Document Object Model (DOM)

The **Document Object Model (DOM)** is a structured representation of an HTML document, forming a tree-like hierarchy where each HTML element, attribute, and piece of text is a node.

What is a DOM Tree?

The **Document Object Model (DOM) Tree** is a hierarchical representation of an HTML document, where each **HTML element, attribute, and text** is considered a **node**. The structure follows a **parent-child** relationship, similar to a tree diagram.

---

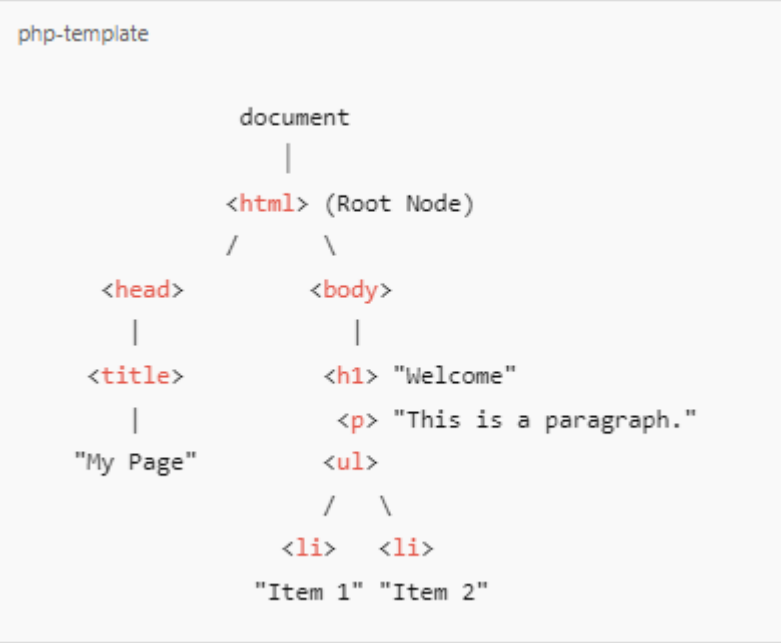
1. Structure of the DOM Tree

When the browser loads an HTML document, it creates a structured tree-like representation. Each **HTML tag** becomes a **node** in the tree, and nested elements become **child nodes** of their parent element.

Example HTML Document

```html
<!DOCTYPE html>
<html>
<head>
    <title>My Page</title>
</head>
<body>
    <h1>Welcome</h1>
    <p>This is a paragraph.</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
</body>
</html>
```

Corresponding DOM Tree Representation



- **Root Node**: `<html>` is the root of the tree.
- **Parent-Child Relationship**: `<html>` is the parent of `<head>` and `<body>`.
- **Sibling Relationship**: `<h1>`, `<p>`, and `<ul>` are siblings under `<body>`.
- **Leaf Nodes**: Text nodes such as `"Welcome"` and `"Item 1"` are leaf nodes, meaning they do not have children.

---

2. Types of Nodes in the DOM Tree

The DOM tree consists of **different types of nodes**:

(i) Document Node

- The root of the DOM tree is always the **document** node.
- It represents the entire webpage.

(ii) Element Nodes

- Every HTML tag (e.g., `<html>`, `<body>`, `<p>`, `<li>`) is an **element node**.
- Element nodes can contain **child elements, attributes, or text**.

(iii) Attribute Nodes

- Represent attributes of elements (e.g., `class`, `id`, `href`).
- Attributes belong to elements but are **not part of the DOM tree** directly.

(iv) Text Nodes

- Represent the text inside elements.
- Text nodes are always children of an element node.

Example:

```html
<p class="intro">Hello World</p>
```

- `<p>` → Element node
- `class="intro"` → Attribute node
- `"Hello World"` → Text node

Why is the DOM Tree Important?

✅**Dynamically modify content** (update text, styles, attributes).
✅**Navigate between elements** to find or modify specific parts of a webpage.
✅**Handle user events** like clicks, form submissions, and key presses.

1. Understanding the DOM

- The **DOM** allows JavaScript to dynamically access and modify the content, structure, and styles of a webpage.
- The document structure follows a **parent-child relationship**, where elements are nested inside other elements.
- It is created by the **browser** when the page is loaded and can be manipulated using JavaScript.

2. Selecting Elements in the DOM

To work with elements in the DOM, we need to select them first. JavaScript provides multiple methods for selecting elements:

2.1 Selecting by ID Attributes

- The `getElementById()` method selects an element by its unique ID.
- Example:

HTML c05/get-element-by-id.html

```
<h1 id="header">List King</h1>
<h2>Buy groceries</h2>
<ul>
  <li id="one" class="hot"><em>fresh</em>
    figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```

JAVASCRIPT c05/js/get-element-by-id.js

```
// Select the element and store it in a variable.
var el = document.getElementById('one');

// Change the value of the class attribute.
el.className = 'cool';
```

RESULT

-

2.2 Selecting by Class Attributes

- The `getElementsByClassName()` method selects all elements with a specific class.
- Example:

c05/js/get-elements-by-class-name.js JAVASCRIPT

```
var elements = document.getElementsByClassName('hot'); // Find hot items

if (elements.length > 2) {                          // If 3 or more are found

    var el = elements[2];           // Select the third one from the NodeList
    el.className = 'cool';          // Change the value of its class attribute

}
```

2.3 Selecting by Tag Name

- The `getElementsByTagName()` method selects all elements with a specific tag name.
- Example:

JAVASCRIPT — c05/js/get-elements-by-tag-name.js

```
var elements = document.getElementsByTagName('li');   // Find <li> elements

if (elements.length > 0) {                             // If 1 or more are found

    var el = elements[0];              // Select the first one using array syntax
    el.className = 'cool';             // Change the value of the class attribute

}
```

2.4 Selecting Using Query Selectors

- `querySelector()`: Selects the **first** element that matches a given CSS selector.
- `querySelectorAll()`: Selects **all** elements that match a given CSS selector.

Examples:

c05/js/query-selector.js JAVASCRIPT

```
// querySelector() only returns the first match
var el = document.querySelector('li.hot');
el.className = 'cool';

// querySelectorAll returns a NodeList
// The second matching element (the third list item) is selected and changed
var els = document.querySelectorAll('li.hot');
els[1].className = 'cool';
```

Working with DOM Nodes

Once elements are selected, we can navigate and modify them.

3.1 Navigating the DOM

- `parentNode` → Selects the parent element.
- `firstChild` → Selects the first child node.
- `lastChild` → Selects the last child node.
- `nextSibling` → Selects the next sibling node.
- `previousSibling` → Selects the previous sibling node.

Example:

PREVIOUS & NEXT SIBLING

c05/sibling.html

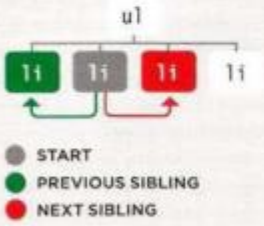
HTML

```
<ul><li id="one" class="hot"><em>fresh</em> figs</li><li id="two"
class="hot">pine nuts</li><li id="three" class="hot">honey</li><li
id="four">balsamic vinegar</li></ul>
```

c05/js/sibling.js

JAVASCRIPT

```
// Select the starting point and find its siblings
var startItem = document.getElementById('two');
var prevItem = startItem.previousSibling;
var nextItem = startItem.nextSibling;

// Change the values of the siblings' class attributes
prevItem.className = 'complete';
nextItem.className = 'cool';
```



START
PREVIOUS SIBLING
NEXT SIBLING

Note how references to sibling nodes are stored in new variables. This means properties such as `className` can be used on that node by adding the dot notation between the variable name and the property.

RESULT

FIRST & LAST CHILD

HTML

c05/child.html

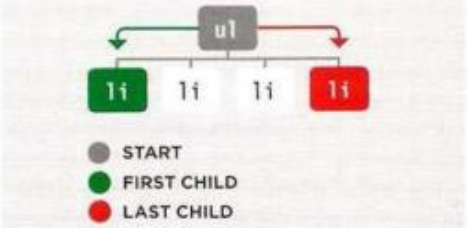
```
<ul
  ><li id="one" class="hot"><em>fresh</em> figs</li
  ><li id="two" class="hot">pine nuts</li
  ><li id="three" class="hot">honey</li
  ><li id="four">balsamic vinegar</li
></ul>
```

JAVASCRIPT

c05/js/child.js

```
// Select the starting point and find its children
var startItem = document.getElementsByTagName('ul')[0];
var firstItem = startItem.firstChild;
var lastItem = startItem.lastChild;

// Change the values of the children's class attributes
firstItem.setAttribute('class', 'complete');
lastItem.setAttribute('class', 'cool');
```

RESULT



Updating Element Content & Attributes

- `innerHTML` → Changes the entire inner HTML of an element.
- `textContent` → Updates only the text content.

ACCESSING & CHANGING A TEXT NODE

JAVASCRIPT

```
var itemTwo = document.getElementById('two');          /
var elText  = itemTwo.firstChild.nodeValue;            /
elText = elText.replace('pine nuts', 'kale');          /
itemTwo.firstChild.nodeValue = elText;                 /
```

RESULT

UPDATE TEXT & MARKUP

JAVASCRIPT c05/js/inner-html.js

```
// Store the first list item in a variable
var firstItem = document.getElementById('one');

// Get the content of the first list item
var itemContent = firstItem.innerHTML;

// Update the content of the first list item so it is a link
firstItem.innerHTML = '<a href=\"http://example.org\">' + itemContent + '</a>';
```

RESULT

ADDING AN ELEMENT TO THE DOM TREE

`createElement()` creates an element that can be added to the DOM tree, in this case an empty `<li>` element for the list.

This new element is stored inside a variable called `newEl` until it is attached to the DOM tree later on.

`createTextNode()` allows you to create a new text node to attach to an element. It is stored in a variable called `newText`.

JAVASCRIPT c05/js/add-element.js

```
// Create a new element and store it in a variable.
var newEl = document.createElement('li');

// Create a text node and store it in a variable.
var newText = document.createTextNode('quinoa');

// Attach the new text node to the new element.
newEl.appendChild(newText);

// Find the position where the new element should be added.
var position = document.getElementsByTagName('ul')[0];

// Insert the new element into its position.
position.appendChild(newEl);
```

RESULT

REMOVING AN ELEMENT FROM THE DOM TREE

This example uses the `removeChild()` method to remove the fourth item from the list (along with its contents).

The first variable, `removeEl`, stores the actual element you want to remove from the page (the fourth list item).

The second variable, `containerEl`, stores the `<ul>` element that *contains* the element you want to remove.

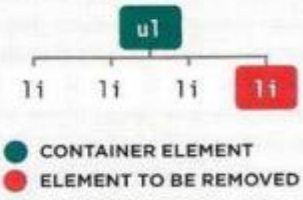
JAVASCRIPT c05/js/remove-element.js

```
var removeEl = document.getElementsByTagName('li')[3]; // The element to remove

var containerEl = removeEl.parentNode;                 // Its containing element

containerEl.removeChild(removeEl);                     // Removing the element
```

RESULT

The `removeChild()` method is used on the variable that holds the container node.

It requires one parameter: the element you want to remove (which is stored in the second variable).



CONTAINER ELEMENT
ELEMENT TO BE REMOVED

CHECK FOR AN ATTRIBUTE AND GET ITS VALUES

Before you work with an attribute, it is good practice to check whether it exists. This will save resources if the attribute cannot be found.

The `hasAttribute()` method of any element node lets you check if an attribute exists. The attribute name is given as an argument in the parentheses.

Using `hasAttribute()` in an `if` statement like this means that the code inside the curly braces will run only if the attribute exists on the given element.

JAVASCRIPT

c05/js/get-attribute.js

```
var firstItem = document.getElementById('one');     // Get first list item

if (firstItem.hasAttribute('class')) {              // If it has class attribute
    var attr = firstItem.getAttribute('class');     // Get the attribute

    // Add the value of the attribute after the list
    var el = document.getElementById('scriptResults');
    el.innerHTML = '<p>The first item has a class name: ' + attr + '</p>';

}
```

RESULT

CREATING ATTRIBUTES & CHANGING THEIR VALUES

The `className` property allows you to change the value of the `class` attribute. If the attribute does not exist, it will be created and given the specified value.

You have seen this property used throughout the chapter to update the status of the list items. Below, you can see another way to achieve the task.

The `setAttribute()` method allows you to update the value of *any* attribute. It takes two parameters: the attribute name, and the value for the attribute.

c05/js/set-attribute.js

JAVASCRIPT

```
var firstItem = document.getElementById('one'); // Get the first item
firstItem.className = 'complete';               // Change its class attribute

var fourthItem = document.getElementsByTagName('li').item(3);// Get fourth item
el2.setAttribute('class', 'cool');              // Add an attribute to it
```

RESULT

REMOVING ATTRIBUTES

To remove an attribute from an element, first select the element, then call `removeAttribute()`. It has one parameter: the name of the attribute to remove.

Trying to remove an attribute that does not exist will not cause an error, but it is good practice to check for its existence before attempting to remove it.

In this example, the `getElementById()` method is used to retrieve the first item from this list, which has an `id` attribute with a value of `one`.

JAVASCRIPT c05/js/remove-attribute.js

```
var firstItem = document.getElementById('one'); // Get the first item
if (firstItem.hasAttribute('class')) {          // If it has a class attribute
    firstItem.removeAttribute('class');         // Remove its class attribute
}
```

RESULT

The script checks to see if the selected element has a `class` attribute and, if so, it is removed.