Internet Programming

- JavaScript Implementation
- JavaScript in HTML
- Language Basics – *Variables *operators *statements *functions
  *Data type conversions *reference types
- Document object Model -
  *browser object model *window object
  *location object *navigator object
  *screen object *history object,
- Events and Event handling
- Button elements
- validations with regular expressions.
- Introduction to Dynamic documents,
- Positioning elements
- moving elements
- elements visibility
- changing colors and fonts
- dynamic content
- Locating mouse cursor
- reacting to a mouse click
- dragging and dropping of elements

DOM(Document Object Model)-

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML and XML documents.

The DOM represents a document as a tree of nodes. It provides API that allows you to add, remove, and modify parts of the document effectively.

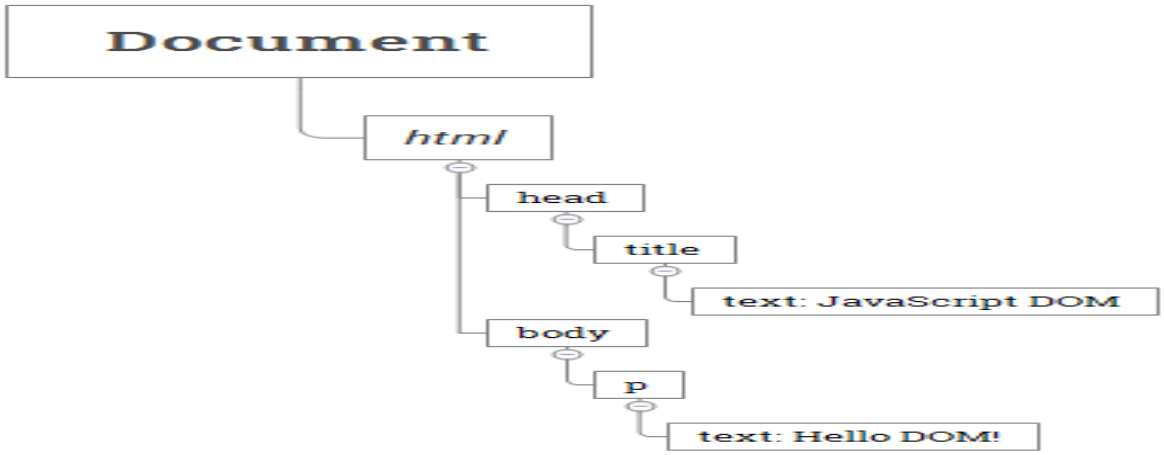
Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

A document as a hierarchy of nodes

The DOM represents an HTML or XML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
    <head>
        <title>JavaScript DOM</title>
    </head>
    <body>
        <p>Hello DOM!</p>
    </body>
</html>
```

The following tree represents the above HTML document:



In this DOM tree, the document is the root node. The root node has one child which is the <html> element. The <html> element is called the *document element*.

Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a node in the tree.

Node Types

Each node in the DOM tree is identified by a node type. JavaScript uses integer numbers to determine the node types.

The following table illustrates the node type constants:

Constant	Value	Description
Node.ELEMENT_NODE	1	An Element node like <p> or .
Node.TEXT_NODE	3	The actual Text inside an Element or Attr.
Node.CDATA_SECTION_NODE	4	A CDATASection, such as <!CDATA[[…]]>.
Node.PROCESSING_INSTRUCTION_NODE	7	A ProcessingInstruction of an XML document, such as <?xml-stylesheet … ?>.
Node.COMMENT_NODE	8	A Comment node, such as <!-- … -->.
Node.DOCUMENT_NODE	9	A Document node.
Node.DOCUMENT_TYPE_NODE	10	A DocumentType node, such as <!DOCTYPE html>.
Node.DOCUMENT_FRAGMENT_NODE	11	A DocumentFragment node.

To get the type of a node, you use the nodeType property:

node.nodeType
You can compare the nodeType property with the above constants to determine the node type. For example:

```
if (node.nodeType == Node.ELEMENT_NODE) {
    // node is the element node
}
```

The nodeName and nodeValue properties

A node has two important properties: nodeName and nodeValue that provide specific information about the node.

The values of these properites depends on the node type. For example, if the node type is the element node, the nodeName is always the same as element's tag name and nodeValue is always null.

For this reason, it's better to test node type before using these properties:

```
if (node.nodeType == Node.ELEMENT_NODE) {
    let name = node.nodeName; // tag name like <p>
}
```

Node and Element

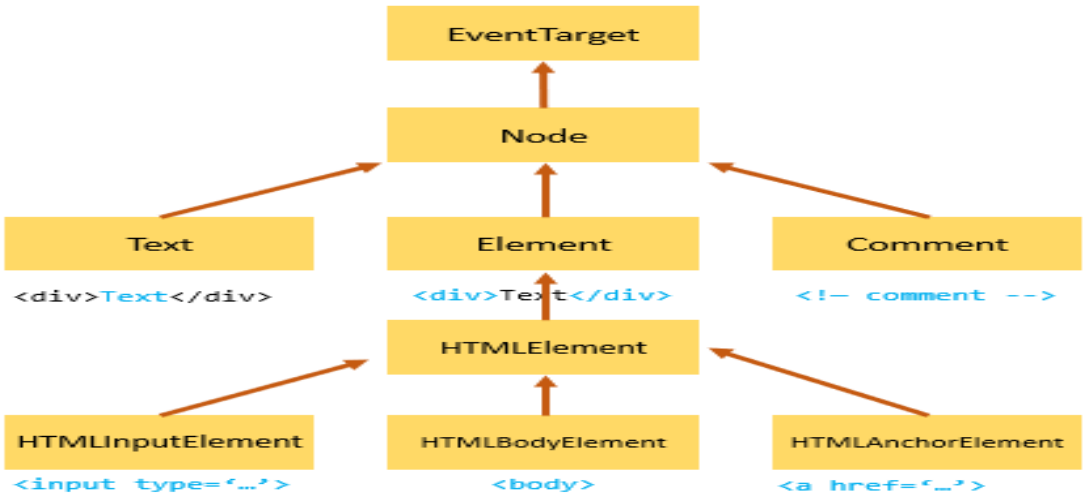
Sometime, it's easy to confuse between the Node and the Element

A node is a generic name of any object in the DOM tree. It can be any built-in DOM element such as the document. Or it can be any HTML tag specified in the HTML document like or <p>.

An element is a node with a specific node type Node.ELEMENT_NODE, which is equal to 1.

In other words, the node is generic type of the element. The element is a specific type of the node with the node type Node.ELEMENT_NODE.

The following picture illustrates the relationship between the Node and Element types:



Note : that the getElementById() and querySelector() returns an object with the Element type while getElementsByTagName() or querySelectorAll() returns NodeList which is a collection of nodes.

Node Relationships

Any node has relationships to other nodes in the DOM tree. The relationships are the same as the one described in a traditional family tree.

For example, <body> is a child node of the <html> node, and <html> is the parent of the <body> node.

The <body> node is the sibling of the <head> node because they share the same immediate parent, which is the <html> element.

The following picture illustrates the relationships between nodes.

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClass	returns all the elements having the given class name.

Name()	

JavaScript getElementById() method

An HTML element often has an id attribute like this:

```
<div id="root"></div>
```

The id is used to uniquely identify an HTML element within the document. By rules, the id root is unique within the document; no other elements can have this root id.

The id is case-sensitive. For example, the 'root' and 'Root' are totally different id.

To select the element by its id, you use the document.getElementById method.

The following shows the syntax of the getElementById() method:

```
let element = document.getElementById(id);
```

In this syntax, the id represents the id of the element that you want to select.

The getElementById() returns an Element object that describes the DOM element object with the specified id. It returns null if there is no element with that id exists.

As mentioned earlier, id is unique within a document. However, HTML is a forgiving language. If a document has more than one element with the same id, the getElementById() method returns the first one it encounters.

JavaScript getElementById() method example

Consider the following HTML document:

```
<html>
```

```
    <head>
        <title>JavaScript getElementById() Method</title>
    </head>
    <body>
        <p id="message">A paragraph</p>
    </body>
</html>
```

The document contains a <p> element that has the id attribute with the value message:

```
const p = document.getElementById('message');
console.log(p);
```

Output:

```
<p id="message">A paragraph</p>
```

Once you selected an element, you can add styles to the element, manipulate its attributes, and traversing to parent and child elements.

JavaScript getElementsByName() method

Every element on an HTML document may have a name attribute:

```
<input type="radio" name="language" value="JavaScript">
```

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

```
<input type="radio" name="language" value="JavaScript">
<input type="radio" name="language" value="TypeScript">
```

To get all elements with a specified name, you use the getElementsByName() method of the document object:

```
let elements = document.getElementsByName(name);
```

The getElementsByName() accepts a name which is the value of the name attribute of elements and returns a live NodeList of elements.

The return collection of elements is live. It means that the return elements are automatically updated when elements with the same name are inserted and/or removed from the document.

JavaScript getElementsByName() example

The following example shows a list of radio buttons that have the same name (rate).

1. When you click the Rate button, the page will show an alert dialog that displays the rating of the service such as Very Poor, Poor, OK, Good, and Very Good:

```
<.!DOCTYPE html>
<.html.>
<head>
    <meta charset="utf-8">
    <title>JavaScript getElementsByName Demo</title>
</head>
<.body.>
    <p>Please rate the service:</p>
    <p>
        <input type="radio" name="rate" value="Very poor"> Very poor
        <input type="radio" name="rate" value="Poor"> Poor
        <input type="radio" name="rate" value="OK"> OK
        <input type="radio" name="rate" value="Good"> Good
        <input type="radio" name="rate" value="Very Good"> Very Good
    </p>
    <p>
        <button id="btnRate">Submit</button>
    </p>
    <script>
        let btn = document.getElementById('btnRate');

        btn.addEventListener('click', () => {
            let rates = document.getElementsByName('rate');
```

```
        rates.forEach((rate) => {
            if (rate.checked) {
                alert(`You rated: ${rate.value}`);
            }
        })
    });
  </script.>
</body>
</html>
```

How it works:

- First, select the Rate button by its id btnRate using the getElementById() method.
- Second, hook a click event to the **Rate** button so that when the button is clicked, an anonymous function is executed.
- Third, call the getElementsByName() in the click event handler to select all radio buttons that have the name rate.
- Finally, iterate over the radio buttons. If a radio button is checked, then display an alert that shows the value of the selected radio button.

Notice that you will learn about events like click later. For now, you just need to focus on the getElementsByName() method.

document.getElementsByTagName() method

The document.getElementsByTagName() method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

document.getElementsByTagName("name")

innerHTML

The innerHTML property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
<script type="text/javascript" >

function showcommentform() {

var data="Name:<inputtype='text' name='name'><br>

Comment:<br><textarea rows='5' cols='80'></textarea>

<br><input type='submit' value='Post Comment'>";

document.getElementById('mylocation').innerHTML=data;

}

</script>

<form name="myForm">

<input type="button" value="comment" onclick="showcommentform()">

<div id="mylocation"></div>

</form>
```

Browser Object Model

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

1. window.alert("hello javatpoint");

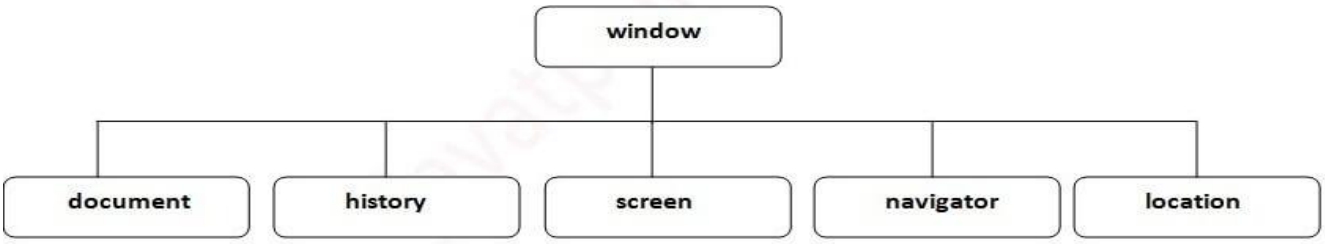
is same as:

1. alert("hello javatpoint");

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

What is the use of browser object model?

The browser object model (BOM) is a hierarchy of browser objects that are used to manipulate methods and properties associated with the Web browser itself. Objects that make up the BOM include the window object, navigator object, screen object, history, location object, and the document object.



Window Object

The window object represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.

Note: if html document contains frame or iframe, browser creates additional window objects for each frame.

Methods of window object

The important methods of window object are as follows:

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Window object represents the browser's window. It represents an open window in a browser. It supports all browsers. The document object is a property of the window object. ... All global variables are properties and functions are methods of the window object.

JavaScript History Object

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.The **window.history** object contains the browsers history.

The history object is the window property, so it can be accessed by:

1. window.history

Or,

1. history

Property of JavaScript history object

There are only 1 property of history object.

No.	Property	Description
1	length	returns the length of the history URLs.

Methods of JavaScript history object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

Example of history object

Let’s see the different usage of history object.

1. history.back();//for previous page
2. history.forward();//for next page
3. history.go(2);//for next 2nd page
4. history.go(-2);//for previous 2nd page

JavaScript Navigator Object

The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

1. window.navigator

Or,

1. navigator

Property of JavaScript navigator object-

There are many properties of navigator object that returns information of the browser

No.	Property	Description
1	appName	returns the name
2	appVersion	returns the version

3	appCodeName	returns the code name
4	cookieEnabled	returns true if cookie is enabled otherwise false
5	userAgent	returns the user agent
6	language	returns the language. It is supported in Netscape and Firefox only.
7	userLanguage	returns the user language. It is supported in IE only.
8	plugins	returns the plugins. It is supported in Netscape and Firefox only.
9	systemLanguage	returns the system language. It is supported in IE only.
10	mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.
11	platform	returns the platform e.g. Win32.
12	online	returns true if browser is online otherwise false.

Methods of JavaScript navigator object-

The methods of navigator object are given below.

No.	Method	Description
1	javaEnabled()	checks if java is enabled.
2	taintEnabled()	checks if taint is enabled. It is deprecated since JavaScript 1.2.

Example of navigator object

Let’s see the different usage of history object.

```
1. <script>
2. document.writeln("<br/>navigator.appCodeName:"+navigator.appCodeNa
   me);
3. document.writeln("<br/>navigator.appName:"+navigator.appName);
4. document.writeln("<br/>navigator.appVersion:"+navigator.appVersion);
5. document.writeln("<br/>navigator.cookieEnabled:"+navigator.cookieEnable
   d);
6. document.writeln("<br/>navigator.language:"+navigator.language);
7. document.writeln("<br/>navigator.userAgent:"+navigator.userAgent);
8. document.writeln("<br/>navigator.platform:"+navigator.platform);
9. document.writeln("<br/>navigator.onLine:"+navigator.onLine);
10.      </script>
```

JavaScript Navigator Object is used to fetch information related to the browser(useragent), like the browser name, browser version, operating system information, etc.

JavaScript Navigator object is also a property of the JavaScript Window object and can be accessed using the read only property window.navigator for the current browser window.

In JavaScript, the Navigator Object includes some properties and methods which help us to navigate from one element to another.

JavaScript Screen Object

The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

1. window.screen

Or,

1. screen

Property of JavaScript Screen Object

There are many properties of screen object that returns information of the browser

No.	Property	Description
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

Example of JavaScript Screen Object

Let’s see the different usage of screen object.

1. <script>

```
2. document.writeln("<br/>screen.width: "+screen.width);
3. document.writeln("<br/>screen.height: "+screen.height);
4. document.writeln("<br/>screen.availWidth: "+screen.availWidth);
5. document.writeln("<br/>screen.availHeight: "+screen.availHeight);
6. document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
7. document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
8. </script>
```

screen.width: 1366
screen.height: 768
screen.availWidth: 1366
screen.availHeight: 728
screen.colorDepth: 24
screen.pixelDepth: 24

The screen object, implementing the Screen interface, is a special object for inspecting properties of the screen on which the current window is being rendered. There are many properies available on this object that can be used to determine and set some properies of the client's screen. For example, Screen.

Location Object

The location object contains information about the current URL.

The location object is part of the window object and is accessed through the window.location property.

Note: There is no public standard that applies to the location object, but all major browsers support it.

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

Location Object Methods

Method	Description
assign()	Loads a new document
reload()	Reloads the current document
replace()	Replaces the current document with a new one

location object is an object that has the URL of the current page. The location object let us various parts of the URL of the current page, and also set them so that the browser will switch out the part that's designated by the property name and then go to the page with the new URL .

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML

, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description

Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event

```
<html>

<head> Javascript Events </head>

<body>

<script language="Javascript" type="text/Javascript">

<!--

function clickevent()
```

```
{
document.write("This is JavaTpoint"); }
//-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
<!--
function mouseoverevent()
{
alert("This is JavaTpoint");
}
//-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
```

```
</body>
</html>
```

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
function focusevent()
{
document.getElementById("input1").style.background=" aqua";
}
//-->
</script>
</body>
</html>
```

Keydown Event

```
<html>
```

```
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
function keydownevent()
{
document.getElementById("input1");
alert("Pressed a key");
}
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
```

```
//-->
</script>
</body>
</html>
```

THE BUTTON ELEMENT

The button object in HTML is used to represent a <button> element. The getElementById() method is used to get the button object.
Creating button object: The button object can be created using JavaScript. The document.createElement() method is used to create <button> element. After creating a button object use appendChild() method to append the particular element (such as div) to display it.

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

What Is a Regular Expression?

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

/*pattern*/*modifiers*;

Example

/w3schools/i;

Example explained:

/w3schools/i is a regular expression.

W3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

Using String search() With a String

The search() method searches a string for a specified value and returns the position of the match:

Example

Use a string to do a search for "W3schools" in a string:

```
let text = "Visit W3Schools!";
let n = text.search("W3Schools");
```

The result in *n* will be:

6

Using String search() With a Regular Expression

Example

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
let text ="Visit W3Schools";
let n = text.search(/w3schools/i);
```

The result in *n* will be:

6

Using String replace() With a String

The replace() method replaces a specified value with another value in a string:

```
let text = "Visit Microsoft!";
let result = text.replace("Microsoft", "W3Schools");
```

Try it Yourself »

Use String replace() With a Regular Expression

Example

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
let text ="Visit MicrosftInt!";
let result = text.replace(/microsoftInt/i, "DisneyWorld");
```

The result in *res* will be:

Visit DisneyWorld!

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character

\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiersdefine quantities:

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*

Using the RegExp Object

In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

Using test()

The test() method is a RegExp expression method.

It searches a string for a pattern, and returns true or false, depending on the result.

The following example searches a string for the character "e":

Example

```
const pattern = /e/;
pattern.test("The best things in life are free!");
```

Since there is an "e" in the string, the output of the code above will be:

true

You don't have to put the regular expression in a variable first. The two lines above can be shortened to one:

/e/.test("The best things in life are free!");

Using exec()

The exec() method is a RegExp expression method.

It searches a string for a specified pattern, and returns the found text as an object.

If no match is found, it returns an empty (null) object.

The following example searches a string for the character "e":

Example

/e/.exec("The best things in life are free!");

JavaScript Dynamic Document:

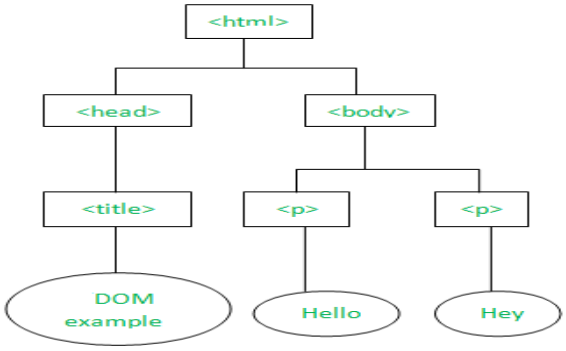
Dynamic Document Creation is the creation of a Web document from within the JavaScript. It may be created while an HTML document is being displayed, and may be influenced by features of the current Web page. As a technique, it is very useful for displaying information from Web pages and creating new HTML documents.
DDC rapidly takes you into frames, new windows, variable expressions and using DHTML to access the DOM

DHTML JavaScript/ dynamic content

DHTML stands for Dynamic HTML. Dynamic means that the content of the web page can be customized or changed according to user inputs i.e. a page that is interactive with the user. In earlier times, HTML was used to create a static page. It only defined the structure of the content that was displayed on the page. With the help of CSS, we can beautify the HTML page by changing various properties like text size, background color etc. The HTML and CSS could manage to navigate between static pages but couldn’t do anything else. If 1000 users view a page that had their information for eg. Admit card then there was a problem because 1000 static pages for this application build to work. As the number of

users increase, the problem also increases and at some point it becomes impossible to handle this problem.
To overcome this problem, DHTML came into existence. DHTML included JavaScript along with HTML and CSS to make the page dynamic. This combo made the web pages dynamic and eliminated this problem of creating static page for each user. To integrate JavaScript into HTML, a Document Object Model(DOM) is made for the HTML document. In DOM, the document is represented as nodes and objects which are accessed by different languages like JavaScript to manipulate the document.

```
<html>
<head>
<title>
 DOM example
</title>
</head>
<body>
<p id = "para1">Hello</p>
<p id =  “para2”>Hey</p>
</body>
</html>
```



HTML document include JavaScript:: The JavaScript document is included in our html page using the html tag. <src> tag is used to specify the source of external JavaScript file.

Following are some of the tasks that can be performed with JavaScript:

- Performing html tasks
- Performing CSS tasks
- Handling events
- Validating inputs

Positioning Element:

The position property in CSS tells about the method of positioning for an element or an HTML entity. There are five different types of position property available in CSS:

- Fixed
- Static
- Relative
- Absolute
- Sticky

The positioning of an element can be done using the top, right, bottom, and left properties. These specify the distance of an HTML element from the edge of the viewport. To set the position by these four properties, we have to declare the positioning method.

Fixed: Any HTML element with position: fixed property will be positioned relative to the viewport. An element with fixed positioning allows it to remain at the same position even we scroll the page. We can set the position of the element using the top, right, bottom, left.

Static: This method of positioning is set by default. If we don't mention the method of positioning for any element, the element has the position: static method by default. By defining Static, the top, right, bottom and left will not have any control over the element. The element will be positioned with the normal flow of the page.

Relative: An element with position: relative is positioned relatively with the other elements which are sitting at top of it. If we set its top, right, bottom, or left, other elements will not fill up the gap left by this element.

Absolute: An element with position: absolute will be positioned with respect to its parent. The positioning of this element does not depend upon its siblings or the elements which are at the same level.

Sticky: Element with position: sticky and top: 0 played a role between fixed & relative based on the position where it is placed. If the element is placed at the middle of the document then when the user scrolls the document, the sticky element starts scrolling until it touches the top. When it touches the top, it will be fixed at that place in spite of further scrolling. We can stick the element at the bottom, with the bottom property.

Hide or show HTML elements using visibility property in JavaScript

The visibility property is used to hide or show the content of HTML elements. The visibility property specifies that the element is currently visible on the page.

The ‘hidden’ value can be used to hide the element. This hides the element but does not remove the space taken by the element, unlike the display property.

Syntax:

element.style.visibility = 'hidden';
element.style.visibility = 'visible';

Example:
html

```
<!DOCTYPE html>
<html>

<head>

    <style>
        .container {
            height: 80px;
            width: 250px;
            border: 2px solid black;
            background-color: green;
            color: white;
        }
    </style>
</head>

<body>
    <div class="container">
        <h1>GeeksforGeeks</h1>
    </div>


<p>
        Click the buttons to show or hide the green box
    </p>


    <button onclick="showElement()">
            Show Element
    </button>
```

```
    <button onclick="hideElement()">
        Hide Element
    </button>

    <script type="text/javascript">
        function showElement() {
            element = document.querySelector('.container');
            element.style.visibility = 'visible';
        }

        function hideElement() {
            element = document.querySelector('.container');
            element.style.visibility = 'hidden';
        }
    </script>
</body>

</html>
```

Locating the Mouse Cursor

- The coordinates of the element that caused an event are available in the clientX and clientY properties of the event object. These are relative to upper left corner of the browser display window. ...
- In order to locate the mouse cursor when the mouse button is clicked, use the click event.

JavaScript | Coordinates of mouse:

The top left corner of the screen is (0, 0) i,e, X and Y coordinate is (0, 0). This means that vertical zero is topmost point and horizontal zero is the leftmost point.
So, the task is to find the coordinates of the mouse over the screen.
It can be find using the clientX and clientY property:

- ClientX: It gives the horizontal coordinate of the event.
- ClientY: It gives the vertical coordinate of the mouse.

Syntax:

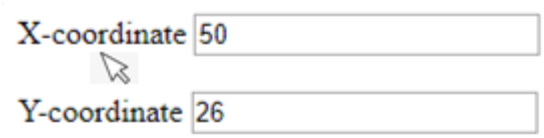
1. For X coordinate: event.ClientX
2. For Y coordinate: event.ClientY

Below is the required implementation:
javascript

```
<html>

<head>
   <script>

     // coordinate function that calculate the X
     // coordinate and Y coordinates
     function coordinate(event) {

     // clientX gives horizontal coordinate
     var x = event.clientX;

     // clientY gives vertical coordinates
     var y = event.clientY;
     document.getElementById("X").value = x;
     document.getElementById("Y").value = y;
  }
    </script>
</head>

<!-- onmousemove event is called when the mouse
     pointer is moving over the screen -->

<!-- calling of coordinate function -->

<body onmousemove="coordinate(event)">

   <!-- X coordinate is stored in X-coordinate variable -->
   X-coordinate
   <input type="text" id="X">
   <br>
   <br>

   <!-- Y coordinate is stored in Y-coordinate variable -->
```

```
   Y-coordinate
   <input type="text" id="Y">

</body>
</html>
```

Output:



HTML Drag and Drop

Drag and Drop is a very interactive and user-friendly concept that makes it easier to move an object to a different location by grabbing it. This allows the user to click and hold the mouse button over an element, drag it to another location, and release the mouse button to drop the element there. In HTML 5 Drag and Drop are much easier to code and any element in it is draggable. Drag and Drop Events: There are various Drag and Drop events, some of them are listed below:

ondrag: It is used to use when the element or text selection is being dragged in HTML.
ondragstart: It is used to call a function, drag(event), that specifies what data to be dragged.
ondragenter: It is used to determine whether or not the drop target is to accept the drop. If the drop is to be accepted, then this event has to be canceled.

ondragleave: It occurs when the mouse leaves an element before a valid drop target while the drag is occurring.
ondragover: It specifies where the dragged data can be dropped.
ondrop: It specifies where the drop has occurred at the end of the drag operation.
ondragend: It occurs when the user has finished dragging an element.
Procedure for Drag and Drop:

Step 1: Make an object draggable
First set the draggable attribute to true for that element to be draggable <img draggable = “true”>
Then, specify what should happen when the element is dragged.
The ondragstart attribute calls a function, drag(event), that specifies what data to be dragged. The dataTransfer.setData() method sets the data type and the value of the dragged data
The event listener ondragstart will set the allowed effects (copy, move, link, or some combination).
Step 2: Dropping the Object
The ondragover event specifies where the dragged data can be dropped. By default, data/elements cannot be dropped in other elements. To allow a drop, it must prevent the default handling of the element. This is done by calling the event.preventDefault() method

Finally, the drop event, which allows the actual drop to be performed
Example 1: This example shows the drag & drop of an image in HTML.
HTML

```
<!DOCTYPE HTML>
<html>
<head>
   <style>
   #getData {
      width: 250px;
      height: 200px;
      padding: 10px;
      border: 1px solid #4f4d4d;
   }
   </style>
   <script>
   function allowDrop(even) {
```

```
        even.preventDefault();
    }
    function drag(even) {
        even.dataTransfer.setData("text", even.target.id);
    }
    function drop(even) {
        even.preventDefault();
        var fetchData = even.dataTransfer.getData("text");
        even.target.appendChild(document.getElementById(fetchData));
    }
    </script>
</head>
<body>
    <h3>Drag the GeekforGeeks image into the rectangle:</h3>
    <div id="getData"
         ondrop="drop(event)"
         ondragover="allowDrop(event)">
      </div>
    <br>
    <img id="dragData"
         src="gfg.png"
         draggable="true"
         ondragstart="drag(event)"
         width="250"
         height="200">
</body>
</html>
```

Output:

Dragging the image into the box

The Data Transfer Object: The data transfer property is used when the whole process of Drag and Drop happens. It is used to hold the dragged data from the source and drop it to the desired location. These are the properties associated with it:

dataTransfer.setData(format, data): It is used to set the data to be dragged.

dataTransfer.clearData(format): It is used for calling this function with no argument that clears all the data. Calling it with a format argument removes only that specific data.

dataTransfer.getData(format): It returns the data of the specified format. If there is no such data, returns the empty string.

dataTransfer.types: This property returns an array of all the formats that were set in the dragstart event.

dataTransfer.files: It is used to return all the files that are to be dropped. dataTransfer.setDragImage(element, x, y): It is used to display an existing image as the drag image instead of the default image alongside the cursor. The coordinates specify the drop location.

dataTransfer.addElement(element): It is used to add the specified element to be drawn as a drag feedback image.

dataTransfer.effectAllowed(value): It will tell the browser that only the listed type(s) of operations are allowed for the user. The property can be set to the following values: none, copy, copyLink, copyMove, link, linkMove, move, all, and uninitialized.

dataTransfer.dropEffect(value): It is used to control the feedback that the user is given during the dragenter and dragover events. When the user hovers over a target element, the browser's cursor will indicate what type of operation is going to take place (e.g. a copy, a move, etc.). The effect can take on one of the following values: none, copy, link, move.

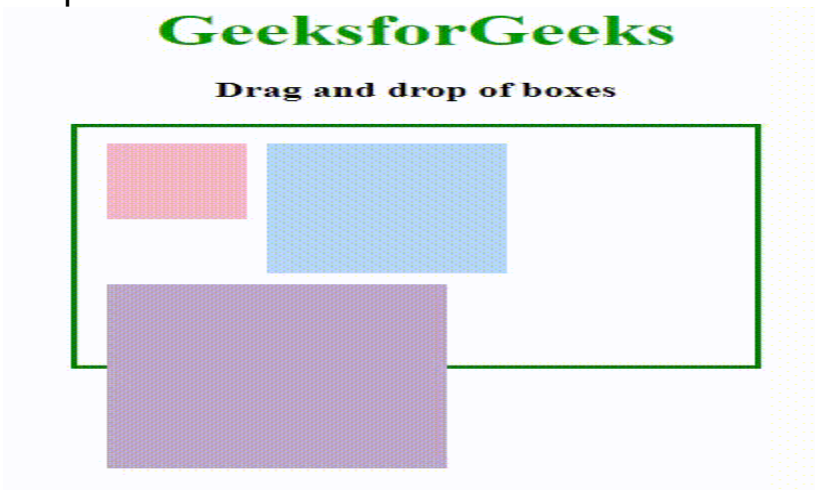
Example 2: This example illustrates the use of the draggable property of the element.
HTML

```
<!DOCTYPE HTML>
<html>

<head>
    <title>Drag and Drop box</title>
    <script>
    function allowDrop(ev) {
        ev.preventDefault();
    }

    function dragStart(ev) {
        ev.dataTransfer.setData("text", ev.target.id);
    }

    function dragDrop(ev) {
        ev.preventDefault();
        var data = ev.dataTransfer.getData("text");
        ev.target.appendChild(document.getElementById(data));
    }
    </script>
    <style>
    #box {
        margin: auto;
        width: 50%;
        height: 200px;
        border: 3px solid green;
        padding: 10px;
    }

    #box1,
    #box2,
    #box3 {
        float: left;
        margin: 5px;
        padding: 10px;
    }

    #box1 {
        width: 50px;
        height: 50px;
```

```
        background-color: #F5B5C5;
    }

    #box2 {
        width: 100px;
        height: 100px;
        background-color: #B5D5F5;
    }

    #box3 {
        width: 150px;
        height: 150px;
        background-color: #BEA7CC;
    }

    p {
        font-size: 20px;
        font-weight: bold;
        text-align: center;
    }

    .gfg {
        font-size: 40px;
        color: #009900;
        font-weight: bold;
        text-align: center;
    }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>

<p>Drag and drop of boxes</p>


    <div id="box">
        <div id="box1" draggable="true"
            ondragstart="dragStart(event)">
        </div>
```

```
        <div id="box2" draggable="true"
            ondragstart="dragStart(event)">
        </div>
        <div id="box3" ondrop="dragDrop(event)"
            ondragover="allowDrop(event)">
        </div>
    </div>
</body>

</html>
```

Explanation:
Start the drag by setting the draggable property of the element to be dragged to true.
Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method.
Call event.preventDefault() method to allow the dropping of elements in other elements by preventing the default handling of the element.
The element is stored in the variable data which we append into the drop element.
Output:



Example 3: This example illustrates the image drag and drop.
HTML

```
<!DOCTYPE HTML>
<html>
<head>
    <script>
    function allowDrop(ev) {
```

```
        ev.preventDefault();
    }

    function dragStart(ev) {
        ev.dataTransfer.setData("text", ev.target.id);
    }

    function dragDrop(ev) {
        ev.preventDefault();
        var data = ev.dataTransfer.getData("text");
        ev.target.appendChild(document.getElementById(data));
    }
    </script>
    <style>
    .div1 {
        width: 240px;
        height: 75px;
        padding: 10px;
        border: 1px solid black;
        background-color: #F5F5F5;
    }

    p {
        font-size: 20px;
        font-weight: bold;
    }

    .gfg {
        font-size: 40px;
        color: #009900;
        font-weight: bold;
    }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>

<p>Image Drag and Drop in boxes</p>
```

```
    <div class="div1"
        ondrop="dragDrop(event)"
        ondragover="allowDrop(event)">
     <img id="drag1"
         src=
"https://media.geeksforgeeks.org/wp-content/cdn-uploads/Geek_logi_-low_res.png"
         draggable="true"
         ondragstart="dragStart(event)"
         width="220"
         height="70">
       </div>
    <br>
    <div class="div1"
        ondrop="dragDrop(event)"
        ondragover="allowDrop(event)">
    </div>
</body>
</html>
```

Output:

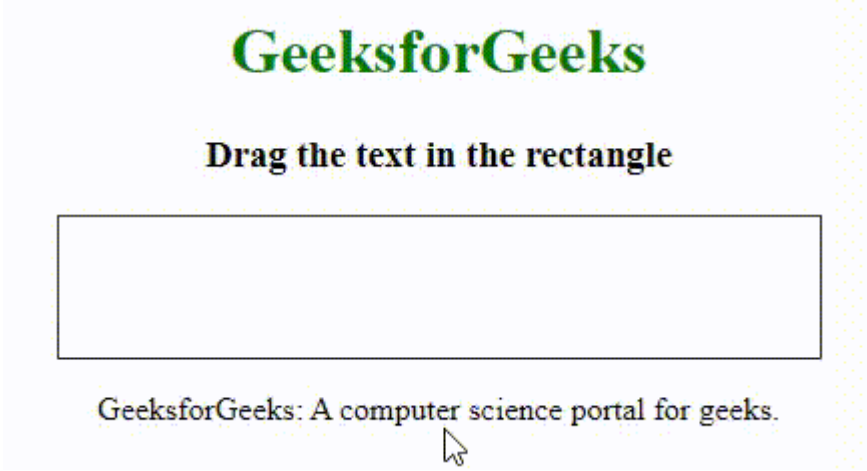
Dragging the image into another box

Example 4: This example demonstrates dragging the text in the rectangular box in HTML.
HTML

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Draggable Text</title>
    <style>
    .dropbox {
        width: 350px;
        height: 40px;
        padding: 15px;
        border: 1px solid #292828;
    }

    h1 {
        color: green;
    }
    </style>
    <script>
    function droppoint(event) {
        var data = event.dataTransfer.getData("Text");
        event.target.appendChild(document.getElementById(data));
        event.preventDefault();
    }

    function allowDropOption(event) {
        event.preventDefault();
    }

    function dragpoint(event) {
        event.dataTransfer.setData("Text", event.target.id);
    }
    </script>
</head>

<body>
    <center>
        <h1>GeeksforGeeks</h1>
        <h3>Drag the text in the rectangle</h3>
```

```
        <div class="dropbox"
            ondrop="droppoint(event)"
            ondragover="allowDropOption(event)">
          </div>
        <p id="drag1"
           draggable="true"
           ondragstart="dragpoint(event)">
       GeeksforGeeks: A computer science portal for geeks.
          </p>


    </center>
</body>
</html>
```

Output:



Dragging the text into the box