Document Object Model

**1.** What is DOM

* Dom stands for Document Object Model.
* The HTML DOM model is constructed as a tree of Objects.
* With the object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can change all the HTML elements in the page

JavaScript can change all the HTML attributes in the page

JavaScript can change all the CSS styles in the page

JavaScript can remove existing HTML elements and attributes

JavaScript can add new HTML elements and attributes

JavaScript can react to all existing HTML events in the page

JavaScript can create new HTML events in the page.

* The HTML DOM is a standard object model and programming interface for HTML. It defines:

The HTML elements as objects

The properties of all HTML elements

The methods to access all HTML elements

The events for all HTML elements

* In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
* A **property** is a value that you can get or set (like changing the content of an HTML element).
* A **method** is an action you can do (like add or deleting an HTML element).
* getElementById is a **method** used id="demo" to find the element.

, while innerHTML is a **property**. The most common way to access an HTML element is to use the id of the element.

* The easiest way to get the content of an element is by using the innerHTML property. The innerHTML property is useful for getting or replacing the content of HTML elements.
* The programming interface is the properties and methods of each object.
* The HTML DOM document object is the owner of all other objects in your web page. The document object represents your web page. If you want to access any element in an HTML page, you always start with accessing the document object.

**2.** What are the methods we use to find html element in DOM.

* getElementById, getElementByClassName, getElementByTagName

**3.** How to change html element by DOM Method & property.

* element.innerHTML =  new html content : Change the inner HTML of an element //method
* element.attribute = new value : Change the attribute value of an HTML element //method
* element.style.property = new style : Change the style of an HTML element //method
* element.setAttribute(attribute, value) : Change the attribute value of an HTML element //property

**4.** How to add & delete html element by DOM Method.

* document.createElement(element): Create an HTML element.
* document.removeChild(element) : Remove an HTML element.
* document.appendChild(element) : Add an HTML element
* document.replaceChild(new, old) : Replace an HTML element
* document.write(text) : Write into the HTML output stream

**5.** How to add a Event Handler.

* document.getElementById(id).onclick = function(){code} : Adding event handler code to an onclick event.

**6.** How to find html objects.

* The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.
* Later, in HTML DOM Level 3, more objects, collections, and properties were added.
* Read in Dom w3 school

**7.** JS Form Validation.

* HTML form validation can be done by JavaScript.
* If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:
* HTML form validation can be performed automatically by the browser.

If a form field (fname) is empty, the “required” attribute prevents this form from being submitted:

**8.** JS Data Validation.

* Data validation is the process of ensuring that user input is clean, correct, and useful.
* Typical validation tasks are:

has the user filled in all required fields?

has the user entered a valid date?

has the user entered text in a numeric field?

* Validation can be defined by many different methods, and deployed in many different ways.
* Server side validation is performed by a web server, after input has been sent to the server.
* Client side validation is performed by a web browser, before input is sent to a web server.

**9.** JS HTML Constraint Validation.

* HTML5 introduced a new HTML validation concept called **constraint validation**.
* HTML constraint validation is based on:

Constraint validation **HTML** **Input Attributes**

Constraint validation **CSS Pseudo Selectors**

Constraint validation **DOM Properties and Methods**

**10.** JS Events.

* The onload and onunload events are triggered when the user enters or leaves the page. The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information. The onload and onunload events can be used to deal with cookies.
* The onchange event is often used in combination with validation of input fields. Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.
* The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.
* The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

**11.** JS addEventListner.

* The addEventListener() method attaches an event handler to the specified element.
* The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
* You can add many event handlers to one element.
* You can add many event handlers of the same type to one element, i.e two "click" events.
* You can add event listeners to any DOM object not only HTML elements. i.e the window object.
* The addEventListener() method makes it easier to control how the event reacts to bubbling.
* When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
* You can easily remove an event listener by using the removeEventListener() method.
* Syntax : element.addEventListener(event, function, useCapture);
* The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the xmlHttpRequest object.

window.addEventListener("resize", function(){  
  document.getElementById("demo").innerHTML = sometext;  
});

* When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

element.addEventListener("click", function(){ myFunction(p1, p2);});

**12.** JS Event Propagation.

* Event propagation is a way of defining the element order when an event occurs. For example: If you have a <p> element inside a <div> element, and the user clicks on the <p> element, Then event propagation tell us which element's "click" event should be handled first?
* There are two ways of event propagation in the HTML DOM, bubbling and capturing.
* In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.
* In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.
* With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(event, function, useCapture);

* The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation

document.getElementById("myP").addEventListener("click", myFunction, false);  
document.getElementById("myDiv").addEventListener("click", myFunction, true);

**13.** JS DOM Node.

* According to the W3C HTML DOM standard, everything in an HTML document is a node. Like The entire document is a document node, Every HTML element is an element node, The text inside HTML elements are text nodes, Every HTML attribute is an attribute node (deprecated), All comments are comment nodes.
* With the HTML DOM, all nodes in the node tree can be accessed by JavaScript. Like new nodes can be created, and all nodes can be modified or deleted.
* With the HTML DOM, you can navigate the node tree using node relationships.

**14.** JS DOM Node relationship.

* With the HTML DOM, you can navigate the node tree using node relationships.
* The nodes in the node tree have a hierarchical relationship to each other. The terms parent, child, and sibling are used to describe the relationships.
* In a node tree, the top node is called the root (or root node), Every node has exactly one parent, except the root (which has no parent), A node can have a number of children, Siblings (brothers or sisters) nodes with the same parent ;;; see examp in w3s
* You can use the following node properties to navigate between nodes with JavaScript: parentNode, childNodes[*nodenumber*], firstChild, lastChild, nextSibling, previousSibling
* The value of the text node can be accessed by the node's innerHTML property.
* Accessing the innerHTML property is the same as accessing the nodeValue of the first child:

**15.** JS DOM Root Node.

* There are two special properties that allow access to the full document:

document.body - The body of the document

document.documentElement - The full document

document.getElementById("demo").innerHTML = document.body.innerHTML;

**16.** JS DOM NodeName Property.

* The nodeName property specifies the name of a node.
* nodeName is read-only
* nodeName of an element node is the same as the tag name
* nodeName of an attribute node is the attribute name
* nodeName of a text node is always #text
* nodeName of the document node is always #document
* nodeName always contains the uppercase tag name of an HTML element
* follow example w3s

**17.** JS DOM NodeType Property.

* The nodeType property is read only. It returns the type of a node.

Node Type Example

* ELEMENT\_NODE 1 <h1 class="heading">W3Schools</h1>
* ATTRIBUTE\_NODE 2 class = "heading" (deprecated)
* TEXT\_NODE 3 3 “W3Schools”
* COMMENT\_NODE 8 <!-- This is a comment -->
* DOCUMENT\_NODE 9 The HTML document itself (the parent of <html>)
* DOCUMENT\_TYPE\_NODE 10 <!Doctype html>

**18.** How to create/remove/replace new html element(nodes).

* To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element by using appendchild() method.
* Bydefault  new element added as the last child of the parent.. To avoid this we can use insertBefore() method instead of appendchild()
* To remove an HTML element, use the remove() method:
* document.getElementById("p1").remove()
* The remove() method does not work in older browsers, see the example below on how to use removeChild() instead. You have to find the parent node to remove an element: example in w3s
* To replace an element to the HTML DOM, use the replaceChild() method:

**19.** HTML Collection Object.

* An HTMLCollection object is an array-like list (collection) of HTML elements.
* The getElementsByTagName() method returns an HTMLCollection object.
* The elements in the collection can be accessed by an index number.

**20.** HTML DOM Node list object.

* A NodeList object is a list (collection) of nodes extracted from a document.
* A NodeList object is almost the same as an HTMLCollection object.
* Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like getElementsByClassName().
* All browsers return a NodeList object for the property childNodes.
* Most browsers return a NodeList object for the method querySelectorAll().
* The following code selects all <p> nodes in a document:

**21 .** HTML Collection Object vs HTML DOM Node list object.

* A NodeList object is a list (collection) of nodes extracted from a document.

A NodeList object