# What is DOM?

Refer: <https://www.geeksforgeeks.org/dom-document-object-model/>

The Document Object Model (DOM) is a programming interface for HTML(HyperText Markup Language) and XML(Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Note: It is called a Logical structure because DOM doesn’t specify any relationship between objects.  The DOM in simple words represents the UI of your application. Everytime there is a change in the state of your application UI, the DOM gets updated to represent that change.

DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document.

With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object.

Using DOM, the JavaScript gets access to HTML as well as CSS of the web page and can also add behavior to the HTML elements.

**Why DOM is required?**

HTML is used to structure the web pages and Javascript is used to add behavior to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). DOM is basically the representation of the same HTML document but in a different format with the use of objects. Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

**Structure of DOM:** DOM can be thought of as a Tree or Forest(more than one tree). The term structure model is sometimes used to describe the tree-like representation of a document.  Each branch of the tree ends in a node, and each node contains objects  Event listeners can be added to nodes and triggered on an occurrence of a given event.

**Why called an Object Model?**

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed like tag elements with attributes in HTML.

Properties of DOM: Let’s see the properties of the document object that can be accessed and modified by the document object.

IMG_256

**Representation of the DOM**

**[Window Object](https://www.geeksforgeeks.org/properties-of-window-object/" \l ":~:text=It represents an array that,frames of a given window.&text=It returns a reference to a DOMPoint object, which represents,point in a coordinate system.&text=It provides information of the URLs visited in the current window.&text=It represents the number of frames in the current window.):** Window Object is object of the browser which is always at top of the hierarchy.  It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.

**Document object:** When an HTML document is loaded into a window, it becomes a document object. The ‘document’ object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the ‘document’ object. **Document object is property of window object.**

Form Object: It is represented by form tags.

[Link Object](https://www.geeksforgeeks.org/html-dom-link-object/): It is represented by link tags.

[Anchor Object](https://www.geeksforgeeks.org/html-dom-anchor-object/): It is represented by a href tags.

Form Control Elements:: Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

**Methods of Document Object:**

[write](https://www.geeksforgeeks.org/html-dom-write-method/)(“string”): Writes the given string on the document.

[getElementById()](https://www.geeksforgeeks.org/html-dom-getelementbyid-method/): returns the element having the given id value.

[getElementsByName()](https://www.geeksforgeeks.org/html-dom-getelementsbyname-method/): returns all the elements having the given name value.

[getElementsByTagName():](https://www.geeksforgeeks.org/html-dom-getelementsbytagname-method/)returns all the elements having the given tag name.

[getElementsByClassName()](https://www.geeksforgeeks.org/html-dom-getelementsbyclassname-method/): returns all the elements having the given class name.

# ReactJS | Virtual DOM

**DOM:** DOM stands for ‘Document Object Model’. In simple terms, it is a structured representation of the HTML elements that are present in a webpage or web-app.

DOM represents the entire UI of your application. The DOM is represented as a tree data structure. It contains a node for each UI element present in the web document.

**Updating DOM:** If you know a little about JavaScript then you might have seen people making use of ‘getElementById()’ or ‘getElementByClass()’ method to modify the content of DOM.

Every time there is a change in the state of your application, the DOM gets updated to reflect that change in the UI. Though doing things like this is not a problem and it works fine, consider a case where we have a DOM that contains nodes in a large number, and also all these web elements have different styling and attributes.  As DOM is represented as a tree itself, updating the tree here is not a costly operation indeed we have a lot of algorithms on trees to make the updates fast. What’s proving to be costly is every time the DOM gets updated, the updated element and its children have to be rendered again to update the UI of our page.

Like this each time there is a component update, the DOM needs to be updated and the UI components have to be re-rendered.

**Example:**

* javascript

|  |
| --- |
| // Simple getElementById() method  document.getElementById('some-id').innerValue = 'updated value'; |

When writing the above code in the console or in the JavaScript file, these things happen:

* The browser parses the HTML to find the node with this id.
* It removes the child element of this specific element.
* Updates the element(DOM) with the ‘updated value’.
* Recalculates the CSS for the parent and child nodes.
* Update the layout.
* Finally, traverse the tree and paint it on the screen(browser) display.

So as we know now that updating the DOM not only involves changing the content, it has a lot more attached to it.

Also recalculating the CSS and changing the layouts involves complex algorithms, and they do affect the performance.

So React has a different approach to dealing with this, as it makes use of something known as Virtual DOM.

**Virtual DOM:** React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM). So for every object that exists in the original DOM, there is an object for that in React Virtual DOM. It is exactly the same, but it does not have the power to directly change the layout of the document. Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen.

So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.

You may still wonder, “Aren’t we doing the same thing again and doubling our work? How can this be faster?” Read below to understand how things will be faster using virtual DOM.

**How Virtual DOM actually make things faster:**When anything new is added to the application, a virtual DOM is created and it is represented as a tree.

Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created. This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes.

After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.

**How Virtual DOM helps React:**In react, everything is treated as a component be it a functional component or class component. A component can contain a state.

Each time we change something in our JSX file or let’s put it in simple terms, whenever the state of any component is changed react updates its Virtual DOM tree.

Though it may sound that it is ineffective but the cost is not much significant as updating the virtual DOM doesn’t take much time. React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM.

Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed.

This process of comparing the current Virtual DOM tree with the previous one is known as **‘diffing’**. Once React finds out what exactly has changed then it updated those objects only, on real DOM.

React uses something called batch updates to update the real DOM. It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component.

We have seen that the re-rendering of the UI is the most expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI. This entire process of transforming changes to the real DOM is called **Reconciliation**