DOM & Javascript

Malintha Adikari

1

Document Object Model

DOM

- The Document Object Model (DOM) is a programming interface for HTML and XML documents.
- Represents the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects.
- Allows a programming language to manipulate the content, structure, and style of a website.

DOM

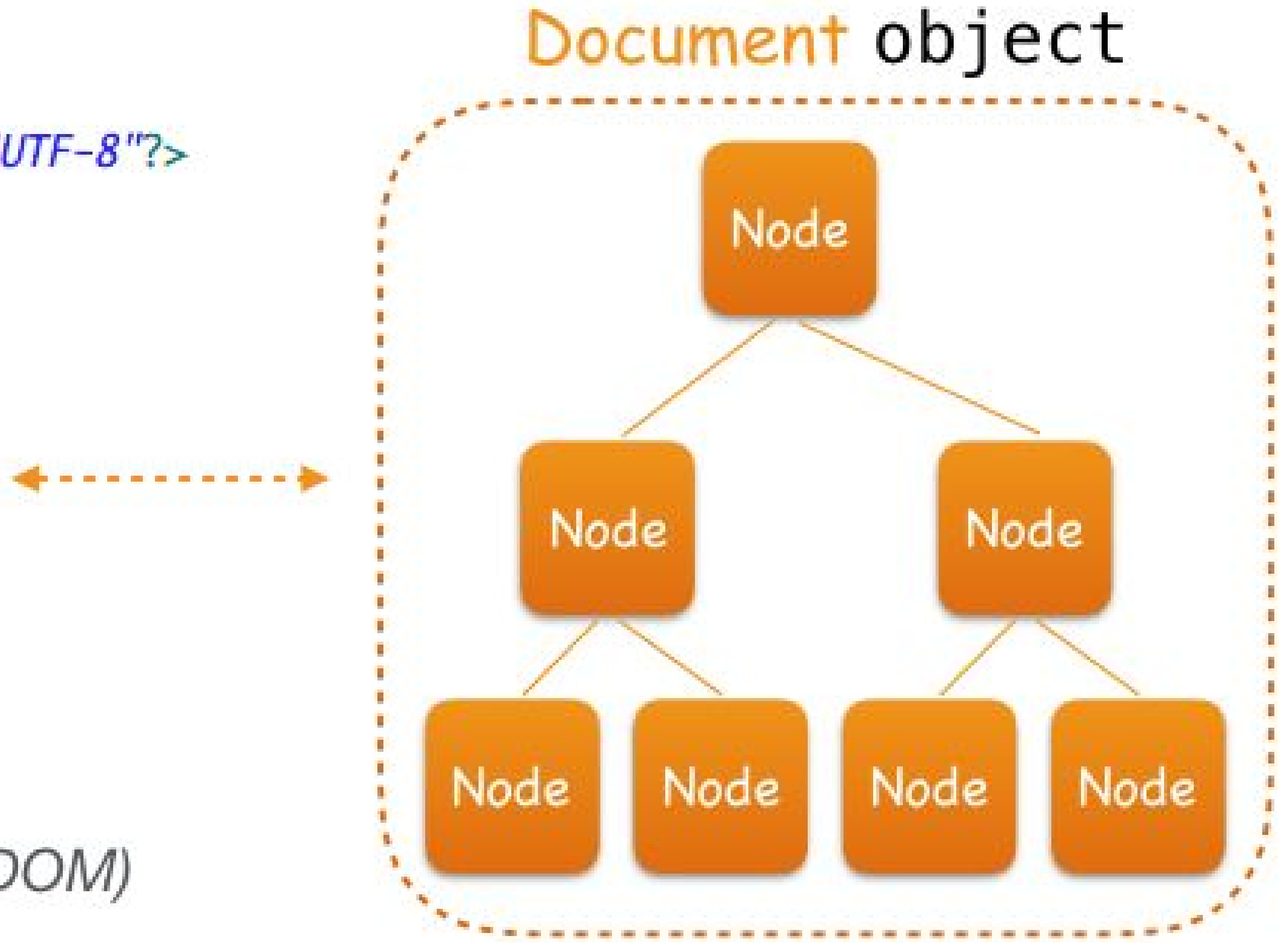
- A Web page is a document.
- This document can be either displayed in the browser window or as the HTML source.
- The Document Object Model (DOM) represents that same document so it can be manipulated.
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.
- Many browsers extend the standard, where documents may be accessed by various browsers with different DOMs.

DOM

XML document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student id="001">
        <name>Tom</name>
        <gender>male</gender>
    </student>
    <student id="002">
        <name>Jerry</name>
        <gender>male</gender>
    </student>
</students>
```

Document Object Model (DOM)



Document Object

- The document object is a built-in object that has many properties and methods used to access and modify websites
- HTML DOM document object is the owner of all other objects in web page.
- The document object represents your web page.
- To access any element in an HTML page, always start with accessing the document object.

```
> document;
```

```
Output
#document
<!DOCTYPE html>
<html lang="en">

  <head>
    <title>Learning the DOM</title>
  </head>

  <body>
    <h1>Document Object Model</h1>
  </body>

</html>
```

Document Object

The DOM Tree and Nodes

All items in the DOM are defined as nodes.

Three main node types:

- **Element nodes** - HTML element is an item in the DOM referred as an element node
- **Text nodes** - Any lone text outside of an element is a text node
- **Comment nodes** - HTML comment is a comment node.

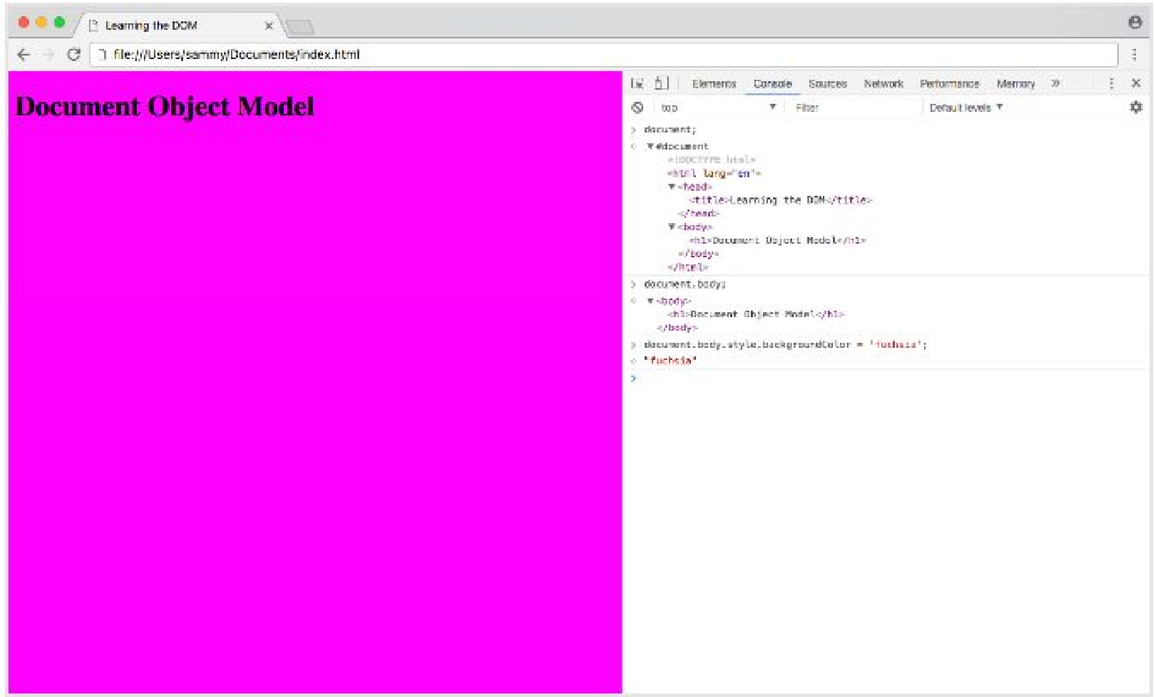
Ex: document.body.nodeType;

Modifying DOM using Javascript

- The DOM is modified by client-side JavaScript

```
> document.body.style.backgroundColor = 'fuchsia';
```

After typing and submitting the above code, you'll see the live update to the site, as the background color changes.



Output

```
<body style="background-color: fuchsia;">
  <h1>Document Object Model</h1>
</body>
```

Accessing DOM Elements

Accessing Elements by ID

document.getElementById();

In order to be accessed by ID, the HTML element must have an id attribute.

<**div** id="demo">Access me by ID</**div**>

Ex: Get the element and assign it to the demoId variable.

```
const demoId = document.getElementById('demo');
```

Accessing an element by ID is an effective way to get an element quickly in the DOM.

Accessing DOM Elements

Accessing Elements by Class

- The class attribute is used to access one or more specific elements in the DOM.
- We can get all the elements with a given class name with the getElementsByClassName() method.

```
document.getElementsByClassName();
```

Accessing DOM Elements

Accessing Elements by Tag

Access an element by tag with the getElementsByTagName() method.

```
document.getElementsByTagName();
```

Just like accessing an element by its class, getElementsByTagName() will return an array-like object of elements, and we can modify every tag in the document with a for loop.

Accessing DOM Elements

Query Selectors

Can access elements using plain javascript

document.querySelector();
document.querySelectorAll(); methods

To access a single element, we will use the querySelector() method.

Ex: <**div** id="demo-query">Access me by query</**div**>

The selector for an id attribute is the hash symbol (#).

```
const demoQuery = document.querySelector('#demo-query');
```

Accessing DOM Elements

Query Selectors

- In the case of a selector with multiple elements, such as a class or a tag, querySelector() will return the first element that matches the query.
- We can use the querySelectorAll() method to collect all the elements that match a specific query.

Traverse DOM

DOM is structured as a tree of nodes with the document node at the root and every other node (including elements, comments, and text nodes) as the various branches.

Root Nodes

- The document object is the root of every node in the DOM.
- This object is actually a property of the window object, which is the global, top-level object representing a tab in the browser.
- The document consists of what is inside of the inner window.

Traverse DOM

The root elements that every document will contain.

Property	Node	Node Type
`document`	`#document`	`DOCUMENT_NODE`
`document.documentElement`	`html`	`ELEMENT_NODE`
`document.head`	`head`	`ELEMENT_NODE`
`document.body`	`body`	`ELEMENT_NODE`

Traverse DOM

Parent Nodes

- The nodes in the DOM are referred to as parents, children, and siblings, depending on their relation to other nodes.
- The parent of any node is the node that is one level above it, or closer to the document in the DOM hierarchy.
- There are two properties to get the parent — parentNode and parentElement.

Property	Gets
`parentNode`	Parent Node
`parentElement`	Parent Element Node

- The parent of almost any node is an element node, as text and comments cannot be parents to other nodes.

Traverse DOM

Children Nodes

- The children of a node are the nodes that are one level below it.
- Any nodes beyond one level of nesting are usually referred to as descendants.
- The childNodes property will return a live list of every child of a node.

Property	Gets
`childNodes`	Child Nodes
`firstChild`	First Child Node
`lastChild`	Last Child Node
`children`	Element Child Nodes
`firstElementChild`	First Child Element Node
`lastElementChild`	Last Child Element Node

Traverse DOM

Sibling Nodes

- The siblings of a node are any node on the same tree level in the DOM.
- Siblings do not have to be the same type of node (text, element, and comment nodes can all be siblings)
- Sibling properties work the same way as the children nodes

Property	Gets
`previousSibling`	Previous Sibling Node
`nextSibling`	Next Sibling Node
`previousElementSibling`	Previous Sibling Element Node
`nextElementSibling`	Next Sibling Element Node

Make Changes in DOM

Creating New Nodes

- In a static website, elements are added to the page by directly writing HTML in an .html file.
- In a dynamic web app, elements and text are often added with JavaScript.
- The createElement() and createTextNode() methods are used to create new nodes in the DOM.

Property/Method	Description
`createElement()`	Create a new element node
`createTextNode()`	Create a new text node
`node.textContent`	Get or set the text content of an element node
`node.innerHTML`	Get or set the HTML content of an element

Make Changes in DOM

Inserting Nodes into the DOM

The methods appendChild() and insertBefore() are used to add items to the beginning, middle, or end of a parent element, and replaceChild() is used to replace an old node with a new node.

Property/Method	Description
`node.appendChild()`	Add a node as the last child of a parent element
`node.insertBefore()`	Insert a node into the parent element before a specified sibling node
`node.replaceChild()`	Replace an existing node with a new node

Make Changes in DOM

Removing Nodes from the DOM

Child nodes can be removed from a parent with **removeChild()**, and a node itself can be removed with **remove()**.

Method	Description
`node.removeChild()`	Remove child node
`node.remove()`	Remove node

Thanks!

Any questions?