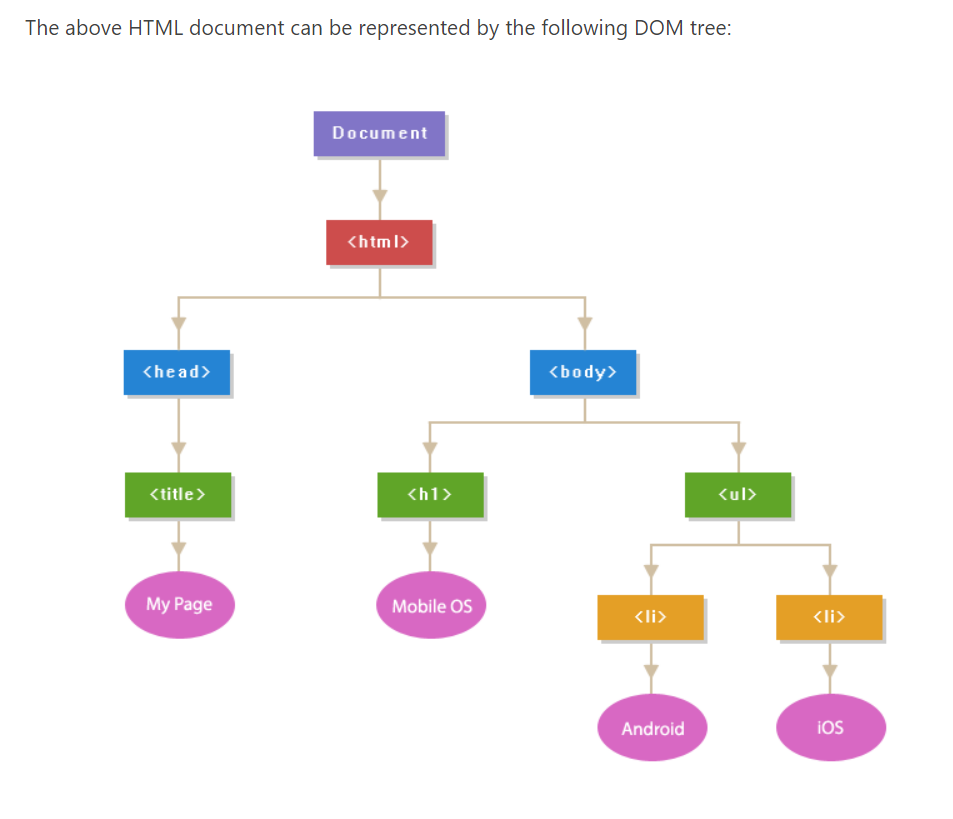
        <li>Android</li>

        <li>iOS</li>

    </ul>

</body>

</html>



The above diagram demonstrates the parent/child relationships between the nodes. The topmost node i.e. the Document node is the root node of the DOM tree, which has one child, the <html> element. Whereas, the <head> and <body> elements are the child nodes of the <html> parent node.

The <head> and <body> elements are also siblings since they are at the same level. Further, the text content inside an element is a child node of the parent element. So, for example, "Mobile OS" is considered as a child node of the <h1> that contains it, and so on.

[Comments](https://www.tutorialrepublic.com/javascript-tutorial/javascript-syntax.php#comments) inside the HTML document are nodes in the DOM tree as well, even though it doesn't affect the visual representation of the document in any way. Comments are useful for documenting the code, however, you will rarely need to retrieve and manipulate them.

HTML attributes such as id, class, title, style, etc. are also considered as nodes in DOM hierarchy but they don't participate in parent/child relationships like the other nodes do. They are accessed as properties of the element node that contains them.

Each element in an HTML document such as image, hyperlink, form, button, heading, paragraph, etc. is represented using a JavaScript object in the DOM hierarchy, and each object contains properties and methods to describe and manipulate these objects.

# Selecting the Topmost Elements

The topmost elements in an HTML document are available directly as document properties. For example, the [<html>](https://www.tutorialrepublic.com/html-reference/html-html-tag.php) element can be accessed with document.documentElement property, whereas the [<head>](https://www.tutorialrepublic.com/html-reference/html-head-tag.php) element can be accessed with document.head property, and the [<body>](https://www.tutorialrepublic.com/html-reference/html-body-tag.php) element can be accessed with document.body property. Here's an example:

<!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>

But, be careful. If document.body is used before the <body> tag (e.g. inside the <head>), it will return [null](https://www.tutorialrepublic.com/javascript-tutorial/javascript-data-types.php#null) instead of the body element. Because the point at which the script is executed, the <body> tag was not parsed by the browser, so document.body is truly null at that point.

Let's take a look at the following example to better understand this:

<!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>

## Selecting Elements by ID

You can select an element based on its unique ID with the getElementById() method. This is the easiest way to find an HTML element in the DOM tree.

The following example selects and highlight an element having the ID attribute id="mark".

<!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");

     alert('document.getElementById("mark")'+match);

    // Highlighting element's background

    match.style.background = "yellow";

    </script>

</body>

</html>

The getElementById() method will return the element as an object if the matching element was found, or null if no matching element was found in the document.

**Note:** Any HTML element can have an id attribute. The value of this attribute must be unique within a page i.e. no two elements in the same page can have the same ID.

## Selecting Elements by Class Name

Similarly, you can use the getElementsByClassName() method to select all the elements having specific class names. This method returns an array-like object of all child elements which have all of the given class names. Let's check out the following example:

<!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 Tag Name

You can also select HTML elements by tag name using the getElementsByTagName() method. This method also returns an array-like object of all child elements with the given tag name.

<!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 with CSS Selectors

You can use the querySelectorAll() method to select elements that matches the specified [CSS selector](https://www.tutorialrepublic.com/css-tutorial/css-selectors.php). CSS selectors provide a very powerful and efficient way of selecting HTML elements in a document. Please check out the CSS tutorial section to learn more about them.

This method returns a list of all the elements that matches the specified selectors. You can examine it just like any array, as shown in the following example:

<!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>

**Note:** The querySelectorAll() method also supports [CSS pseudo-classes](https://www.tutorialrepublic.com/css-tutorial/css-pseudo-classes.php) like :first-child, :last-child, :hover, etc. But, for [CSS pseudo-elements](https://www.tutorialrepublic.com/css-tutorial/css-pseudo-elements.php) such as ::before, ::after, ::first-line, etc. this method always returns an empty list.

## Styling DOM Elements in JavaScript

You can also apply style on HTML elements to change the visual presentation of HTML documents dynamically using JavaScript. You can set almost all the styles for the elements like, fonts, colors, margins, borders, background images, text alignment, width and height, position, and so on.

In the following section we'll discuss the various methods of setting styles in JavaScript.

## Setting Inline Styles on Elements

Inline styles are applied directly to the specific HTML element using the style attribute. In JavaScript the style property is used to get or set the inline style of an element.

The following example will set the color and font properties of an element with id="intro".

<!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>

### Naming Conventions of CSS Properties in JavaScript

Many CSS properties, such as [font-size](https://www.tutorialrepublic.com/css-reference/css-font-size-property.php), [background-image](https://www.tutorialrepublic.com/css-reference/css-background-image-property.php), [text-decoration](https://www.tutorialrepublic.com/css-reference/css-text-decoration-property.php), etc. contain hyphens (-) in their names. Since, in JavaScript hyphen is a reserved operator and it is interpreted as a minus sign, so it is not possible to write an expression, like: elem.style.font-size

Therefore, in JavaScript, the CSS property names that contain one or more hyphens are converted to intercapitalized style word. It is done by removing the hyphens and capitalizing the letter immediately following each hyphen, thus the CSS property font-size becomes the DOM property fontSize, border-left-style becomes borderLeftStyle, and so on.

<!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>

The style property isn't very useful when it comes to getting style information from the elements, because it only returns the style rules set in the element's style attribute not those that come from elsewhere, such as style rules in the [embedded style sheets](https://www.tutorialrepublic.com/html-tutorial/html-styles.php#embedded-style-sheet), or [external style sheets](https://www.tutorialrepublic.com/html-tutorial/html-styles.php#external-style-sheet).

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

<!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>

**Tip:** The value 700 for the CSS property [font-weight](https://www.tutorialrepublic.com/css-reference/css-font-weight-property.php) is same as the keyword bold. The color keyword red is same as rgb(255,0,0), which is the [rgb notation](https://www.tutorialrepublic.com/css-reference/css-color-values.php) of a color.

## Adding CSS Classes to Elements

You can also get or set [CSS classes](https://www.tutorialrepublic.com/css-tutorial/css-selectors.php) to the HTML elements using the className property.

Since, class is a [reserved word in JavaScript](https://www.tutorialrepublic.com/javascript-reference/javascript-reserved-keywords.php), so JavaScript uses the className property to refer the value of the HTML class attribute. The following example will show to how to add a new class, or replace all existing classes to a [<div>](https://www.tutorialrepublic.com/html-reference/html-div-tag.php) element having id="info".

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Add or Replace CSS Classes on an Element</title>

<style>

    .highlight {

        background: yellow;

    }

    .disabled{

        background: lightgray;

        opacity: 0.7;

    }

</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>

## Getting Element's Attribute Value

The getAttribute() method is used to get the current value of a attribute on the element.

If the specified attribute does not exist on the element, it will return null. Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Get the Value of an Attribute</title>

</head>

<body>

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

    <script>

        // Selecting the element by ID attribute

        var link = document.getElementById("myLink");

        // Getting the attributes values

        var href = link.getAttribute("href");

        document.write(href); // Prints: https://www.google.com/

        document.write("<br>");

        var target = link.getAttribute("target");

        document.write(target); // Prints: \_blank

    </script>

</body>

</html>

## Setting Attributes on Elements

The setAttribute() method is used to set an attribute on the specified element.

If the attribute already exists on the element, the value is updated; otherwise a new attribute is added with the specified name and value. The JavaScript code in the following example will add a class and a disabled attribute to the [<button>](https://www.tutorialrepublic.com/html-reference/html-button-tag.php) element.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Set an Attribute on an Element</title>

</head>

<body>

    <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>

</body>

</html>

## Manipulating DOM Elements in JavaScript

Now that you've learnt how to select and style HTML DOM elements. In this chapter we will learn how to add or remove DOM elements dynamically, get their contents, and so on.

## Adding New Elements to DOM

You can explicitly create new element in an HTML document, using the document.createElement() method. This method creates a new element, but it doesn't add it to the DOM; you'll have to do that in a separate step, as shown in the following example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Insert New Element into the DOM</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

  <button type="button" onclick="insertElement()">Insert Element</button>

  <script>

  function insertElement() {

    // 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>

</body>

</html>

The appendChild() method adds the new element at the end of any other children of a specified parent node. However, if you want to add the new element at the beginning of any other children you can use the insertBefore() method, as shown in example below:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Insert New Element at the Beginning</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

  <button type="button" onclick="insertElement()">Insert Element</button>

  <script>

  function insertElement() {

    // 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>

</body>

</html>

## Getting or Setting HTML Contents to DOM

You can also get or set the contents of the HTML elements easily with the innerHTML property. This property sets or gets the HTML markup contained within the element i.e. content between its opening and closing tags. Checkout the following example to see how it works:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Get Set Inner HTML of an Element</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

  <button type="button" onclick="getContents()">Get Contents</button>

  <button type="button" onclick="setContents()">Set Contents</button>

  <script>

  function getContents() {

    // Getting inner HTML conents

    var contents = document.getElementById("main").innerHTML;

    alert(contents); // Outputs inner html contents

  }

  function setContents() {

    // Setting inner HTML contents

    var mainDiv = document.getElementById("main");

    mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";

  }

  </script>

</body>

</html>

As you can see how easily you can insert new elements into DOM using the innerHTML property, but there is one problem, the innerHTML property replaces all existing content of an element. So if you want to insert the HTML into the document without replacing the existing contents of an element, you can use the insertAdjacentHTML() method.

This method accepts two parameters: the position in which to insert and the HTML text to insert. The position must be one of the following values: "beforebegin", "afterbegin", "beforeend", and "afterend". This method is supported in all major browsers.

The following example shows the visualization of position names and how it works.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Insert HTML without Replacing the Existing Content</title>

</head>

<body>

    <!-- beforebegin -->

  <div id="main">

    <!-- afterbegin -->

    <h1 id="title">Hello World!</h1>

    <!-- beforeend -->

  </div>

  <!-- afterend -->

  <button type="button" onclick="insertContent()">Insert Content</button>

  <script>

  function insertContent() {

    // 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>

</body>

</html>

**Note:** The beforebegin and afterend positions work only if the node is in the DOM tree and has a parent element. Also, when inserting HTML into a page, be careful not to use user input that hasn't been escaped, to prevent XSS attacks.

## Removing Existing Elements from DOM

Similarly, you can use the removeChild() method to remove a child node from the DOM. This method also returns the removed node. Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Remove an Element from the DOM</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

  <button type="button" onclick="removeElement()">Remove Element</button>

  <script>

  function removeElement() {

    var parentElem = document.getElementById("main");

    var childElem = document.getElementById("hint");

    parentElem.removeChild(childElem);

  }

  </script>

</body>

It is also possible to remove the child element without exactly knowing the parent element. Simply find the child element and use the parentNode property to find its parent element. This property returns the parent of the specified node in the DOM tree. Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Remove Child Element without Knowing its Parent</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

  <button type="button" onclick="removeElement()">Remove Element</button>

  <script>

  function removeElement() {

    var childElem = document.getElementById("hint");

    childElem.parentNode.removeChild(childElem);

  }

  </script>

</body>

</html>

## Replacing Existing Elements in DOM

You can also replace an element in HTML DOM with another using the replaceChild() method. This method accepts two parameters: the node to insert and the node to be replaced. It has the syntax like parentNode.replaceChild(newChild, oldChild);. Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Replace an Element with another Element in the DOM</title>

</head>

<body>

    <div id="main">

    <h1 id="title">Hello World!</h1>

    <p id="hint">This is a simple paragraph.</p>

  </div>

    <button type="button" onclick="replaceParagraph()">Replace Paragraph</button>

  <script>

    function replaceParagraph() {

        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>

</body>

</html>

## Navigating Between DOM Nodes

In the previous chapters you've learnt how to select individual elements on a web page. But there are many occasions where you need to access a child, parent or ancestor element. See the [JavaScript DOM nodes](https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-nodes.php) chapter to understand the logical relationships between the nodes in a DOM tree.

DOM node provides several properties and methods that allow you to navigate or traverse through the tree structure of the DOM and make changes very easily. In the following section we will learn how to navigate up, down, and sideways in the DOM tree using JavaScript.

## Accessing the Child Nodes

You can use the firstChild and lastChild properties of the DOM node to access the first and last direct child node of a node, respectively. If the node doesn't have any child element, it returns null.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Get First and Last Child Nodes</title>

</head>

<body>

    <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>

  <p><strong>Note:</strong> Please check out the browser console by pressing the f12 key on the keyboard.</p>

</body>

</html>

**Note:** The nodeName is a read-only property that returns the name of the current node as a string. For example, it returns the tag name for element node, #text for text node, #comment for comment node, #document for document node, and so on.

If you notice the above example, the nodeName of the first-child node of the main DIV element returns #text instead of H1. Because, whitespace such as spaces, tabs, newlines, etc. are valid characters and they form #text nodes and become a part of the DOM tree. Therefore, since the <div> tag contains a newline before the <h1> tag, so it will create a #text node.

To avoid the issue with firstChild and lastChild returning #text or #comment nodes, you could alternatively use the firstElementChild and lastElementChild properties to return only the first and last *element node*, respectively. But, it will not work in IE 9 and earlier.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Get Only Element Child Nodes</title>

</head>

<body>

    <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.firstElementChild.nodeName); // Prints: H1

  main.firstElementChild.style.color = "red";

  var hint = document.getElementById("hint");

  console.log(hint.firstElementChild.nodeName); // Prints: SPAN

  hint.firstElementChild.style.color = "blue";

  </script>

  <p><strong>Note:</strong> Please check out the browser console by pressing the f12 key on the keyboard.</p>

</body>

</html>

Similarly, you can use the childNodes property to access all child nodes of a given element, where the first child node is assigned index 0. Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>JavaScript Get All Child Nodes</title>

</head>

<body>

    <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++) {

      console.log(nodes[i].nodeName);

    }

  }

  </script>

  <p><strong>Note:</strong> Please check out the browser console by pressing the f12 key on the keyboard.</p>

</body>

</html>

The childNodes returns all child nodes, including non-element nodes like text and comment nodes. To get a collection of only elements, use children property instead.

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