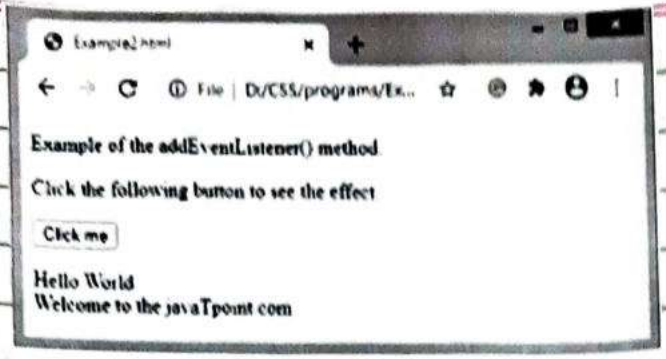


Event Bubbling or Event Capturing

Bubbling, the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first and then the outermost element's event will be handled.

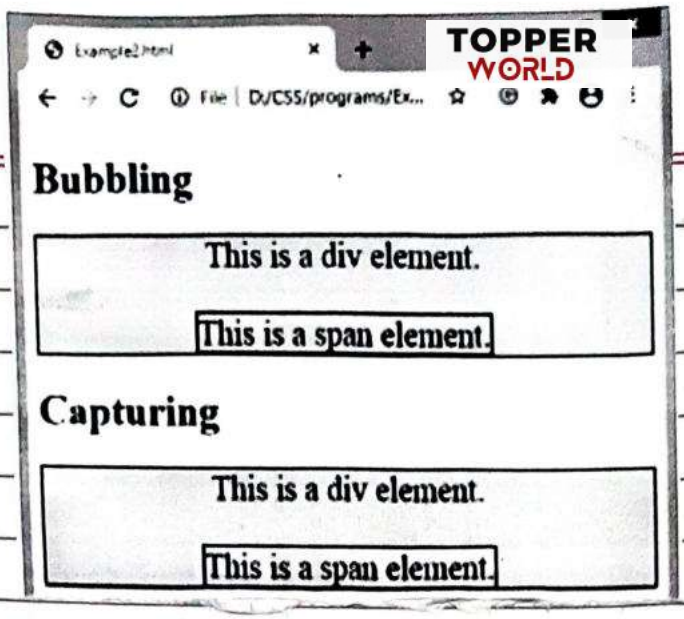
In Capturing the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

```
addEventListener(event, function, useCapture);
```

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
 div {
 background-color: lightblue;
 border: 2px solid red;
 font-size: 25px;
 text-align: center
 }
  span {
```

```
border: 2px solid blue;
}
</style>
</head>
<body>
<h1> Bubbling</h1>
<div id = "d1">
This is a div element.
<br><br>
<span id ="s1"> This is a span element.</span>
</div>
<h1> Capturing </h1>
<div id = "d2"> This is a div element.
<br><br>
<span id = "s2"> This is a span element.</span>
</div>
<script>
document.getElementById("d1").addEventListener("dblclick",
function(){alert("You have double clicked on div element")},
false);
document.getElementById("s1").addEventListener("dblclick",
function(){alert("you have double clicked on span
element")}, false);
document.getElementById("d2").addEventListener("dblclick",
function(){alert('you have double clicked on div element')},
true);
document.getElementById("s2").addEventListener("dblclick",
function(){alert('you have double clicked on span
element')}, true);
</script>
</body>
</html>
```



JavaScript Onclick event

This event can be dynamically added to any element. It supports all HTML elements except <html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>,
, <meta>, and <param>. It means we cannot apply the onclick event on the given tags.

In HTML, we can use the onclick attribute and assign a Javascript function to it. We can also use the JavaScript's addEventListener() method and pass a click event to it for greater flexibility.

In HTML

```
<element onclick = "fun()">
```

In JavaScript

```
Object.onclick = function(){myScript};
```

In Javascript by using the addEventListener() method

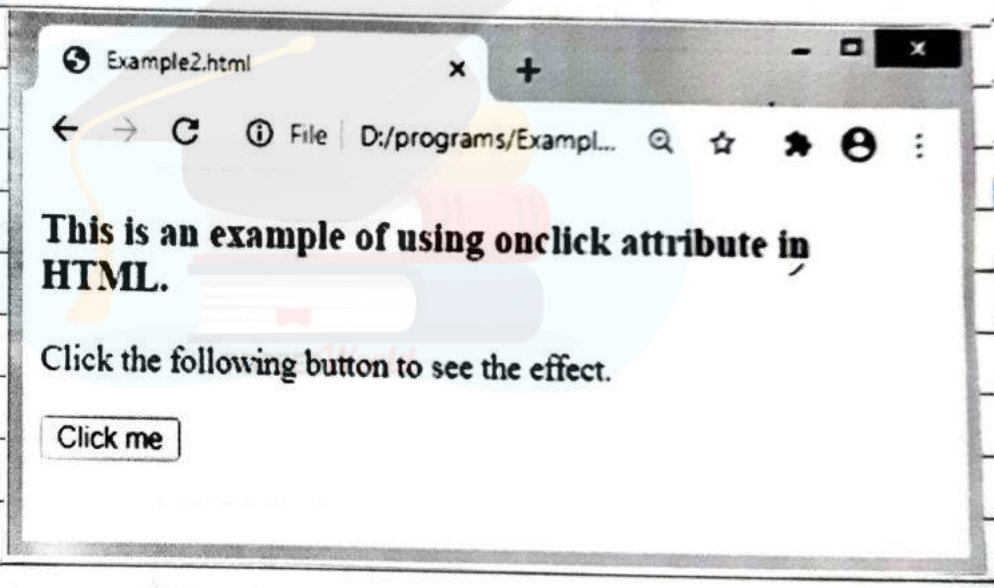
```
Object.addEventListener("click", myScript);
```

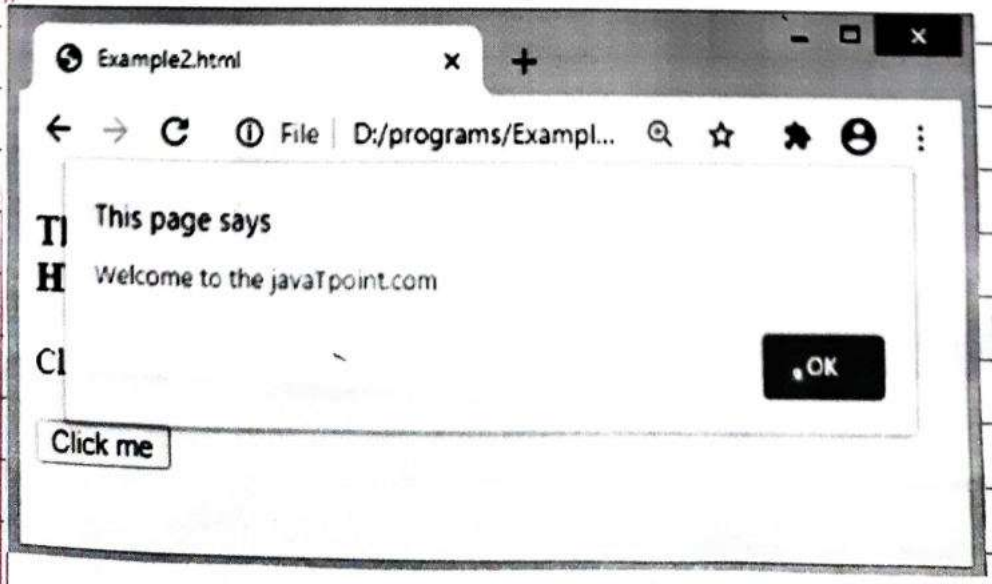
Example

```
<!DOCTYPE html>
<html>
```

```
<head>
<script>
function fun(){
alert("Welcome to the javaTpoint.com");
}
</script>
</head>
<body>
<h3>This is an example of using onclick attribute in
    HTML.</h3>
<p>Click the following button to the see effect.</p>
<button onclick="fun()">Click me</button>
</body>
</html>
```

Output





Example - Using JavaScript

```
<!DOCTYPE html>
<html>
<head>
<title>onclick event</title>
</head>
<body>
<h3>This is an example of using onclick event.</h3>
<P>Click the following text to see the effect.</P>
<P id = "para">Click me</P>
<script>
document.getElementById("para").Onclick =
    function(){
fun()
};
function fun() {
document.getElementById("para").innerHTML =
    "Welcome to the javaTpoint.com";
document.getElementById("para").style.Color="blue";
document.getElementById("para").style.
        backgroundcolor = "yellow";
document.getElementById("para").style.fontSize=
        "25px";
document.getElementById("para").style.border=
        "4px solid red";
}
</script>
</body>
</html>
```

JavaScript dblclick event

The dblclick event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's addEventListener() method to fire the double click event.

In HTML

We can use the ondblclick attribute to create a double click event.

In HTML

```
<element on dblclick = "fun()">
```

In JavaScript

```
object.ondblclick = function() { myScript };
```

In JavaScript by using the addEventListener() method

```
object.addEventListener("dblclick", myScript);
```

Example - Using ondblclick attribute in HTML

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id = "heading" ondblcick = "fun()">Hello World :):)</h1>
<h2 > Double click the text "Hello World" to see the
    effect. </h2>
<P> This is an example of using the <b> Ondblclick
    </b> attribute. </P>
<script>
```

```
function fun() {
document.getElementById("heading").innerHTML =
    "Welcome to the javaTpoint.com";
}
</script>
</body>
</html>
```

Example - Using JavaScript

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id = "heading"> Hello World :):)</h1>
<h2> Double Click the text "Hello World" to see
    the effect. </h2>
<P> This is an example of Creating the double
    Click event using JavaScript. </p>
<Script>
document.getElementById("heading").ondblclick =
 function() { fun() };
function fun() {
document.getElementById("heading").innerHTML =
    "Welcome to the javaTpoint.com";
}
</script>
</body>
</html>
```

Example - Using JavaScript's addEventListener() method

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
 <h1 id = "heading">Hello World :):) </h1>
 <h2> Double Click the text "Hello World" to see the
      effect </h2>
 <p> This is an example of creating the double
  Click event using the <b> addEventListener() method </b>
 <script>
 document.getElementById("heading") addEventListener
        ("dblclick", fun);
 function fun() {
 document.getElementById("heading"). innerHTML =
        "Welcome to the javaTpoint. com";
 }
 </script>
 </body>
 </html>
```

JavaScript onload

In HTML, the Onload attribute is generally used with the <body> element to execute a script once the content (including CSS files, images, scripts, etc.) of the Webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

```
Window.onlolad = fun()
```

Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "utf-8">
<title> Window.onload()</title>
<style type = "text/css">
#bg {
width : 200px;
height : 200px;
border: 4px solid blue;
}
</style>
<script type = "text/javascript">
Window.onload = function(){
document.getElementById("bg").style.background
        Color = "red";
document.getElementById("bg").Style.Width = "300px";
document.getElementById("bg")style.height = "300px";
}
</script>
</head>
<body>
<h2> This is an example of Window.Onload()</h2>
<div id = "bg" ></div>
<body>
<html>
```

JavaScript onresize event

In HTML, We can use the onresize attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a resize event to it for greater flexibility.

In HTML

```
<element onresize = "fun()" >
```

In JavaScript

```
Object.onresize = function() { my script };
```

In JavaScript by using the addEventListener() method

```
Object.addEventListener("resize", my script);
```

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
var i = 0;
function fun() {
var res = "Width=" + Window.outerWidth+"<br>" +"Height = "
        + Window.outerHeight;
document.getElementById("para").innerHTML = res;
var res1 = i += 1;
document.getElementById("S1")innerHTML=res1;
}
</script>
</head>
<body onresize = "fun()" >
<h3>This is an example of using onresize attribute. </h3>
```

```
<p> Try to resize the browser's Window to See
    the effect. </p>
 <P id = "para" ></p>
 <P> You have resized the Window <span id =
    "S1" >0 </span >times. </p>
    </body>
    </html>
```

Example - Using JavaScript

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h3> This is an example of using JavaScripts
      onresize event. </h3>
<p> Try to resize the browser's Window to
      See the effect. </p>
<P id = " para "> </p>
<P> You have resized the Window <spanid=
   "S1" >0 </span> times. </p>
<script>
document.getElementsByTagName("BODY")[0].
   Onresize = function() { fun()};
Var i =0;
function fun() {
Var res = "Width=" + Window.OuterWidth + "<br>"+
   "Height=" + Window.OuterHeight;
document.getElement.ById("para").innerHTML=res;
Varres1 = i + = 1;
document.get Element By Id('S1'). innerHTML=res1;
}
```

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

Example : Using addEventListener() method

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h3> This is an example of using JavaScript's addEventListener
    ()method. </h3>
<p> Try to resize the browser's Window to see the
    effect. </p>
<p id = "para"></p>
<p> you have resized the Window <span id ="s1">0
    </span > times. </p>
<script>
Window.addEventListener("resize", fun);
Var i = 0;
function fun() {
Varres = "Width =" + Window.outerWidth + "<br>" +
    "Height =" + Window.outer Height;
document.getElementById("para").innerHTML =res;
Var res1 = i +=1;
document.getElementById("s1").innerHTML= res1;
}
</script>
</body>
</html>
```

Exception Handling in JavaScript

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code.

In exception handling
A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.
The thrown exception is handled by Wrapping the code into the try block statements will get executed.
Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types Of Errors

While coding, there can be three types of errors in the code:

1. Syntax Error :
When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.

2. Runtime Error:
When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors

are known as Exceptions. Thus, exception handlers are used for handling runtime errors.

3. Logical Error:
An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an Object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. name:
This is an object property that sets or returns an error name.

2. message:
This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. EvalError:
It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the JS string code.

2. InternalError:
It creates an instance when the js engine throws an internal error.

3. RangeError:
It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.

4. ReferenceError:
It creates an instance for the error that occurs when an invalid reference is de-referenced.

5. SyntaxError:
An instance is created for the syntax error that may occur while parsing the eval().

6. TypeError:
When a variable is not a valid type, an instance is created for such an error.

7. URIError:
An instance is created for the error that occurs when invalid parameters are passed in encodeURI() or decodeURI().

JavaScript try... Catach

try{}Statement;
Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the Catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

Catch{} statement:
The block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the buit-in handler. This block executes only when any error-prone code needs to be handled in the try block. otherwise, the catch block is skipped.

try.... catch example

```
<html>
<head> Exception Handling </br></head>
<body>
<script>
try{
Var a = ["34", "32", "5", "31", "24", "44", "67"]; //a is an array
document.Write(a); //displays elements of a
document.Write(b); //b is undefined but still trying to fetch
its Value. Thus Catch block Will be invoked
}catch(e){
alert("There is error Which Shows"+e.message); //Handling
    error
}
</script>
</body>
</html>
```

Throw Statement

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statements is executed, the statements present after it will not execute. The control

Will directly pass to the catch block.

throw example With try....catch

```
<html>
<head> Exception Handling </head>
<body>
<script>
try {
   throw new Error ('This is the throw keyword'); //
    user-defined throw statement
}
 Catch (e) {
  document. Write (e.message); //This Will generate
    an error message
 }
 </script>
 </body>
  </html>
```

try...catach...finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present execute. It does not care for the output too.

try... Catch... finally example

```
<html>
<head> Exception Handling </head>
<body>
<script>
```

```
try {
Var a = 2;
if (a == 2)
document.Write("Ok")
}
Catch(Error){
document.Write("Error found"+e.message);
}
finally{
document.Write("Value of a is 2")
}
</script>
</body>
</html>
```

JavaScript this keyword

The this keyword is a reference variable that refers to the current object. Here, we will learn about this keyword with help of different examples.

Example

```
<Script>
Var address =
{
Company:"Javatpoint,"
City: "Noida",
State: "UP",
full Address: function()
{
return this.company + " " + this.city + " " + this.state;
}
```

```
};
Var fetch = address.fullAddress();
 document.Writeln(fetch);
 </script>
```

Output:
Javatpoint Noida UP

The following ways can be used to know which object is referred by this keyword.

Global Context

In global context, variables are declared outside the functio
Here, this keyword refers to the window object.

```
<script>
Var Website = "Javatpoint";
function Web()
{
document.Write(this.Website)
}
Web();
</script>
```

The call() and apply() method

The call() and apply() method allows us to write a method that can be used on different objects.

```
<script>
Var emp_address = {
   fullAddress: function(){
      return this.company +" "+ this.city +" " +this.state;
   }
}
```

```
Var address = {
   Company: "Javatpoint",
   City : " Noida",
   State: " UP"
}
document.Writeln(emp_address.fullAddress.call(address));
document.Writeln(emp_address.fullAddress.apply(address));</script>
```

The bind() Method

The bind()method Was introduced in ECMAScript 5. It creates a new function Whose this keyword refers to the provided Value. With a given sequence of arguments.

```
<Script>
Var lang = " Java";
function lang_name(call)
{
  Call();
};
Var obj = {
   lang."JavaScript";
   language.function()
   {
   document.Writeln(thislang+"is a popular Programming language");
   }
};
lang_name(obj.language);
lang_name(obj.languge.bind(obj));
</Script>
```

JavaScript Debugging

Sometimes a code may contain mistakes. Being a scripting language JavaScript didn't show any error message in a browser. But

these mistakes can affect the Output.

JavaScript Debugging Example

Using console.log()method
The console.log() method displays the result in the console of the browser. If there is any mistake in the code, it generates the error message.

Example

```
<script>
 X = 10;
 Y = 15;
 Z = x+y;
 Console.log (z);
 Console.log(a);//a is not intialized
 </script>
```

Using debugger keyword
In debugging, generally we set breakpoints to examine each line of code step by step. There is no requirement to perform this task manually in JavaScript.

```
<script>
 x = 10;
 Y = 15;
 Z = x+y;
 debugger;
 document.Write(z);
 document.Write(a);
 </script>
```

JavaScript Hoisting

Hoisting is a mechanism in JavaScript that moves the

declaration of Variables and functions at the top. So, in JavaScript We can use Variables and functions before declaring them.

JavaScript Hoisting Example
Here, We Will use the Variable and function before declaring them.

JavaScript Variable Holsting

```
<script>
 x = 10;
document.Write(x);
Var x;
</script>
```

JavaScript Function Hoisting

```
<script>
document.Write (Sum(10,20));
 function sum(a,b)
 {
return a+b;
 }
 </script>
```

JavaScript Strict Mode

Being a scripting language, sometimes the JavaScript code displays the correct result even it has some errors. To overcome this problem We Can use the Javascript Strict mode.
The JavaScript provides "use strict"; expression to enable the strict mode. If there is any silent error or mistake in the code, it throws an error.

Example
TO print sum of two numbers.

```
<script>
console.log(sum(10,20));
function sum(a,a)
{
"use strict";
return a+a;
}
</script>
```

JavaScript Promise

Promises in real-life express a trust between two or more persons and an assurance that a particular thing will surely happen. In javascript, a Promise is an object which ensures to produce a single value in the future. Promise in javascript is used for managing and tacking asynchronous operations.

Terminology of Promise

A promise can be present in one of the following states:

1. pending:
The pending promise is neither rejected nor fulfilled yet.

2. fulfilled:
The related promise action is fulfilled sucessfully

3. rejected:
The related promise action is failed to be fulfilled

4. settled:
Either the action is fulfilled or rejected.
Thus, a promise represents the completion of an asynchornous operation with its result. It can be either successful completion of the promise, or its failure, but eventually completed. Promise uses a then() which is executed only after the completion of the promise resolve.

Promises of Promise

A JavaScript Promise promises that:

1. Unless the current execution of the js event loop is not completed (success or failure) callbacks will never be called before it.
2. Even if the callbacks with then() are present, but they will be called only after the execution of the asynchronous operations completely.
3. When multiple callbacks can be included by invoking then() many times, each of them will be executed in a chain, ie, one after the other, following the sequence in which they were inserted.

Methods in Promise

The functions of Promise are executable almost on every trending Web browsers such as chrome, Mozilla, Opera, etc.

1. Promise.resolve(promise)
This method returns promise only if Promis.constructor == Promise.

2. Promise.resolve(thenable)
Makes a new promise from thenable containing then().

3. Promise.resolve(obj)
Makes a promise resolved for an object.

4. Promise.reject(obj)
Makes a promise rejection for the object.

5. Promise.all(array)
Makes a promise resolved when each item in an array is fulfilled or objects when items in the array are not fulfilled.

6. Promise.race(array)
If any item in the array is fulfilled as soon, it resolves the promise, or if any item is rejected as soon it rejects the promise.

Constructor in Promise

new Promise(function(resolve,reject){});
Here, resolve(thenable) denotes that the promise will be resolved with then()
Resolve(obj) denotes promise will be fulfilled with the object.
Reject (Obj) denotes promise rejected with the object.

Promise Implementation

```
<html>
<head>
<h2> Javascript Promise </h2>
</br> </head>
<body>
<script>
```

```
Var p = new Promise (function(resolve, reject) {
Var x = 2+3;
if (x == 5)
   resolve ("executed and resolved successfully");
else
  reject("rejected");
});
P. then(function (from Resolve) {
document. Write("Promise is" + from resolve);
}). catch (function( from Reject) {
document. Write("Promise is" + from Reject);
});
</script>
</body>
</html>
```

In the above Promise implementation the Promise constructor takes an argument that callbacks the function. This Callback function takes two arguments, ie.,

1. Resolve:
When the promise is executed successfully. the resolve argument is invoked, which provides the result.

2. Reject:
When the promise is rejected, the reject argument is invoked. which results in an error.

It means either resolve is called or reject is called. Here, then() has taken one argument which will execute, if the promise is resolved. Otherwise, catch() will be called with the rejection of the promise.

Advantages of using Promises

1. A better option to deal With asynchronous operations.
2. Provides easy error handling and better code readability.

Javascript Compare dates

In the previous section, we discussed the date methods as well the constructors. Here, With the help of those methods, We will learn to compare dates.
Basically there are different Ways to Which we can Compare dates, Such as;

1. Comparing two dates With one another.
2. Comparing date With time.
3. Comparing dates using getTime()

Comparing two dates With One another

Example

```
<html>
<head> Comparing Dates</br></head>
<body>
<Script>
function compare()
{
Vard1 = new Date ('2020-01-23');//YYYY-mm-dd
Vard2 = new Date ('2020-01-21');//YYYY-mm-dd
if ( d1 > d2)
{
document.Write ("True, First date is greater than Second
                    date");
}
```

```
else if (d1 < d2)
{
document.Write("False.Second date is smaller than the first.");
}
else
{
document. Write("Both date are same and equal");
}
}
Compare(); // invoking compare()
</script>
</body>
</html>
```

Comparing date With time

Example

Comparing different dates With different things

```
<html>
<head> Comparing Date and time </br></head>
<body>
<Script>
Vard1 = new Date("APr 17, 2019, 12:10:10"); //mm dd.yyyyhh.mm:ss
Vard2 = new Date ("Dec 1, 2019 12:10:30"); //mm. ddyyyyhh.mmss
if (d1 > d2)
{
document.Write (false, d1 date and time is smaller than
        d2 date and time");
}
else if (d1 < d2)
{
document. Write ("True, d2 is greater than in terms of
        both time and date");
```

```
}
else
document.Write("Both date and time are same
                and equal");
}
</script>
</body>
</html>
```

Comparing date With getTime()

Example

Comparing current date and time With a given date and time.

```
<html>
<head> Comparing Dates</br></head>
<body>
 <script>
Var d1 = new Date("2019-10-10, 10:10:10");//Yyyy-mm-dd
                                          hh:mm:ss
Var currentdate = new Date();//fetch the current date
                              value
if(d1.getTime()< currentdate.getTime())
{
document.Write("True. currentdate and time are greater
                than d1");
}
else if(d1.getTime()> currentdate.getTime())
{
document.Write("False");
}
else
```

```
{
document.Write("True, equal");
}
</script>
</body>
</html>
```

JavaScript array.length property

The length property can also be used to set the number of elements in an array. We have to use the assignment operator in conjuction with the length property to set an array's length.

The array.length property in JavaScript is same as the array.size() method in jQuery. In Javascript, it is invalid to use array.size() method so. We use array.length property to calculate the size of an array.

```
array.length

array.length = number
```

Example

```
<html>
<head>
<title> array.length </title>
</head>
<body>
<h3> Here, we are finding the length of an array. </h3>
<script>
Var arr = new Array (100,200, 300, 400, 500, 600);
document.write("The elements of array are" +arr);
```

```
document.Write("<br> The length of the array is:" +
                arr.length);
</script>
</body>
</html>
```

JavaScript alert()

The alert() method displays an alert box with a message and an ok button.
The alert() method is used when you want information to come through to the user.

Note
The alert box takes the focus away from the current window, and forces the user to read the message.

```
alert(message)
```

Example

```
<html>
 <head>
  <script type = "text/javascript">
     function fun() {
     alert ("This is an alert dialog box");
    }
  </script>
 </head>
 <body>
  <p> Click the following button to see the effect</p>
  <form>
   <input type = "button" value = "Click me" onclick =
                "fun();" />
```

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

JavaScript eval() function

The eval() function in JavaScript is used to evaluate the expression. It is JavaScirpt's global function. Which evaluates the specified string as JavaScript code and executes it. The parameter of the eval() function is a string. If the parameter represents the statements, eval() evaluates the statements. If the parameter is an expression, eval() evaluates the expression. If the parameter of eval() is not a string, the function returns the parameter unchanged.

```
eval(String)
```

Example

```
<html>
<head>
<script>
Var a=10, b=20, C=30, sum, mul, sub;
Sum = eval("a+b+c");
mul = eval("a*b*c");
Sub = eval("a-b");
document.Write(sum+"<br>");
document.Write(mul+"<br>");
document.Write(sub);
</script>
</head>
<body>
</body>
```

```
</html>
```

Output

```
60
6000
-10
```

JavaScript closest()

The closest() method in Javascript is used to retrieve the closest ancestor, or parent of the element matches the selectors. If there is no ancestor found, the method returns null.
This method traverses the element and its parents in the document tree, and the traversing continues until the first node is found that matches the provided selector string.

```
targetElement.closest(selectors);
```

Example

```
<!DOCTYPE html>
 <html>
  <head>
  </head>
  <body>
  <div id = "div1"> This is the first div element.
    <h3 id = "h" >This is a heading inside the div.</h3>
  <divid = "div2"> This is the div inside the div element.
  <divid = "div3> This is the div element inside the
            second by div element.</div>
   </div>
```

```
</div>
  <script>
var val1 = document.getElementById("div3");
  var01 = val1.closest("# div1");
  var02 = val1 closest("div div");
  var03 = val1 closest("div > div");
  var04 = val1 closest(":not(#div3)");
    console.log(01);
    console.log(02);
    console.log(03);
    console.log(04);
      </script>
    </body>
    </html>
```

JavaScript continue statement

The continue statment in JavaScript is used to jumps over an iteration of the loop. Unlike the break statement, the continue statement breaks the current iteration and continues the execution of next iteration of the loop. It can be used in for loop, and do-while loop. When itis used in a while loop, then it jumps back to the condition. It is used in for loop, the flow moves to the update expression.

When we apply the continue statement, the program's flow immediately moves the conditional expression, and if the condition is true, then the next iteration will be started; otherwise, the control exits the loop.

```
continue;
OR
continue[label]; // Using the label reference
```

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <h1> Example of the Continue statement in JavaScript</h1>
    <h3> Here, you can see that "a = = 4" is skipped.</h3>
    <p id = "para">
    </p>
    <script>
    Var res = " ";
 Var a;
 for (a = 1; a <= 7; a++) {
   if (a = = 4){
   Continue;
 }
  res += "The Value of a is : " +a + "<br>";
}
  document.getElementById("para").innerHTML = res;
        </script>
    </body>
    </html>
```

JavaScript getAttribute() method

The getAttribute() method is used to get the value of an attribute of the particular element. If the attribute exists, it returns the string representing the value of the corresponding attribute. If the corresponding attribute does not exist, it will return an empty string or null.

element.getAttribute(attributename)

Example

```
<!DOCTYPE html>
<html>
<head>
<title>
The getAttribute Method
</title>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
</h1>
<h2>
Example of the getAttribute() Method
</h2>
<div id = "div1" style = "background-color: yellow; front-size:
25px; color: Red; border: 2px solid red; ">
This is first div element.
</div>
<br>
<div id = "div2" style = "background-color: lightblue;
size 25px; color: blue; border : 2px solid blue; ">
This is second div element.
</div>
<br>
<button onclick = "fun()">
Click me!
</button>
<p id = "p"></p>
<p id = 'p1"></p>
```

```
<Script>
function fun() {
Var val = document.getElementById("div1").getAttribute
                    ("Style");
document.getElementById("p") innerHTML = val;
Var val1 = document.getElementById ("div2").
          getAttribute("style");
document.getElementById("p1").innerHTML = val1;
}
</script>
<|body>
</html>
```

JavaScript hide elements

In JavaScript, we can hide the elements using the style.display or by using the style.visibility. The visibility property in JavaScript is also used to hide and element. The difference between the style.display and style.visibility is when using visibility: hidden, the tag is not visible, but space is allocated. Using display: none, the tag is also not visible, but there is no space allocated on the page.
In HTML, we can use the hidden attribute to hide the specified element. When the hidden attribute in HTML sets to true, the element is hidden, or when the value is false, the element is visible.

Using Style.hidden

```
document.getElementById("element").style.display = "none";
```

Using Style.visibility

```
document.getElement By Id ("element").style.visibility = "none";
```

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>
Style.display
</title>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
</h1>
<h2>
<div id="div" style="background-color:yellow; font-size:25px;
color:red; border:2 px solid red;">
This is the div element.
</div>
<p id="p">This is a paragraph element.</p>
<button onclick="fun()" id="btn">
click me!
</button>
<script>
function fun() {
document.getElementById("div").Style.display = "none"
document.getElementById("p").Style.display = "none"
}
</script>
</body>
</html>
```

JavaScript prompt() clialog box

The prompt() method in JavaScript is used to display a prompt

box that prompts the user for the input. It is generally used to take the input from the user before entering the page. It can be written without using the window perfix. When the prompt box pops up, we have to click "Ok" or "Cancel" to proceed.

The box is displayed using the prompt() method. Which takes two arguments: The first argument is the label which displays in the text box, and the second argument is the default string, which displays in the textbox. The prompt box consists of two buttons, Ok and Cancel. It returns null or the string entered by the user. When the user clicks "Ok", the box returns the input value. otherwise, it returns null on clicking "Cancel".

```
Prompt(message, default)
```

Example

```
<html>
<head>
<Script type = "text/javascript">
function fun() {
Prompt("This is a prompt box", "Hello World");
}
</script>
</head>
<body>
<p> Click the following button to see the effect </p>.
<form>
<input type = "button" Value = "Click me" onclick = "fun()"/>
</form>
</body>
</html>
```

Output

After the execution of the above code and clicking the clickme button, the output will be -

JavaScript removeAttribute() method

The method is used to remove the specified attribute from the element. It is different from the removeAttributeNode() method. The removeAttributeNode() method removes the particular Attr object, but the removeAttribute() method removes the attribute with the specified name.

```
element.removeAttribute(attributename)
```

Example

```
<!DOCTYPE html>
<html>
<head>
<title>
The removeAttribute Method
</title>
<style>
jtp{
color: red;
background-color: yellow;
}
</style>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
```

```
</h1>
<h2>
Example of the removeAttribute Method
</h2>
<p id = "para" class = "jtp">
This is a paragraph element.
</p>
<p id = "para1" class = "jtp">
This is second paragraph element
</p>
<button onclick = "fun()">
Click me!
</button>
<script>
function fun(){
document.getElementById("para").removeAttribute("class");
document.getElementById("para1").removeAttribute("class");
}
</script>
</body>
</html>
```

JavaScript reset

In HTML, we can use the reset button to reset the form. In this article, we are discussing how to reset the form using JavaScript.

In JavaScript, the reset() method does the same thing as the HTML reset button. It is used to clear all the values of the form elements. It can be used to set the valus to defalult. It does not require any parameter values and also does not return any value.

formElement.reset()

Example

```
<!DOCTYPE html>
<html>
<head>
<title>reset()method</title>
</head>
<body style="text-align:center;">
<div style="background:pink;">
<font color="red" size="6px">
<b>Example of the reset() method</b>
</font>
</div>
<div style="background:lightblue;">
<form id="myForm" action="#"Style="font-size:20px;">
<p>First Name:<input type="text" id="fname"/></p>
<p>Last Name:<input type="text" id="lname"/></p>
<p>E-mail Id:<input type="email"id="email"/></p>
<p>Age:  <input type="number"id="age"/></p>
<input type="submit">
<input type="button"value="Reset data"onclick="fun()"/>
</form>
</div>
<script>
function fun(){
document.getElementById("myForm").reset();
}
</script>
</body>
</html>
```

JavaScript return

The return statement is used to return a particular value from the function to the function caller. The function will stop executing when the return statement is called. The return statement should be the last statement in a function because the code after the return statement will be unrechable.

We can return primitive values and Object types by using the return statement.

We can also return multiple values using the return statement. It cannot be done directly. We have to use an Array or Object to return multiple values from a function.

```
return expression;
```

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h2> Welcome to the javaTpoint.com </h2>
<h3> Example of the Javascript's return Statement </h3>
    <script>
Vares = fun(12,30);
function fun(x,Y)
{
return x * y;
}
document.Write(res);
  </script>
```

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

JavaScript String split()

As the name implies, the split() method in JavaScript splits the string into the array of substrings, puts these substrings into an array, and returns the new array. If does not change the original string.

When the string is empty, rather than returning an empty array, the split() method returns the array with an empty string. The empty array is returned When both string and separator are empty strings

```
String.split(separator, limit)
```

Example

```
<html>
<head>
<script>
Var Str = 'Welcome to the javaTpoint.com'
Var arr = Str.split(" ", 3);
document.Write(arr);
</script>
</head>
<body>
</body>
</html>
```

Output

Welcome, to, the

JavaScript typeof operator

The JavaScript typeof operator is used to return a string that represents the type of JavaScript for a given value. It returns the data type of the operand In the form of a string. The operand can be a literal or a data structure like a function, an object or a variable.

```
type.of operand
or
typeof(operand)
```

The possible returns values of the typeof operator are tabulated as follows:

Type of the operand	Result
Object	"object"
number	"number"
string	"string"
boolean	"boolean"
function	"function"
undefined	"undefined"

Example

```
<html>
<head>
<script>
document.write(typeof(45)+"<br>"); //results: "number"
document.write(typeof(-90)+"<br>"); //results: "number"
document.write(typeof(0)+"<br>"); //results: "number"
document.write(typeof(22.6)+"<br>"); //results: "number"
document.write(typeof(Infinity)+"<br>"); //results: "number"
```

```
document.write(typeof(NaN));//results:"number". Although
        Nan is "Not-A-Number"
</script>
</head>
<body>
</body>
</html>
```

Output

After the execution of the above code, the output will be -

```
number
number
number
number
number
number
```

JavaScript ternary operator

During coding in any language, we use various ways to handle conditional situations. The common one is the use of if statement; instead of using the if statement, we can use the ternary operator in JavaScript. The ternary operator assigns a value to the variable based on a condition provided to it.

```
Var a = (condition)? expr1 : expr2;
```

Example

```
<!DOCTYPEhtml>
<head>
<Script>
```

```
let a = 358;
let val = (a % 2 == 0) ? 'Even Number' : 'Odd Number';
alert(val);
</script>
</head>
<body>
<h1>Welcome to the javaTpoint.com </h1>
<h3>This is an example of ternary operator. </h3>
</ body>
</html>
```

JavaScript reload() method

In Java Script, the reload() method is used to reload a webpage. It is similar to the refresh button of the browser. This method does not return any value.

```
location.reload()
```

Example

```
<! DOCTYPE html>
<html>
<head>
<title>
location.reload()method
</ title>
<script>
function fun() {
location.reload();
}
</ script>
</ head>
<body>
```

```
<h1> Welcome to the JavaTpoint.com </h1>
<h2> This an example of location.reload()method</h2>
<p> click the following 'Reload' button to see the effect
                                        </p>
<button onclick = "fun()" > Reload </button>
</body>
</html>
```

JavaScript setAttribute()

The setAttribute() method is used to set or add an attribute to a particular element and provides a value to it. If the attribute already exists, It only set or changes the value of the attribute. So, we can also use the setAttribute() method to update the existing attribute's value. If the corresponding attribute does not exist, It will create a new attribute with the specified name and value. This method does not return any value. The attribute name automatically converts into lowercase when we use it on an HTML.

element.setAttribute(attributeName, attributeValue)

Example

```
<html>
<head>
<title> JavaScript setAttribute()method </title>
<Script>
function fun(){
document.getElementById("link").setAttribute("href", "https:
            //WWW.javaTpoint.com/ ");
}
```

```
</script>
</head>
<body style = "text-align: center;" >
<h2> It is an example of adding an attribute
   using the setAttribute() method. </h2>
<a id = "link" > javaTpoint.com </a>
<p> Click the following button to see the effect. </p>
<button onclick = "fun()" >Add attribute </button>
</body>
</html>
```

JavaScript SetInterval() method

The setInterval() method in JavaScript is used to repeat a specified function at every given time-interval. It evaluates an expression or calls a function at given intervals. This method continues the calling of function until the Window is closed or the clear Interval() method is called. This method is returns a numeric value or a non-zero number that identifies the created timer.

```
Window.setInterval(function, milliseconds);
```

Example

```
<html>
<head>
<title>setInterval() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the sentInterval() method
                                    </h3>
```

```
<p> Here, an alert dialog box displays on every three seconds.
                                        </p>
<script>
var a;
a = setInterval (fun, 3000);
function fun() {
alert ("Welcome to the javaTpoint.com")
} </script>
 </body>
 </html>
```

JavaScript setTimeout() method

The setTimeout() method in JavaScript, is used to execute a function after waiting for the specified time interval. This method returns a numeric value that represents the Id value of the timer.

We can use the clearTimeout() method to stop the timeout or to prevent the execution of the function specified in the setTimeout() method. The value returned by the setTimeout() method can be used as the argument of the clearTimeout() method to cancel the timer.

```
Window.setTimeout (function, milliseconds);
```

Example

```
<html>
<head>
<title> setTimeout() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the setTimeout() method </h3>
```

```
<p>Here, an alert dialog box will display after two
        seconds. </p>
<script>
var a;
a = setTimeout(fun, 2000);
function fun() {
alert("Welcome to the javaTpoint.com");
}
</script>
</body>
</html>
```

JavaScript String includes()

The JavaScript String includes() method is used to determine whether or not the specified substring is present in the given string. It is a case-sensitive method. It returns the Boolean value, either true or false. It returns true if the string contains the specified substring and returns false if not.
It does not change the value of the Original String.

```
String.includes(searchValue, start);
```

Example

```
<DOCTYPE html>
<html>
<head>
</head>
<body>
<h1> Hello World :):)</h1>
<h3> This is an example of using the JavaScript's
        String includes() method </h3>
```

```
<script>
let str = "Welcome to the javaTpoint.com";
document.Write("<b> The given string is: </b>", str);
document.Write ("<br>");
let res = str.includes('tO');
document.write("<b> The result is: </b>", res);
</script>
</body>
</html>
```

JavaScript String trim()

The trim() is a built-in string function in JavaScript, which is used to trim a string. This function removes the whitespace from both the ends; ie, start and end of the string. As the trim() is a string method, so it is invoked by an instance of the string class. We have to create an instance of String class.

```
Str.trim()
```

Example

```
<html>
<body>
<script>
function func_trim() {
    //original string with whitespace in beginning and end
  Var str = " javatpoint tutorial website ";
  //String trimmed using trim()
  Var trimmedstr = str.trim();
  document.write(trimmedstr);
}
func_trim();
```

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

Output
Javatpoint tutorial Website

Javascript Setinterval

Javascript can be made to execute a block of code at specific intervals of time. These intervals are critically defined as time events. There are usually two methods for the same. They can be specifically used according to your requirements. Those two methods are:

1. SetInterval()
2. setTimeout()

Example Code:

```
function Display()
{
  Console.log("Hello JavaTpoint");
}
SentInterval(Display, 2000);
```

Code 2:

```
Var logic = setInterval(Time, 2000);
function Time (){
  Varx = mew Date();
  Vary = X.toLocaleTimeString();
}
function FunctionStop(){
  ClearInterval (logic);
}
```

JavaScript print() method

A print() method is used to print the currently visible contents like a web page, text, image, etc., on the computer screen. When we use a print() method in JavaScript and execute the code, it opens a print dialog box that allows the user or programmer to select an appropriate option for printing the current content of the window.

Example

```
<html>
<head>
<title>
Use print() method in JavaScript
</title>
<script type = "text/javascript">
<!--
//-->
</script>
</head>
<body>
<h2> program to print the Current Content of the
    Window using print() method </h2>
<br > <br>
<p>
As the name suggests, the print() method is used to
print the contents of the current window.
<form>
<!-
-When a user click on the print button, the onclick functio
 Calls the Window.print()method
->
<input type = "button" value = "Print" onclick = "window.print()"/>
```

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

JavaScript TypedArray

The JavaScript TypedArray object illustrates an array like view of an underlying binary data buffer. There are many number of different global properties, whose values are TypedArray constructors for specific element types.

Types of TypedArray

Int8Array

- Size in bytes: 1
- Description: 8-bit two's compliment signed integer.
- Type: byte.
- Value Range: -128 to 127

Unit8Array

- Size In bytes: 1
- Description: 8-bit two's compliment signed octet.
- Type: Octet.
- Value Range: 0 to 255.

Unit8clamped Array

- Size in bytes: 1
- Description: 8-bit unsigned integer(clamped) octet.
- Type: Octet
- Value Range: 0 to 255.

Int16Array

- Size in bytes : 2
- Description : 16-bit two's complement signed integer.
- Type : short
- Value Range : -32768 to 32767.

Unit16Array
- Size in bytes : 2
- Description: 16- bit unsigned integer.
- Type: unsigned short.
- Value Range : 0 to 65535.

Int32Array
- Size in bytes : 4
- Description : 32-bit two's complement signed integer.
- Type : long.
- Value Range :-2147483648 to 2147483647.

Unit32Array
- Size in bytes : 4
- Description: 32-bit unsigned integer.
- Type : unsigned long.
- Value Range: 0 to 4294967295.

Float32Array
- Size in bytes : 4
- Description: 32-bit IEEE floating point number unrestricted float.
- Type : unrestricted float.
- Value Range: 1.2×10^{-38} to 3.4×10^{38}

Float 64Array
- Size in bytes : 8
- Description: 64-bit IEEE floating point number unrestricted double.

- Type: unrestricted double.
- Value Range: 5.4 x 10-324 to 1.8 x 10308

JavaScript Set Object

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references.

```
new Set([iterable])
```

Parameter

interable- It represents an iterable object whose elements will be added to the new set.

Points to remember

- A set object uses the concept of keys internally.
- A set object cannot contain the duplicate values.
- A set object iterates its elements in insertion order.

JavaScript Set Methods

1. add()
   It adds the specified values to the Set Object.

2. Clear()
   It removes all the elements from the Set object.

3. delete()
   It deletes the specified element from the Set Object.

4. entries()

It returns an object of set iterator that contains an array of [value, value] for each element.

5. forEach()
It executes the specified function once for each value.

6. has()
It indicates whether the set object contains the specified value element.

7. Values()
It returns an object of Set iterator that contains the values for each element.

JavaScript Map Object

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key.

new Map ([Iterable])

Parameter

Iterable - It represents an array and other iterable object whose elements are in the form of key-value pair.

Points to remember

- A map object cannot contain the duplicate keys.
- A map object can contain the duplicate values.
- The key and value can be of any type (allows both object and primitive values).
- A map object iterates its elements in insertion order.

JavaScript Map Methods

1. clear()
It removes all the elements from a Map object.

2. delete()
It deletes the specified element from a Map object.

3. entries()
It returns an object of map interator that contains the key-value pair of each element.

4. forEach()
It executes the specified function once for each key/value pair.

5. get()
It returns the value of specified key.

6. has()
It indicates whether the map object contains the specified key element.

7. keys()
It returns an object of Map iterator that contains the keys for each element.

8. set()
It adds or updates the key-value pairs to Map object.

9. values()
It returns an object of map iterator that contains the values for each element.

JavaScript WeakSet Object

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike set, the weakset are the collections of objects only. It doesn't contain the arbitrary values.

```
new WeakSet([iterable])
```

Parameter

iterable - It represents the iterable object whose elements will be added to a new WeakSet.

Points to remember

- A WeakSet object contains unique objects only.
- In WeakSet, if there is no reference to a stored object, they are targeted to garbage collection.
- In WeakSet, the objects are not enumerable. So, It doesn't provide any method to get the specified objects.

JavaScripts WeakSet Methods

1. add()
   It adds a new object to the end of WeakSet object.

2. delete()
   It removes the specified object from the WeakSet object.

3. has()
   It indicates whether the WeakSet object contains the specified object element.

JavaScript Closures

A closure can be defined as a JavaScript feature in which the inner function has access to the outer function variable. In JavaScript every time a closure in created with the creation of a function.
The closure has three scope chains listed as follows:

- Access to its own scope.
- Access to the variables of the outer function.
- Access to the global variables.

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function fun()
{
Var a = 4; //'a' is the local variable, created by the fun()
function innerfun() //the innerfun() is the inner function,
    or a closure
{
return a;
}
return innerfun;
}
Var output = fun();
document.Write(output());
document.Write(" ");
document.Write(output());
</script>
</head>
<body>
```

JavaScript date format

The JavaScript date object can be used to get a year, month and day. We can display a timer on the Web-page with the help of a JavaScript date object. There are many types of date formats in JavaScript: ISO Date, Short Date and Long Date. The format's of JavaScript's date are defined as follows:

ISO date

"2020-08-01" (The International Standard)

Short date

"01/08/2020"

Long date

"Aug 01 2020" or "01 Aug 2020"

ISO date

The ISO 8601 is the international standard for the times and dates, and the syntax (YYYY-MM-DD) of this standard is the preferred date format in JavaScript

Example

```
<!DOCTYPEhtml>
<html>
<head>
</head>
<body>
<div>
<h1>Welcome to the JavaTpoint.com</h1>
<h3>It is an example of JavaScript's ISO date</h3>
<P id = "para"></p>
```

```
</div>
<script>
let val = new Date("2020-08-01");
document.getElementById("para").innerHTML=val;
</script>
</body>
</html>
```

We can Write the ISO dates using the following syntaxes:

1. This is a complete date format using ISO date.
```
let val = new Date("2020-08-01");
```
Sat Aug 01 2020 05:30:00 GMT +0530 (India Standard Time)

2. In this format, we specify only year and month (YYYY-MM) Without day.
```
let val = new Date("2020-08");
```
Sat Aug 01 2020 05:30:00 GMT+0530 (India standard Time)

3. In the third syntax, we only specify the year (YYYY) Without month and day.
```
let val = new Date("2020")
```
Wed Jan 01 2020 05:30:00 GMT+0530 (India standard Time)

4. Now, in the forth syntax, we specify the date With added hours, minutes and seconds. (YYYY-MM-DDTHH:MM:SSZ). In this format, the date and time are separated With the letter 'T' and the letter 'Z'. We get different results in different browsers if We remove these characters.
```
let val = new Date("2020-08-01T07:05:00Z");
```
Sat Aug 01 2020 12:35:00 GMT+0530 (India Standard Time)

JavaScript Short Date

The "MM/DD/YYYY" is the format used to Write short dates. Now we understand the short date by using an example.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div>
<h1> Welcome to the JavaTpoint.com </h1>
<h3> It is an example of JavaScript's Short date </h3>
</div>
<script>
let val = new Date("08/01/2020");
document.Write(val);
</script>
</body>
</html>
```

JavaScript Long Date

The "MMM DD YYYY" is the format used to Write Long dates. The month and date can be written it any order. and it is allowed to Write a month in addreviated (Aug) form or in full (August)

Example

```
<!DOCTYPE html>
<html>
<head>
```

```
</head>
<body>
<div>
<h1> Welcome to the JavaTpoint.com</h1>
<h3> It is an example of JavaScript's Long date</h3>
</div>
<script>
let val= new Date ("Aug 01 2020");
document.Write(val);
</script>
</ body>
</ html>
```

JavaScript date parse() method

The parse() method in JavaScript is used to parse the specified date string and returns the number of milliseconds between the specified date and January 1, 1970. If the string does not have valid values or if it is not recognized, then the month method returns NaN.

The counting of milliseconds between two specified dates helps us to find the number of hours, days, months, etc. by doing easy calculations.

```
date.parse(datestring)
```

Example

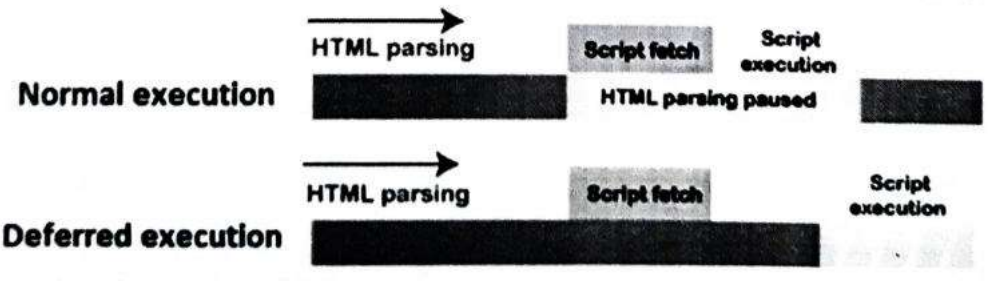
```
<html>
<head>
</head>
<body>
<h1> Hello World :) :) </h1>
```

```
<p>Here, we are finding the number of milliseconds between
the given date and midnight of JavaScript
<script>
Var d1 = "June 19, 2020";
Var m1 = Date.parse(d1);
document.Write("The number of milliseconds between <b>"
  +d1 + "</b> and <b> January 1, 1970 </b> is: <b>" + m1 +
                "</b> ");
</script>
</body>
</html>
```

JavaScript defer

The defer is Boolean value, used to indicate that script is executed after the document has been parsed. It works only with externals scripts.(i.e; works only when we are specifying the src attribute in <script> tag). It declares that the script will not create any content. So the browser can continue the parsing of the rest of the page. The <script> with the defer attribute does not block the page.

We can understand the use of defer attribute in the following image:



This attribute tells the browser to execute the <script> file when the entire HTML document gets fully parsed. sometimes, the application consumes more memory by adding the <script> tag in the HTML head section, and it also causes performance issues. To improve the performance. We can add the defer attribute in the <script> tag.

```
<script defer>
```

Example

```
<!DOCTYPE html>
<html>
<head>
<script src = "myscript.js" defer>
</script>
</head>
<body>
<div>
<div>
<h1> javaTpoint.com </h1>
<h3> This is an example of defer attribute. </h3>
</div>
</body>
</html>
```

myscript.js

```
alert (" Hello World. \n Welcome to the javaTpoint.com\n
 This is an example of the defer attribute.");
```

JavaScript redirect

Redirect is nothing but a mechanism of sending search engines and users on a different URL from the original one. The redirected page can be on the same server or on a different server. It can also be on the same website or on different websites. Sometimes when we clicked on a URL, we directed to another URL. It happen because of the page redirection. It is different from refreshing a page.

```
<link rel = "canonical" href = "https://www.javatpoint.com/" />
```

location.replace()

It is one of the commonly used window.location object. It is used for replacing the original document with a new one.

In this method, we can pass a new URL, and then it will perform an HTTP redirect. It is different from href as it removes the current document from the document's history, so it is not possible to navigate back to the original document.

```
Window.location.replace("new URL");
```

Example

```
<html>
<head>
<Sript type = "text/javascript">
function page_redirect() {
Window.location = "https://www.javatpoint.com/";
}
</Script>
```

```
</head>
<body>
<h2>This is an example of the page redirection</h2>
<p>Click the following button to see the effect.</p>
<form>
<input type="button" value="Redirect" onclick=
"page_redirect()" />
</form>
</body>
</html>
```

JavaScript Scope

A scope can be defined as the region of the execution, a region where the expressions and values can be referenced.
There are two scopes in JavaScript that are globle and local:

Global Scope:
In the global scope, the variable can be accessed from any part of the JavaScript code.

Local scope:
In the local scope, the variable can be accessed within a function where it is declared.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
```

```
<script>
Var $var12=200;
function example(){
   Var $var12 = 300;
   document.Write("inside example() function = " + $ var12);
}
document.Write("Outside example() function =" + $ var12);
document.Write("<br>");
example();
</script>
</body>
</html>
```

JavaScript scroll

The Onscroll event in JavaScript occurs When a scrollbar is used for an element. The event is fired when the user moves the Scrollbar up or down. We can use the CSS overflow property for creating a scrollbar.

In HTML

We can use the onscroll attribute and assign a JavaScript function to it. We can also use the Java Script's addEventListener() method and pass a scroll event to it for greater flexibility.

In HTML

```
<element onscroll = "fun()">
```

In JavaScript

```
Object.Onscroll = function() { myscript };
```

In Javascript by using the addEventListener() method

```
Object.addEventListener("scroll", my script);
```

JavaScript Sleep/Wait

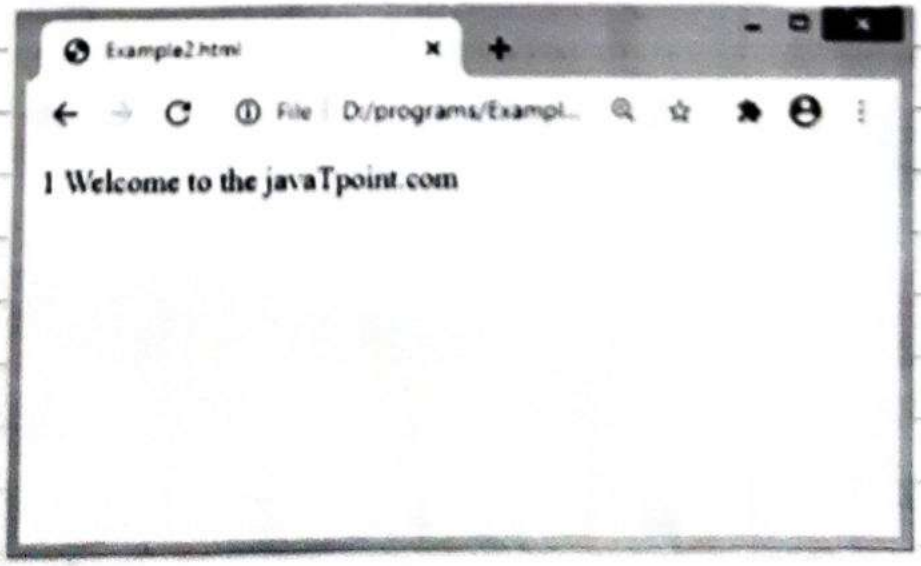
The programming languages such and PHP, C has a sleep(sec) function to pause the execution for a fixed amount of time. Java has a thread.sleep(), phyton has time.sleep(),and Go has time.Sleep (2*time.second)

Unlike other languages, JavaScript doesn't have any sleep() function. We can use some approaches for simulating the sleep() function in JavaScript. The features such as promises and async/await function in Javascript helped us to use the sleep() function in an easier Way. The await is used to wait for a promise and can only be used in an async function. The behavior of JavaScript is asynchronous, so there is a concept of promises to handle such asynchronous behavior of this asynchronous behavior. Its continues Work and does not wait for anything during execution. Async/await functions help us to write the code in a synchronous manner.

Example

```
<html>
<head>
</head>
<body>
<h1>Example of using sleep()in JavaScript </h1>
<Script>
   function sleep (milliseconds){
     return new promise(resolve => setTimeout(resolve,
                                    milliseconds));
   }
  async function fun1() {
```

```
document.Write('Hello World');
for(let i = 1; i <= 10; i++) {
    await sleep(2000);
    document.Write(i + " " + "Welcome to the javaTpoint.
                com" + " " + "</br>");
    }
}
fun();
</script>
</body>
</html>
```



JavaScript: void(0)

The Void operator is used to evaluate an expression and returns the undefined. Generally, this operator is used for obtaining the undefined primitive value. It is often used with hyperlinks. Usually the browser refreshes the page or loads a new page on clicking a link. The javascript: void(0) can be used when we don't want to refresh or load a new page in the browser on clicking a hyperlink.

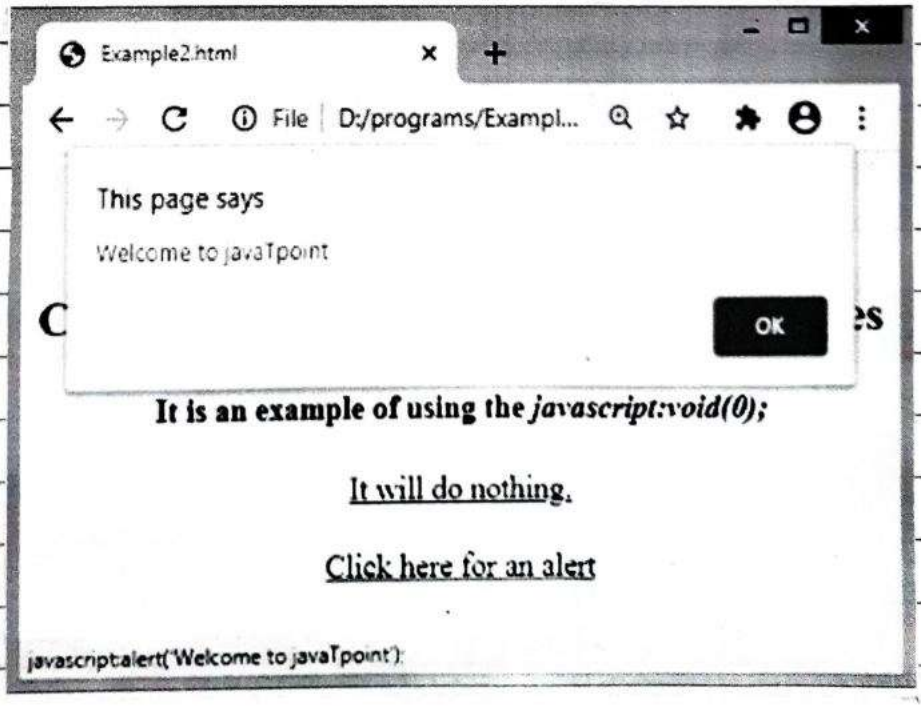
We can use the operand 0 in two ways that are void(0) or void 0. Both of the ways work the same. The JavaScript: void(0) tells the browser to "do nothing" i.e., prevents the browser from reloading or refreshing the page. It is useful when we insert links that have some important role on the webpage without any reloading. So, using void(0) on such links prevents the reloading of the page but allows to perform a useful function such as updating a value on the webpage.

It is also used to prevent unwanted redirecting of of the page.

Example

```
<html>
<head>
</head>
<body>
<center>
<h1> Hello World :):)</h1>
<h2> Click the following links to see the changes </h2>
<h4> It is an example of using the <i> javascript:
        Void(0); </i> </h4>
```

```
<a href = "JavaScript:void(0);">
It Will do nothing
</a>
<br/><br/>
<a href = "Javascript:alert('Welcome to the JavaTpoint');">
        Click here for an alert </a>
</center>
</body>
</html>
```



JavaScript Form

- Form name tag is used to define the name of the form. The name of the form here is "Login_form". This name will be referenced in the JavaScript form.
- The action tag defines the action, and the browser will take to tackle the form when it is submitted. Here, We have taken no action.
- The method to take action can be either post or get. Which is used when the form is to be submitted to the server. Both types of methods have their own properties and rules.
- The input type tag defines the type of inputs we went to create in our form. Here We have used input type as 'text'. Which means We will input values as text in the textbox.
- Net, We have taken input type as 'password' and the input values as text in the textbox.
- Next, We have taken input type as 'button' where on clicking, we get the value of the form and get displayed.

Other than action and methods, there are the following usful methods also which are provided by the HTML Form Element

- **Submit()**:
  The method is used to submit the form.

- **reset()**:
  The method is used to reset the form values.

Referencing forms

Now, We have created the form element using HTML, but We also need to make its connectivity to

JavaScript. For this, we use the getElementById() method that references the html form element to the JavaScript code.

```
let form = document.getElementById('subscribe')
```

Submitting the form

Next, We need to submit the form by submitting its value. for which we use the onSubmit() method. Generally to submit, We use a submit button that submits the value entered in the form.
The syntax of the submit() method is as follows:

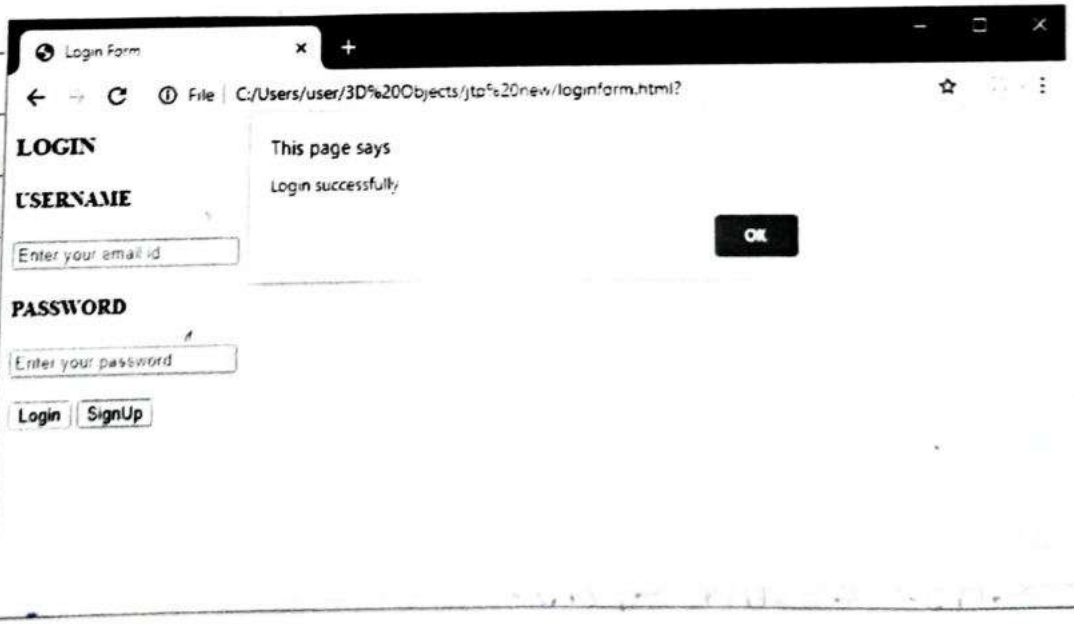
```
<input type = "submit" value = "Subscribe">
```

When We submit the form, the action is taken just before the request is sent to the server. It allows us to add an event listner that enables us to place various validations on the form. Finally, the form gets ready with a combination of HTML and JavaScript code.
Let's collect and use all these to create a Login form and signup form and use both.

Login Form

```
<html>
<head>
<title>Login Form</title>
</head>
<body>
<h3> LOGIN </h3>
<formform = "Login_form" Onsubmit = "Submit_form()">
```

```
<h4> USERNAME </h4>
<input type = "text" placeholder = "Enter your email id" />
<h4> PASSWORD </h4>
<input type = "password" placeholder = "Enter your password"
    /> </br></br>
<input type = "submit" value = "Login" />
<input type = "button" value = 'signup" onclick =
    "Create" />
</form>
<script type = "text/javascript">
function submit_form(){
alert("Login Successfully");
}
function Create(){
Window.location = "Signup.html";
}
</script>
</body>
</html>
```



SignUp Form

```
<html>
<head>
<title> SignUp Page</title>
</head>
<body align = "center">
<h1> CREATE YOUR ACCOUNT </h1>
< table cellspacing = "2" align = "center" cellpadding = "8" border = "0">
<tr > < td > Name </td >
<td ><input type = "text" placeholder = "Enter your name" id =
        "n1"></td></tr>
<tr><td> Email </td>
<td><input type = "text" placeholder = "Enter your email id"
        id = "e1"></td></tr>
<tr><td> Set Password </td>
<td><input type = "password" placeholder = "Confirm your
        password" id = "p2" > </td></tr>
<tr><td>
<input type = "Submit" value = "Create" onclick = Create_account
        ()"/>
</table>
<script type = "text/javascript" >
function create_account() {
Var n = document. getElement Byld ("n1"). Value;
Vare = document. getElement Byld ("e1"). value;
Varp = document. getElement Byld ("p1") Value;
Varcp = document. get ElementByld ("p2") Value;
// code for password validation
      var letters = /^(A-ZA-z)+$/;
      var email_val = /^([a-zA-Z0-9_.|-])+|@(([a-zA-
          Z0-9\-])+\.)+ ([a-zA-Z0-9]{2,4})+$/;
// other validations required code
```

```
if (n=="||e=="||p=="||cp=="){
alert("Enter each details correctly");
}
 else if (!letters.tests(n))
        {
         alert('Name is incorrect must contain alaphbets only');
        }
 elseif(!emailval.test(e))
        {
         alert('Invalid email format please enter Valid email id');
        }
 elseif(p!=cp)
 {
 alert("Passwords not matching");
 }
 else if(document.getElementById("p1").value length>12)
 {
 alert("Password maximum length is 12");
 }
 else if(document.getElementById("p1").value.length<6)
 {
 alert("Password minimum length is 6");
 }
 else{
 alert("Your account has been created sucessfully...
        Redirecting to JavaTpoint.com");
 window.location = "https://www.javatpoint.com/";
}
}

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

Adding JavaScript to HTML Pages

There are following three ways in which users can add JavaScript to HTML pages.

1. Embedding Code
2. Inline Code
3. External Code

1. Embedding code

Example

```
<!DOCTYPE html>
<html>
 <head>
 <title>page title</title>
 <script>
 document.Write("Welcome to Javatpoint");
 </script>
  </head>
  <body>
   <p>In this example we saw how to add Javascript in the
              head section</p>
    </body>
    </html>
```

2. Inline Code

Example

```
<!DOCTYPE html>
<html>
<title> page title</title>
</head>
<body>
```

```
<p>
<a href="#" onclick="alert('Welcome!')">clickme</a>
</p>
<p> in this example We saw how to use inline JavaScript
    Or directly in an HTML tag. </p>
</body>
</html>
```

3. External file

```
<html>
<head>
<meta Charset = "utf-8">
<title>including a External JavaScript File</title>
</head>
<body>
 <form>
 <input type="button" value = "Result" onclick = "display()"/>
 </form>
  <script stc = "hello.js" >
 </script>
  </body>
  </html>
```

What is hoisting in JavaScript?

In JavaScript, Hoisting is a kind of default behavior in which all the declarations either variable declaration or function declaration are moved at the top of the scope just before executing the program's code. However, it can be considered an advantage because all funtions and variable declarations are placed to top of their scope no matter where they are all declared anywhere in the whole program.

even regardless of whether they are declared global or local.
Due to the concept of hoisting in Javascript, we can call a function even before we define the function definition in our program's code.
In simple words. We can say that we can use the variables and functions in Javascript, before declaring them because as we disscussed above Javascript compiler moves the declarations of all the variables and functions at the top of their scope so that there will not be an error of any kind. The concept of Javascript of moving all declarations of the variable and functions to the top of their scop by compiler itself just before the execution of code is known as Hoisting.

What is the difference between Java and Javascript

	Java Language	Javascript Language
1.	It is a programming language.	It is a scripting language.
2.	Java is a pure Object Oriented programming Language.	Java Script is object-based Language.
3.	Java is a standalone language.	Javascript is not a standalone language, as it needs to be integrated into an HTML program for execution.

4.	Java is a strongly typed language. Which means that the user has to decided the data type of the variable before declaring and using it. Example "int a", the variable "a" can store the value of integer type only.	JavaScript is a loosely typed language. Which means that the user does not have to worry about the data-type of the variable before and after the declaration. Example "var a"; the "a" variable can store the value of any data-type.
5.	Java program should be compiled before execution.	JavaScript needs to be integrated into the HTML program for the execution.
6.	The Web-browser is not required to run java programs.	The Web-browser is essential to run the JavaScript programs.
7.	It is one of the complex languages to learn.	It is one of the easy languages to learn.
8.	In Java, by utilizing the multithreading, users can perform complicated tasks.	In JavaScript, user is not able to perform complicated tasks.
9.	It requires a large amount of memory.	It does not require that amount of memory.
10.	Java programming language was developed by the "Sun Microsystems."	JavaScript programming language was developed by the "Netscape".

11.	In Java programming language. programs are saved with the ".java" exentsion.	On the other hand, programs in JavaScript are saved with the ".js" extension.
12	Java is stored on the host machine as the "Byte" Code.	JavaScript is stored on the Host machine (client machine) as the "source" text.