Array function

1. concat() Method

The JavaScript array concat() method **combines two or more arrays and returns a new string**. This method doesn't make any change in the original array.

Syntax:- **array.concat(arr1,arr2,....,arrn)**

Return:- A new array object that represents a joined array.

```
<script>
      var arr1=["C","C++","Python"];
      var arr2=["Java","JavaScript","Android"];
      var result=arr1.concat(arr2);
     document.writeln(result);
</script>
```

Output

C,C++,Python,Java,JavaScript,Android

2. push() Method

The JavaScript array push() method **adds one or more elements to the end of the given array**. This method changes the length of the original array.

Syntax:- array.push(element1,element2....elementn)

Return:- The original array with added elements.

```
<script>
var arr=["AngularJS","Node.js"];
arr.push("JQuery");
document.writeln(arr);
</script>
```

Output

AngularJS,Node.js,JQuery

3. pop() Method

The JavaScript array pop() method **removes the last element** from the given array and return that element. This method changes the length of the original array.

Syntax:- array.pop()

Return:- The last element of given array.

```
<script>
      var arr=["AngularJS","Node.js","JQuery"];
      document.writeln("Orginal array: "+arr+"<br>");
      document.writeln("Extracted element: "+arr.pop()+"<br>");
      document.writeln("Remaining elements: "+ arr);
</script>
```

Output

Orginal array: AngularJS,Node.js,JQuery
Extracted element: JQuery
Remaining elements: AngularJS,Node.js

4. reverse() Method

The JavaScript array reverse() method **changes the sequence of elements** of the given array and returns the reverse sequence. In other words, the arrays last element becomes first and the first element becomes the last. This method also made the changes in the original array.

Syntax:- array.reverse()

Return:-The original array elements in reverse order.

```
<script>
      var arr=["AngulaJS","Node.js","JQuery"];
      var rev=arr.reverse();
     document.writeln(rev);
</script>
```

Output

```
JQuery,Node.js,AngulaJS
```

5. shift() Method

Javascript array **shift()**method removes the first element from an array and returns that element.

Syntax:- array.shift();
Return:- Returns the removed single value of the array.

```
<script type = "text/javascript">
      fruits = ["Banana", "Orange", "Apple", "Mango"];
      a= fruits.shift();
      document.write("Removed element is : " + a);
 </script>
```

Output

```
Removed element is : Banana
```

6. unshift() Method

Add new items to the beginning of an array.
The unshift() method adds new items to the beginning of an array, and returns the new length.

Syntax:- *array*.unshift(*item1*, *item2*, ..., *itemX*)
Parameter Details: - item1, ..., itemX − The elements to add to the front of the array.
Return value: - This function returns the new length of the array after inserting the arguments at the beginning of the array.

```
<script>
      fruits = ["Banana", "Orange", "Apple", "Mango"];
      document.write("<br>Original Array==>   "+fruits)

      fruits.unshift("Lemon", "Pineapple")
      document.write("<br>Using Unshift==>  "+fruits)
</script>
```

Output

```
Original Array==> Banana,Orange,Apple,Mango
Using Unshift==> Lemon,Pineapple,Banana,Orange,Apple,Mango
```

7. sort() Method

The JavaScript array sort() method is used to arrange the array elements in some order. By default, sort() method follows the ascending order.

Syntax: array.sort(compareFunction)
Parameter Details:- compareFunction − Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.
Return: An array of sorted elements

```
<script>
        fruits = ["Banana", "Orange", "Apple", "Mango"];
        document.write("<br>Using Sort==>  "+ fruits.sort())
</script>
```

Output

Using Sort==> Apple,Banana,Mango,Orange

8. reverse() Method

Javascript array reverse() method reverses the element of an array. The first array element becomes the last and the last becomes the first.

Syntax: array.reverse();
Return Value: Returns the reversed single value of the array.

```
<script type = "text/javascript">
    fruits = ["Banana","Apple", "Orange", "Mango"];
    document.write("<br>Using Sort==>  "+ fruits.reverse())
</script>
```

Output

Using Sort==> Mango,Orange,Apple,Banana

JavaScript Events

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something.

1. onclick Event Type

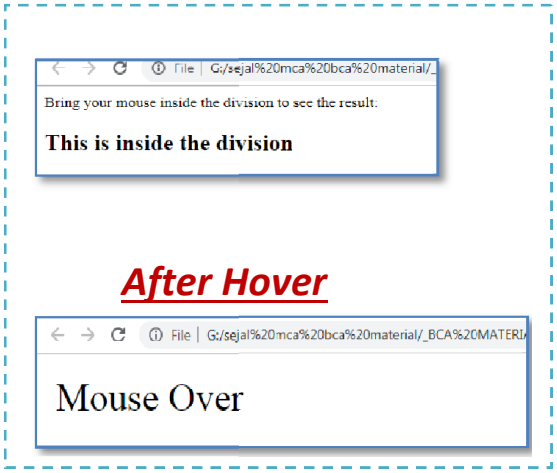
- This is the most frequently used event type which occurs when a user clicks the left button of his mouse.
- You can put your validation, warning etc., against this event type.

```
<html> <head>
      <script type = "text/javascript">
        function sayHello()
           {
              alert("Hello World")
           }
      </script>
   </head> <body>
      <p>Click the following button and see result</p>
      <form>
           <input type = "button" onclick = "sayHello()" value = "Say Hello" />
       </form>
    </body></html>
```

2. onmouseover and onmouseout

- These two event types will help you create nice effects with images or even with text as well.
- The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element.

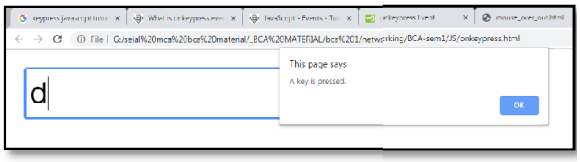
```
<html>
<head>
   <script type = "text/javascript">
           function over()
           {
           document.write ("Mouse Over");
           }
           function out()
           {
          document.write ("Mouse Out");
       }
   </script>
  </head>  <body>
      <p>Bring your mouse inside the division to see the result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
          <h2> This is inside the division </h2>
      </div>
  </body>
</html>
```



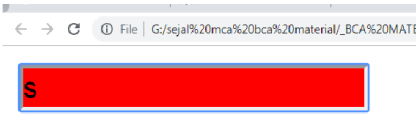
3. onkeypress event

The onkeypress event triggers when a key is pressed and released. You can try to run the following code to learn how to work with *onkeypress* event in JavaScript

```
<script>
        function sayHello()
        {
           alert("A key is pressed.")
         }
  </script>
<input type = "text" onkeypress = "sayHello()">
```



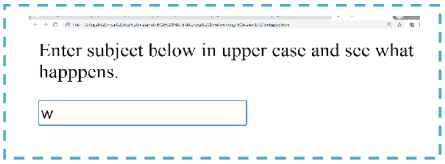

```
<html>
<body>
        <input type="text" id="demo">
        <script>
                document.getElementById("demo").onkeypress = function() { myFunction() };

                function myFunction()
                {
                        document.getElementById("demo").style.backgroundColor = "red";
                }
        </script>
</body>
</html>
```



4. onkeyup event

The onkeyup event triggers when a key is released. You can try to run the following code to learn how to work with onkeyup event in JavaScript

```
<script>
    function sayHello()
   {
        var str = document.getElementById("subject");
        str.value = str.value.toLowerCase();
    }
</script>
<p>Enter subject below in upper case and see what happpens.</p>
<input type = "text" onkeyup = "sayHello()" id = "subject">
```

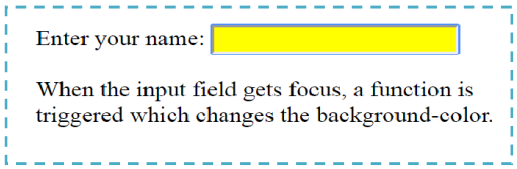


5. onfocus Event

This onfocus attribute works when the element gets focused. This event attribute is mostly used with <input>, <select>, <a> elements.

This event attribute is supported by all HTML elements excepts <base>, <bdo>,
, <head>, <html>, <iframe>, <meta>, <param>, <script>, <style>, and <title> elements.

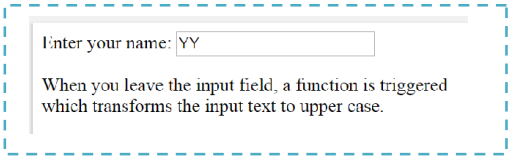
```
Enter your name: <input type="text" onfocus="myFunction(this)">
<p>When the input field gets focus, a function is triggered which changes the background-color.</p>

<script>
function myFunction(x)
 {
    x.style.background = "yellow";
}
</script>
```



6. onblur Event

- The onblur event occurs when **an object loses focus.**
- The onblur event is most often **used with form validation code** (e.g. when the user leaves a form field).
- **Tip:** The onblur event is the opposite of the onfocus event.

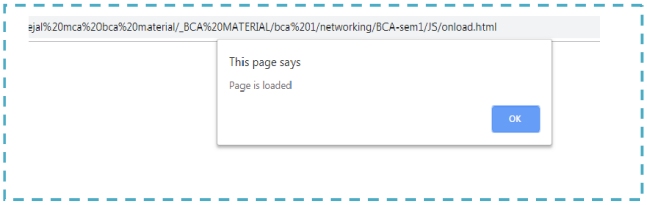
```
Enter your name:    <input type="text" id="fname" onblur="myFunction()">
<p>When you leave the input field, a function is triggered which transforms the input text to upper
case.</p>
<script>
function  myFunction()
{
 var x = document.getElementById("fname");
 x.value = x.value.toUpperCase();
}
</script>
```



7. onload Event

- The onload event occurs **when an object has been loaded**.
- onload is most often **used within the <body> element** to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).
- 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.

```
<html>
<body onload="myFunction()">
<h1>Hello World!</h1>
<script>
        function myFunction()
        {
                alert("Page is loaded");
        }
</script></body></html>
```



8. onchange Event

- The onchange event occurs when the value of an element has been changed.
- For radiobuttons and checkboxes, the onchange event occurs when the checked state has been changed.

Supported HTML tags: <input type="checkbox">, <input type="color">, <input type="date">, <input type="datetime">, <input type="email">, <input type="file">, <input type="month">, <input type="number">, <input type="password">, <input type="radio">, <input type="range">, <input type="search">, <input type="tel">, <input type="text">, <input type="time">, <input type="url">, <input type="week">, <select> and <textarea>

9. onsubmit Event

- The **onsubmit** event occurs when a form is submitted.
- The **onsubmit** event is an event that occurs when you try to submit a form.
- You can put your form validation against this event type. The following example shows how to use onsubmit.
- Here we are calling a **validate()** function before submitting a form data to the web server. If **validate()** function returns true, the form will be submitted, otherwise, it'll not submit the data.

```
<script>
        function sub()
        {
                x= document.getElementById("nm").value
                alert("value is::"+x)
        }
 </script>
<form onsubmit="sub()">
  <input type="text" id="nm">
  <input type = "submit" value = "Submit" />
</form>
```

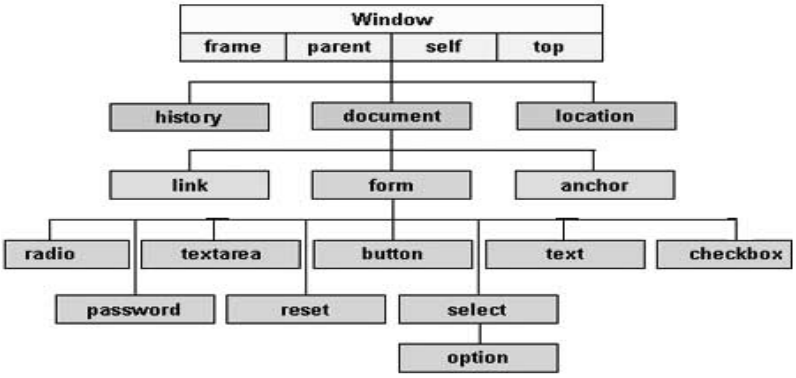
Document Object Model

- A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

- The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

  - **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
  - **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
  - **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
  - **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects –



Methods of document object

- We can access and change the contents of document by its methods.
- The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the document.
writeln("string")	writes the given string on the document 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.
getElementsByClassName()	returns all the elements having the given class name.

```
<script type="text/javascript">
Function  printvalue()
{
        x = document.getElementById("name").value;
        document.getElementById("demo").innerHTML = x;
}
</script>
 <form>
        Enter Name:<input type="text" id="name"/>
        <input type="button" onclick="printvalue()" value="print name"/>
 </form>
 <p id="demo"></p>
```

Output

Enter Name: Pooja Patel [print name]

Pooja Patel

History Object

- The history object contains the URLs visited by the user (within a browser window).
- The history object is part of the **window object** and is accessed through the **window.history** property.
- When you launch the web browser and open a new webpage, the web browser creates a new entry in its history stack.
- If you navigate to another webpage, the web browser also creates a new entry in the history stack.

- The history stack stores the current page and previous pages that you visited.
- To manipulate the history stack, you use the history object which is a property of the window object:

The history object provides three methods for navigating between pages in the history stack:

1. **back() (Move backward)**

- This behaves like you click the Back button in the toolbar of the web browser.
- To move backward through history, you use the back() method:

```
window.history.back();
```
OR
```
history.back();
```

2. **forword() (Move forward)**

- It works like when you click the **Forward** button.
- Similarly, you can move forward by using the forward() method:

```
history.forward();
```

3. **go() (Move to a specific URL in the history)**

- To move to a specific URL in the history stack, you use the go() method.
- The go() method accepts an integer that is the relative position to the current page. The current page's position is 0.

For example,

To move backward you use:

```
history.go(-1);
```

To move forward a page, you just call:

```
history.go(1)
```

To refresh the current page, you either pass 0 or no argument to the go() method:

```
history.go(0);
history.go();
```

To determine the number of URLs in the history stack, you use the length property:

```
history.length
```

```
here use window.history.back();<br>
<button onclick="goBack()">Go Back</button>
<script>
function goBack()
{
 window.history.back();
}
</script>
```

JavaScript - Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

1. Basic Form Validation

- First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring.
- The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
    // Form validation code will come here.
    function validate() {

      if( document.myForm.Name.value == "" ) {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
      }
```

```
      if( document.myForm.EMail.value == "" )
      {
        alert( "Please provide your Email!" );
        document.myForm.EMail.focus() ;
        return false;
      }

      if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 ) {

        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
      }

      if( document.myForm.Country.value == "-1" ) {
        alert( "Please provide your country!" );
        return false;
      }

      return( true );
   }
</script>
```

2. Data Format Validation

- Now we will see how we can validate our entered form data before submitting it to the web server.
- The following example shows how to validate an entered email address.
- An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```
<script type = "text/javascript">
   <!--
      function validateEmail() {
        var emailID = document.myForm.EMail.value;
        atpos = emailID.indexOf("@");
        dotpos = emailID.lastIndexOf(".");

        if (atpos < 1 || ( dotpos - atpos < 2 )) {
          alert("Please enter correct email ID")
          document.myForm.EMail.focus() ;
          return false;
        }
        return( true );
      }
</script>
```