What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

## Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

## History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

* Client-side validation,
* Dynamic drop-down menus,
* Displaying date and time,
* Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
* Displaying clocks etc.

### **JavaScript Example**

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let’s create the first JavaScript example.

**<script>**

document.write("Hello JavaScript by JavaScript");

**</script>**

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. Outside of body and head tag and inside html element.
4. In .js file (external javaScript)

## 1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let’s see the simple example of JavaScript that displays alert dialog box.

<html>

<body>

<script type="text/javascript">

alert("Hello iHub");

</script>

</body>

</html>

## 2) JavaScript Example : code between the head tag

Let’s see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function\_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

<html>

<head>

<script type="text/javascript">

function msg(){

alert("Hello iHub");

}

</script>

</head>

<body>

<p>Welcome to Javascript</p>

<form>

<input type="button" value="click" onclick="msg()"/>

</form>

</body>

</html>

# **External JavaScript file**

We can create external JavaScript file and embed it in many html page.

It provides **code re-usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external Javascriptfile that prints Hello iHub in an alert dialog box.

**message.js**

function msg(){

 alert("Hello iHub");  }

Let's include the JavaScript file into [html](https://www.javatpoint.com/html-tutorial) page. It calls the [JavaScript function](https://www.javatpoint.com/javascript-function) on button click.

**index.html**

**<html>**

**<head>**

**<script** type="text/javascript" src="message.js"**></script>**

**</head>**

**<body>**

**<p>**Welcome to JavaScript**</p>**

**<form>**

**<input** type="button" value="click" onclick="msg()"**/>**

**</form>**

**</body>**

**</html>**

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

# **JavaScript Comment**

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

#### **Advantages of JavaScript comments**

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

## JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let’s see the example of single-line comment i.e. added before the statement.

**<script>**

// It is single line comment

document.write("hello javascript");

**</script>**

Let’s see the example of single-line comment i.e. added after the statement.

**<script>**

var a=10;

var b=20;

var c=a+b;//It adds values of a and b variable

document.write(c);//prints sum of 10 and 20

**</script>**

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

/\* your code here  \*/

It can be used before, after and middle of the statement.

**<script>**

/\* It is multi line comment.

It will not be displayed \*/

document.write("example of javascript multiline comment");

**</script>**

# **JavaScript Variable**

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

var x = 10;

var \_value="sonoo";

Incorrect JavaScript variables

var  123=30;

var \*aa=320;

## Example of JavaScript variable

Let’s see a simple example of JavaScript variable.

<html>

<body>

<script>

var x = 10;

var y = 20;

var z=x+y;

document.write(z);

</script>

</body></html>

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

**<script>**

function abc(){

var x=10;//local variable

}

**</script>**

Or,

**<script>**

If(10**<13**){

var y=20;//JavaScript local variable

}

**</script>**

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

**<script>**

var data=200;//gloabal variable

function a(){

document.writeln(data);

}

function b(){

document.writeln(data);

}

a();//calling JavaScript function

b();

**</script>**

#### **Declaring JavaScript global variable within function**

To declare JavaScript global variables inside function, you need to use **window object**. For example:

window.value=90;

Now it can be declared inside any function and can be accessed from any function. For example:

function m(){

window.value=100;//declaring global variable by window object

}

function n(){

alert(window.value);//accessing global variable from other function

}

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

var value=50;

function a(){

alert(window.value);//accessing global variable

}

# **Javascript Data Types**

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

var a=40;//holding number

var b="Rahul";//holding string

## JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

## JavaScript non-primitive data types

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

# **JavaScript Operators**

JavaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

## JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | 10+20 = 30 |
| - | Subtraction | 20-10 = 10 |
| \* | Multiplication | 10\*20 = 200 |
| / | Division | 20/10 = 2 |
| % | Modulus (Remainder) | 20%10 = 0 |
| ++ | Increment | var a=10; a++; Now a = 11 |
| -- | Decrement | var a=10; a--; Now a = 9 |

## JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Is equal to | 10==20 = false |
| === | Identical (equal and of same type) | 10==20 = false |
| != | Not equal to | 10!=20 = true |
| !== | Not Identical | 20!==20 = false |
| > | Greater than | 20>10 = true |
| >= | Greater than or equal to | 20>=10 = true |
| < | Less than | 20<10 = false |
| <= | Less than or equal to | 20<=10 = false |

## JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Bitwise AND | (10==20 & 20==33) = false |
| | | Bitwise OR | (10==20 | 20==33) = false |
| ^ | Bitwise XOR | (10==20 ^ 20==33) = false |
| ~ | Bitwise NOT | (~10) = -10 |
| << | Bitwise Left Shift | (10<<2) = 40 |
| >> | Bitwise Right Shift | (10>>2) = 2 |
| >>> | Bitwise Right Shift with Zero | (10>>>2) = 2 |

## JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (10==20 && 20==33) = false |
| || | Logical OR | (10==20 || 20==33) = false |
| ! | Logical Not | !(10==20) = true |

## JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assign | 10+10 = 20 |
| += | Add and assign | var a=10; a+=20; Now a = 30 |
| -= | Subtract and assign | var a=20; a-=10; Now a = 10 |
| \*= | Multiply and assign | var a=10; a\*=20; Now a = 200 |
| /= | Divide and assign | var a=10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a=10; a%=2; Now a = 0 |

## JavaScript Special Operators

The following operators are known as JavaScript special operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | Comma Operator allows multiple expressions to be evaluated as single statement. |
| delete | Delete Operator deletes a property from the object. |
| in | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| new | creates an instance (object) |
| typeof | checks the type of object. |
| void | it discards the expression's return value. |
| yield | checks what is returned in a generator by the generator's iterator. |

# **JavaScript If-else**

The **JavaScript if-else statement** is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

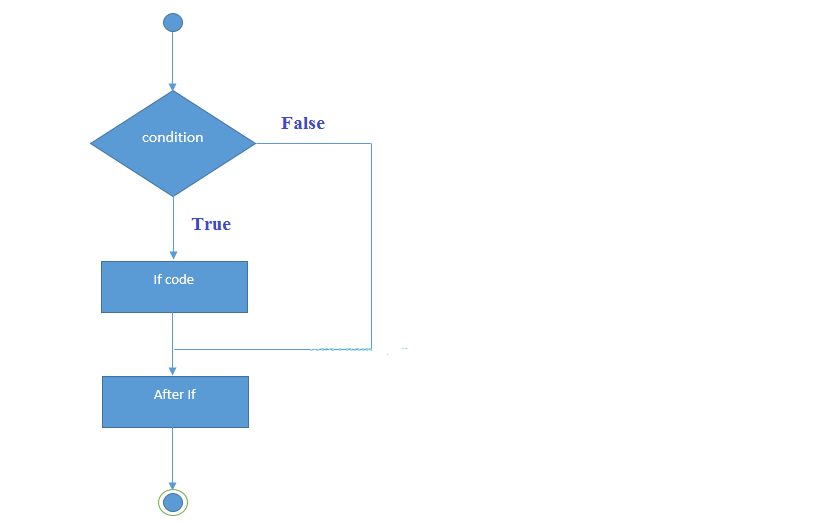
1. If Statement
2. If else statement
3. if else if statement

### **JavaScript If statement**

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

if(expression){  //content to be evaluated  }

### **Flowchart of JavaScript If statement**



Let’s see the simple example of if statement in javascript.

<html>

<body>

<script>

var a=20;

if(a>10){

document.write("value of a is greater than 10");

}

</script>

</body>

</html>

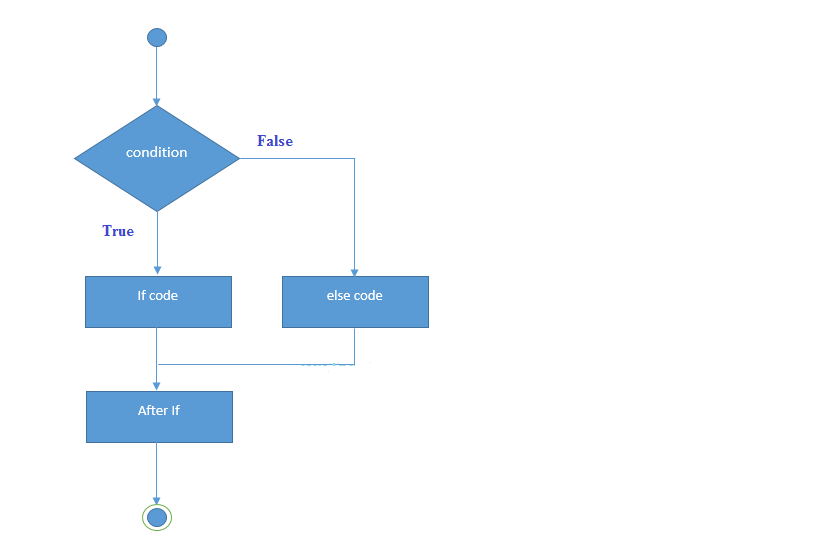
### **JavaScript If...else Statement**

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

if(expression){ //content to be evaluated if condition is true

else{  //content to be evaluated if condition is false  }

### **Flowchart of JavaScript If...else statement**



Let’s see the example of if-else statement in JavaScript to find out the even or odd number.

<html>

<body>

<script>

var a=20;

if(a%2==0){

document.write("a is even number");

}

else{

document.write("a is odd number");

}

</script>

</body>

</html>

### **JavaScript If...else if statement**

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

if(expression1){

//content to be evaluated if expression1 is true

}

else if(expression2){

//content to be evaluated if expression2 is true

}

else if(expression3){

//content to be evaluated if expression3 is true

}

else{

//content to be evaluated if no expression is true

}

Let’s see the simple example of if else if statement in javascript.

<html>

<body>

<script>

var a=20;

if(a==10){

document.write("a is equal to 10");

}

else if(a==15){

document.write("a is equal to 15");

}

else if(a==20){

document.write("a is equal to 20");

}

else{

document.write("a is not equal to 10, 15 or 20");

}

</script>

</body>

</html>

# **JavaScript Switch**

The **JavaScript switch statement** is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

switch(expression){

case value1:

  code to be executed;

break;

case value2:

  code to be executed;

break;

......

default:

  code to be executed if above values are not matched;

}

Let’s see the simple example of switch statement in javascript.

<!DOCTYPE html>

<html>

<body>

<script>

var grade='B';

var result;

switch(grade){

case 'A':

result="A Grade";

break;

case 'B':

result="B Grade";

break;

case 'C':

result="C Grade";

break;

default:

result="No Grade";

}

document.write(result);

</script>

</body>

</html>

#### **The switch statement is fall-through i.e. all the cases will be evaluated if you don't use break statement.**

Let’s understand the behaviour of switch statement in JavaScript.

<!DOCTYPE html>

<html>

<body>

<script>

var grade='B';

var result;

switch(grade){

case 'A':

result+=" A Grade";

case 'B':

result+=" B Grade";

case 'C':

result+=" C Grade";

default:

result+=" No Grade";

}

document.write(result);

</script>

</body>

</html>

# **JavaScript Loops**

The **JavaScript loops** are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

for (initialization; condition; increment)

{

    code to be executed

}

Let’s see the simple example of for loop in javascript.

<!DOCTYPE html>

<html>

<body>

<script>

for (i=1; i<=5; i++) {

document.write(i + "<br/>")

}

</script>

</body>

</html>

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

while (condition)

{

    code to be executed

}

Let’s see the simple example of while loop in javascript.

<!DOCTYPE html>

<html>

<body>

<script>

var i=11;

while (i<=15) {

document.write(i + "<br/>");

i++;

}

</script>

</body>

</html>

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

do{

    code to be executed

}while (condition);

Let’s see the simple example of do while loop in javascript.

<!DOCTYPE html>

<html>

<body>

<script>

var i=21;

do{

document.write(i + "<br/>");

i++;

}while (i<=25);

</script>

</body>

</html>

## 4) JavaScript for in loop

The **JavaScript for in loop** is used to iterate the properties of an object.

### **Syntax**

for (key in object) {  
   // code block to be executed  
}

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For In Loop</h2>

<p>The for in statement loops through the properties of an object:</p>

<p id="demo"></p>

<script>

const person = {fname:"John", lname:"Doe", age:25};

let txt = "";

for (let x in person) {

txt += person[x] + " ";

}

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

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

# **JavaScript Functions**

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

#### **Advantage of JavaScript function**

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

function functionName([arg1, arg2, ...argN]){

  //code to be executed

}

JavaScript Functions can have 0 or more arguments.

## JavaScript Function Example

Let’s see the simple example of function in JavaScript that does not has arguments.

<html>

<body>

<script>

function msg(){

alert("hello! this is message");

}

</script>

<input type="button" onclick="msg()" value="call function"/>

</body>

</html>

## JavaScript Function Arguments

We can call function by passing arguments. Let’s see the example of function that has one argument.

<html>

<body>

<script>

function getcube(number){

alert(number\*number\*number);

}

</script>

<form>

<input type="button" value="click" onclick="getcube(4)"/>

</form>

</body>

</html>

## Function with Return Value

We can call function that returns a value and use it in our program. Let’s see the example of function that returns value.

<html>

<body>

<script>

function getInfo(){

return "hello iHub! How r u?";

}

</script>

<script>

document.write(getInfo());

</script>

</body>

</html>

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

new Function ([arg1[, arg2[, ....argn]],] functionBody)

Parameter

**arg1, arg2, .... ,argn** - It represents the argument used by function.

**functionBody** - It represents the function definition.

JavaScript Function Methods

Let's see function methods with description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [apply()](https://www.javatpoint.com/javascript-function-apply-method) | It is used to call a function contains this value and a single array of arguments. |
| [bind()](https://www.javatpoint.com/javascript-function-bind-method) | It is used to create a new function. |
| [call()](https://www.javatpoint.com/javascript-function-call-method) | It is used to call a function contains this value and an argument list. |
| [toString()](https://www.javatpoint.com/javascript-function-tostring-method) | It returns the result in a form of a string. |

## JavaScript Function Object Examples

### **Example 1**

Let's see an example to display the sum of given numbers.

<!DOCTYPE html>

<html>

<body>

<script>

var add=new Function("num1","num2","return num1+num2");

document.writeln(add(2,5));

</script>

</body>

</html>

### **Example 2**

Let's see an example to display the power of provided value.

<!DOCTYPE html>

<html>

<body>

<script>

var pow=new Function("num1","num2","return Math.pow(num1,num2)");

document.writeln(pow(2,3));

</script>

</body>

</html>

# **JavaScript Objects**

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

<html>

<body>

<script>

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

</body>

</html>

2) By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

<html>

<body>

<script>

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

</body>

</html>

## 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

</script>

</body>

</html>

## Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);

document.write("<br>"+e.id+" "+e.name+" "+e.salary);

</script>

</body>

</html>

## JavaScript Object Methods

The various methods of Object are as follows:

|  |  |  |
| --- | --- | --- |
| **S.No** | **Methods** | **Description** |
| 1 | [Object.assign()](https://www.javatpoint.com/javascript-object-assign-method) | This method is used to copy enumerable and own properties from a source object to a target object |
| 2 | [Object.create()](https://www.javatpoint.com/javascript-object-create-method) | This method is used to create a new object with the specified prototype object and properties. |
| 3 | [Object.defineProperty()](https://www.javatpoint.com/javascript-object-defineproperty-method) | This method is used to describe some behavioral attributes of the property. |
| 4 | [Object.defineProperties()](https://www.javatpoint.com/javascript-object-defineproperties-method) | This method is used to create or configure multiple object properties. |
| 5 | [Object.entries()](https://www.javatpoint.com/javascript-object-entries-method) | This method returns an array with arrays of the key, value pairs. |
| 6 | [Object.freeze()](https://www.javatpoint.com/javascript-object-freeze-method) | This method prevents existing properties from being removed. |
| 7 | [Object.getOwnPropertyDescriptor()](https://www.javatpoint.com/javascript-object-getownpropertydescriptor-method) | This method returns a property descriptor for the specified property of the specified object. |
| 8 | [Object.getOwnPropertyDescriptors()](https://www.javatpoint.com/javascript-object-getownpropertydescriptors-method) | This method returns all own property descriptors of a given object. |
| 9 | [Object.getOwnPropertyNames()](https://www.javatpoint.com/javascript-object-getownpropertynames-method) | This method returns an array of all properties (enumerable or not) found. |
| 10 | [Object.getOwnPropertySymbols()](https://www.javatpoint.com/javascript-object-getownpropertysymbols-method) | This method returns an array of all own symbol key properties. |
| 11 | [Object.getPrototypeOf()](https://www.javatpoint.com/javascript-object-getprototypeof-method) | This method returns the prototype of the specified object. |
| 12 | [Object.is()](https://www.javatpoint.com/javascript-object-is-method) | This method determines whether two values are the same value. |
| 13 | [Object.isExtensible()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is extensible |
| 14 | [Object.isFrozen()](https://www.javatpoint.com/javascript-objects) | This method determines if an object was frozen. |
| 15 | [Object.isSealed()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is sealed. |
| 16 | [Object.keys()](https://www.javatpoint.com/javascript-objects) | This method returns an array of a given object's own property names. |
| 17 | [Object.preventExtensions()](https://www.javatpoint.com/javascript-object-preventextensions-method) | This method is used to prevent any extensions of an object. |
| 18 | [Object.seal()](https://www.javatpoint.com/javascript-object-seal-method) | This method prevents new properties from being added and marks all existing properties as non-configurable. |
| 19 | [Object.setPrototypeOf()](https://www.javatpoint.com/javascript-object-setprototypeof-method) | This method sets the prototype of a specified object to another object. |
| 20 | [Object.values()](https://www.javatpoint.com/javascript-object-values-method) | This method returns an array of values. |

# **JavaScript Array**

**JavaScript array** is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

var arrayname=[value1,value2.....,valueN];

As you can see, values are contained inside [ ] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

<html>

<body>

<script>

var emp=["Sonoo","Vimal","Ratan"];

for (i=0;i<emp.length;i++){

document.write(emp[i] + "<br/>");

}

</script>

</body>

</html>

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

var arrayname=new Array();

Here, **new keyword** is used to create instance of array.

Let's see the example of creating array directly.

<html>

<body>

<script>

var i;

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

for (i=0;i<emp.length;i++){

document.write(emp[i] + "<br>");

}

</script>

</body>

</html>

## 3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

<html>

<body>

<script>

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i<emp.length;i++){

document.write(emp[i] + "<br>");

}

</script>

</body>

</html>

## JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [concat()](https://www.javatpoint.com/javascript-array-concat-method) | It returns a new array object that contains two or more merged arrays. |
| [copywithin()](https://www.javatpoint.com/javascript-array-copywithin-method) | It copies the part of the given array with its own elements and returns the modified array. |
| [entries()](https://www.javatpoint.com/javascript-array-entries-method) | It creates an iterator object and a loop that iterates over each key/value pair. |
| [every()](https://www.javatpoint.com/javascript-array-every-method) | It determines whether all the elements of an array are satisfying the provided function conditions. |
| [flat()](https://www.javatpoint.com/javascript-array-flat-method) | It creates a new array carrying sub-array elements concatenated recursively till the specified depth. |
| [flatMap()](https://www.javatpoint.com/javascript-array-flatmap-method) | It maps all array elements via mapping function, then flattens the result into a new array. |
| [fill()](https://www.javatpoint.com/javascript-array-fill-method) | It fills elements into an array with static values. |
| [from()](https://www.javatpoint.com/javascript-array-from-method) | It creates a new array carrying the exact copy of another array element. |
| [filter()](https://www.javatpoint.com/javascript-array-filter-method) | It returns the new array containing the elements that pass the provided function conditions. |
| [find()](https://www.javatpoint.com/javascript-array-find-method) | It returns the value of the first element in the given array that satisfies the specified condition. |
| [findIndex()](https://www.javatpoint.com/javascript-array-findindex-method) | It returns the index value of the first element in the given array that satisfies the specified condition. |
| [forEach()](https://www.javatpoint.com/javascript-array-foreach-method) | It invokes the provided function once for each element of an array. |
| [includes()](https://www.javatpoint.com/javascript-array-includes-method) | It checks whether the given array contains the specified element. |
| [indexOf()](https://www.javatpoint.com/javascript-array-indexof-method) | It searches the specified element in the given array and returns the index of the first match. |
| [isArray()](https://www.javatpoint.com/javascript-array-isarray-method) | It tests if the passed value ia an array. |
| [join()](https://www.javatpoint.com/javascript-array-join-method) | It joins the elements of an array as a string. |
| [keys()](https://www.javatpoint.com/javascript-array-keys-method) | It creates an iterator object that contains only the keys of the array, then loops through these keys. |
| [lastIndexOf()](https://www.javatpoint.com/javascript-array-lastindexof-method) | It searches the specified element in the given array and returns the index of the last match. |
| [map()](https://www.javatpoint.com/javascript-array-map-method) | It calls the specified function for every array element and returns the new array |
| [of()](https://www.javatpoint.com/javascript-array-of-method) | It creates a new array from a variable number of arguments, holding any type of argument. |
| [pop()](https://www.javatpoint.com/javascript-array-pop-method) | It removes and returns the last element of an array. |
| [push()](https://www.javatpoint.com/javascript-array-push-method) | It adds one or more elements to the end of an array. |
| [reverse()](https://www.javatpoint.com/javascript-array-reverse-method) | It reverses the elements of given array. |
| [reduce(function, initial)](https://www.javatpoint.com/javascript-array-reduce-method) | It executes a provided function for each value from left to right and reduces the array to a single value. |
| [reduceRight()](https://www.javatpoint.com/javascript-array-reduceright-method) | It executes a provided function for each value from right to left and reduces the array to a single value. |
| [some()](https://www.javatpoint.com/javascript-array-some-method) | It determines if any element of the array passes the test of the implemented function. |
| [shift()](https://www.javatpoint.com/javascript-array-shift-method) | It removes and returns the first element of an array. |
| [slice()](https://www.javatpoint.com/javascript-array-slice-method) | It returns a new array containing the copy of the part of the given array. |
| [sort()](https://www.javatpoint.com/javascript-array-sort-method) | It returns the element of the given array in a sorted order. |
| [splice()](https://www.javatpoint.com/javascript-array-splice-method) | It add/remove elements to/from the given array. |
| [toLocaleString()](https://www.javatpoint.com/javascript-array-tolocalestring-method) | It returns a string containing all the elements of a specified array. |
| [toString()](https://www.javatpoint.com/javascript-array-tostring-method) | It converts the elements of a specified array into string form, without affecting the original array. |
| [unshift()](https://www.javatpoint.com/javascript-array-unshift-method) | It adds one or more elements in the beginning of the given array. |
| [values()](https://www.javatpoint.com/javascript-array-values-method) | It creates a new iterator object carrying values for each index in the array. |

# **JavaScript String**

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

## 1) By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given below:

var stringname="string value";

Let's see the simple example of creating string literal.

<!DOCTYPE html>

<html>

<body>

<script>

var str="This is string literal";

document.write(str);

</script>

</body>

</html>

2) By string object (using new keyword)

The syntax of creating string object using new keyword is given below:

var stringname=new String("string literal");

Here, **new keyword** is used to create instance of string.

Let's see the example of creating string in JavaScript by new keyword.

<!DOCTYPE html>

<html>

<body>

<script>

var stringname=new String("hello javascript string");

document.write(stringname);

</script>

</body>

</html>

## JavaScript String Methods

Let's see the list of JavaScript string methods with examples.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [charAt()](https://www.javatpoint.com/javascript-string-charat-method) | It provides the char value present at the specified index. |
| [charCodeAt()](https://www.javatpoint.com/javascript-string-charcodeat-method) | It provides the Unicode value of a character present at the specified index. |
| [concat()](https://www.javatpoint.com/javascript-string-concat-method) | It provides a combination of two or more strings. |
| [indexOf()](https://www.javatpoint.com/javascript-string-indexof-method) | It provides the position of a char value present in the given string. |
| [lastIndexOf()](https://www.javatpoint.com/javascript-string-lastindexof-method) | It provides the position of a char value present in the given string by searching a character from the last position. |
| [search()](https://www.javatpoint.com/javascript-string-search-method) | It searches a specified regular expression in a given string and returns its position if a match occurs. |
| [match()](https://www.javatpoint.com/javascript-string-match-method) | It searches a specified regular expression in a given string and returns that regular expression if a match occurs. |
| [replace()](https://www.javatpoint.com/javascript-string-replace-method) | It replaces a given string with the specified replacement. |
| [substr()](https://www.javatpoint.com/javascript-string-substr-method) | It is used to fetch the part of the given string on the basis of the specified starting position and length. |
| [substring()](https://www.javatpoint.com/javascript-string-substring-method) | It is used to fetch the part of the given string on the basis of the specified index. |
| [slice()](https://www.javatpoint.com/javascript-string-slice-method) | It is used to fetch the part of the given string. It allows us to assign positive as well negative index. |
| [toLowerCase()](https://www.javatpoint.com/javascript-string-tolowercase-method) | It converts the given string into lowercase letter. |
| [toLocaleLowerCase()](https://www.javatpoint.com/javascript-string-tolocalelowercase-method) | It converts the given string into lowercase letter on the basis of host?s current locale. |
| [toUpperCase()](https://www.javatpoint.com/javascript-string-touppercase-method) | It converts the given string into uppercase letter. |
| [toLocaleUpperCase()](https://www.javatpoint.com/javascript-string-tolocaleuppercase-method) | It converts the given string into uppercase letter on the basis of host?s current locale. |
| [toString()](https://www.javatpoint.com/javascript-string-tostring-method) | It provides a string representing the particular object. |
| [valueOf()](https://www.javatpoint.com/javascript-string-valueof-method) | It provides the primitive value of string object. |
| split() | It splits a string into substring array, then returns that newly created array. |
| trim() | It trims the white space from the left and right side of the string. |

## 1) JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

<!DOCTYPE html>

<html>

<body>

<script>

var str="javascript";

document.write(str.charAt(2));

</script>

</body></html>

## 2) JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="javascript ";

var s2="concat example";

var s3=s1+s2;

document.write(s3);

</script>

</body>

</html>

## 3) JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="javascript from iHubTalentindexof";

var n=s1.indexOf("from");

document.write(n);

</script>

</body>

</html>

## 4) JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="javascript from iHubTalentindexof";

var n=s1.lastIndexOf("java");

document.write(n);

</script>

</body>

</html>

## 5) JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="JavaScript toLowerCase Example";

var s2=s1.toLowerCase();

document.write(s2);

</script>

</body>

</html>

## 6) JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="JavaScript toUpperCase Example";

var s2=s1.toUpperCase();

document.write(s2);

</script>

</body>

</html>

## 7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

<!DOCTYPE html>

<html>

<body>

<script>

var s1="abcdefgh";

var s2=s1.slice(2,5);

document.write(s2);

</script>

</body>

</html>

## 8) JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

<!DOCTYPE html>

<html>

<body>

<script>

var s1=" javascript trim ";

var s2=s1.trim();

document.write(s2);

</script>

</body>

</html>

### **9) JavaScript String split() Method**

**<script>**

var str="This is iHubTalent website";

document.write(str.split(" ")); //splits the given string.

**</script>**

# **JavaScript Date Object**

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

## Constructor

You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

## JavaScript Date Methods

Let's see the list of JavaScript date methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [getDate()](https://www.javatpoint.com/javascript-date-getdate-method) | It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time. |
| [getDay()](https://www.javatpoint.com/javascript-date-getday-method) | It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time. |
| [getFullYears()](https://www.javatpoint.com/javascript-date-getutcfullyear-method) | It returns the integer value that represents the year on the basis of local time. |
| [getHours()](https://www.javatpoint.com/javascript-date-gethours-method) | It returns the integer value between 0 and 23 that represents the hours on the basis of local time. |
| [getMilliseconds()](https://www.javatpoint.com/javascript-date-getmilliseconds-method) | It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time. |
| [getMinutes()](https://www.javatpoint.com/javascript-date-getminutes-method) | It returns the integer value between 0 and 59 that represents the minutes on the basis of local time. |
| [getMonth()](https://www.javatpoint.com/javascript-date-getmonth-method) | It returns the integer value between 0 and 11 that represents the month on the basis of local time. |
| [getSeconds()](https://www.javatpoint.com/javascript-date-getseconds-method) | It returns the integer value between 0 and 60 that represents the seconds on the basis of local time. |
| [getUTCDate()](https://www.javatpoint.com/javascript-date-getutcdate-method) | It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time. |
| [getUTCDay()](https://www.javatpoint.com/javascript-date-getutcday-method) | It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time. |
| [getUTCFullYears()](https://www.javatpoint.com/javascript-date-getutcfullyears-method) | It returns the integer value that represents the year on the basis of universal time. |
| [getUTCHours()](https://www.javatpoint.com/javascript-date-getutchours-method) | It returns the integer value between 0 and 23 that represents the hours on the basis of universal time. |
| [getUTCMinutes()](https://www.javatpoint.com/javascript-date-getutcminutes-method) | It returns the integer value between 0 and 59 that represents the minutes on the basis of universal time. |
| [getUTCMonth()](https://www.javatpoint.com/javascript-date-getutcmonth-method) | It returns the integer value between 0 and 11 that represents the month on the basis of universal time. |
| [getUTCSeconds()](https://www.javatpoint.com/javascript-date-getutcseconds-method) | It returns the integer value between 0 and 60 that represents the seconds on the basis of universal time. |
| setDate() | It sets the day value for the specified date on the basis of local time. |
| setDay() | It sets the particular day of the week on the basis of local time. |
| setFullYears() | It sets the year value for the specified date on the basis of local time. |
| [setHours()](https://www.javatpoint.com/javascript-date-sethours-method) | It sets the hour value for the specified date on the basis of local time. |
| [setMilliseconds()](https://www.javatpoint.com/javascript-date-setmilliseconds-method) | It sets the millisecond value for the specified date on the basis of local time. |
| [setMinutes()](https://www.javatpoint.com/javascript-date-setminutes-method) | It sets the minute value for the specified date on the basis of local time. |
| setMonth() | It sets the month value for the specified date on the basis of local time. |
| [setSeconds()](https://www.javatpoint.com/javascript-date-setseconds-method) | It sets the second value for the specified date on the basis of local time. |
| [setUTCDate()](https://www.javatpoint.com/javascript-date-setutcdate-method) | It sets the day value for the specified date on the basis of universal time. |
| setUTCDay() | It sets the particular day of the week on the basis of universal time. |
| [setUTCFullYears()](https://www.javatpoint.com/javascript-date-setutcfullyear-method) | It sets the year value for the specified date on the basis of universal time. |
| [setUTCHours()](https://www.javatpoint.com/javascript-date-setutchours-method) | It sets the hour value for the specified date on the basis of universal time. |
| setUTCMilliseconds() | It sets the millisecond value for the specified date on the basis of universal time. |
| [setUTCMinutes()](https://www.javatpoint.com/javascript-date-setutcminutes-method) | It sets the minute value for the specified date on the basis of universal time. |
| [setUTCMonth()](https://www.javatpoint.com/javascript-date-setutcmonth-method) | It sets the month value for the specified date on the basis of universal time. |
| [setUTCSeconds()](https://www.javatpoint.com/javascript-date-setutcseconds-method) | It sets the second value for the specified date on the basis of universal time. |
| [toDateString()](https://www.javatpoint.com/javascript-date-todatestring-method) | It returns the date portion of a Date object. |
| [toISOString()](https://www.javatpoint.com/javascript-date-toisostring-method) | It returns the date in the form ISO format string. |
| [toJSON()](https://www.javatpoint.com/javascript-date-tojson-method) | It returns a string representing the Date object. It also serializes the Date object during JSON serialization. |
| [toString()](https://www.javatpoint.com/javascript-date-tostring-method) | It returns the date in the form of string. |
| [toTimeString()](https://www.javatpoint.com/javascript-date-totimestring-method) | It returns the time portion of a Date object. |
| [toUTCString()](https://www.javatpoint.com/javascript-date-toutcstring-method) | It converts the specified date in the form of string using UTC time zone. |
| [valueOf()](https://www.javatpoint.com/javascript-date-valueof-method) | It returns the primitive value of a Date object. |

### **JavaScript Date Example**

Let's see the simple example to print date object. It prints date and time both.

<html>

<body>

Current Date and Time: <span id="txt"></span>

<script>

var today=new Date();

document.getElementById('txt').innerHTML=today;

</script>

</body>

</html>

Let's see another code to print date/month/year.

**<script>**

var date=new Date();

var day=date.getDate();

var month=date.getMonth()+1;

var year=date.getFullYear();

document.write("**<br>**Date is: "+day+"/"+month+"/"+year);

**</script>**

### **JavaScript Current Time Example**

Let's see the simple example to print current time of system.

<html>

<body>

Current Time: <span id="txt"></span>

<script>

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

</script>

</body>

</html>

### **JavaScript Digital Clock Example**

Let's see the simple example to display digital clock using JavaScript date object.

There are two ways to set interval in JavaScript: by setTimeout() or setInterval() method.

<html>

<body>

Current Time: <span id="txt"></span>

<script>

window.onload=function(){getTime();}

function getTime(){

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

// add a zero in front of numbers<10

m=checkTime(m);

s=checkTime(s);

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

setTimeout(function(){getTime()},1000);

}

//setInterval("getTime()",1000);//another way

function checkTime(i){

if (i<10){

i="0" + i;

}

return i;

}

</script>

</body> </html>

# **JavaScript Math**

The **JavaScript math** object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

## JavaScript Math Methods

Let's see the list of JavaScript Math methods with description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [abs()](https://www.javatpoint.com/javascript-math-abs-method) | It returns the absolute value of the given number. |
| [acos()](https://www.javatpoint.com/javascript-math-acos-method) | It returns the arccosine of the given number in radians. |
| [asin()](https://www.javatpoint.com/javascript-math-asin-method) | It returns the arcsine of the given number in radians. |
| [atan()](https://www.javatpoint.com/javascript-math-atan-method) | It returns the arc-tangent of the given number in radians. |
| [cbrt()](https://www.javatpoint.com/javascript-math-cbrt-method) | It returns the cube root of the given number. |
| [ceil()](https://www.javatpoint.com/javascript-math-ceil-method) | It returns a smallest integer value, greater than or equal to the given number. |
| [cos()](https://www.javatpoint.com/javascript-math-cos-method) | It returns the cosine of the given number. |
| [cosh()](https://www.javatpoint.com/javascript-math-cosh-method) | It returns the hyperbolic cosine of the given number. |
| [exp()](https://www.javatpoint.com/javascript-math-exp-method) | It returns the exponential form of the given number. |
| [floor()](https://www.javatpoint.com/javascript-math-floor-method) | It returns largest integer value, lower than or equal to the given number. |
| [hypot()](https://www.javatpoint.com/javascript-math-hypot-method) | It returns square root of sum of the squares of given numbers. |
| [log()](https://www.javatpoint.com/javascript-math-log-method) | It returns natural logarithm of a number. |
| [max()](https://www.javatpoint.com/javascript-math-max-method) | It returns maximum value of the given numbers. |
| [min()](https://www.javatpoint.com/javascript-math-min-method) | It returns minimum value of the given numbers. |
| [pow()](https://www.javatpoint.com/javascript-math-pow-method) | It returns value of base to the power of exponent. |
| [random()](https://www.javatpoint.com/javascript-math-random-method) | It returns random number between 0 (inclusive) and 1 (exclusive). |
| [round()](https://www.javatpoint.com/javascript-math-round-method) | It returns closest integer value of the given number. |
| [sign()](https://www.javatpoint.com/javascript-math-sign-method) | It returns the sign of the given number |
| [sin()](https://www.javatpoint.com/javascript-math-sin-method) | It returns the sine of the given number. |
| [sinh()](https://www.javatpoint.com/javascript-math-sinh-method) | It returns the hyperbolic sine of the given number. |
| [sqrt()](https://www.javatpoint.com/javascript-math-sqrt-method) | It returns the square root of the given number |
| [tan()](https://www.javatpoint.com/javascript-math-tan-method) | It returns the tangent of the given number. |
| [tanh()](https://www.javatpoint.com/javascript-math-tanh-method) | It returns the hyperbolic tangent of the given number. |
| [trunc()](https://www.javatpoint.com/javascript-math-trunc-method) | It returns an integer part of the given number. |

## Math.sqrt(n)

The JavaScript math.sqrt(n) method returns the square root of the given number.

<!DOCTYPE html>

<html>

<body>

Square Root of 17 is: <span id="p1"></span>

<script>

document.getElementById('p1').innerHTML=Math.sqrt(17);

</script>

</body>

</html>

## Math.random()

The JavaScript math.random() method returns the random number between 0 to 1.

Random Number is: **<span** id="p2"**></span>**

**<script>**

document.getElementById('p2').innerHTML=Math.random();

**</script>**

## Math.pow(m,n)

The JavaScript math.pow(m,n) method returns the m to the power of n that is mn.

3 to the power of 4 is: **<span** id="p3"**></span>**

**<script>**

document.getElementById('p3').innerHTML=Math.pow(3,4);

**</script>**

## Math.floor(n)

The JavaScript math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

Floor of 4.6 is: **<span** id="p4"**></span>**

**<script>**

document.getElementById('p4').innerHTML=Math.floor(4.6);

**</script>**

## Math.ceil(n)

The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

Ceil of 4.6 is: **<span** id="p5"**></span>**

**<script>**

document.getElementById('p5').innerHTML=Math.ceil(4.6);

**</script>**

## Math.round(n)

The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

Round of 4.3 is: **<span** id="p6"**></span><br>**

Round of 4.7 is: **<span** id="p7"**></span>**

**<script>**

document.getElementById('p6').innerHTML=Math.round(4.3);

document.getElementById('p7').innerHTML=Math.round(4.7);

**</script>**

## Math.abs(n)

The JavaScript math.abs(n) method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

Absolute value of -4 is: **<span** id="p8"**></span>**

**<script>**

document.getElementById('p8').innerHTML=Math.abs(-4);

**</script>**

# **JavaScript Number Object**

The **JavaScript number** object enables you to represent a numeric value. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, you can create number object in JavaScript. For example:

var n=new Number(value);

If value can't be converted to number, it returns NaN(Not a Number) that can be checked by isNaN() method.

You can direct assign a number to a variable also. For example:

<!DOCTYPE html>

<html>

<body>

<script>

var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object

document.write(x+" "+y+" "+z+" "+n);

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

## JavaScript Number Constants

Let's see the list of JavaScript number constants with description.

|  |  |
| --- | --- |
| **Constant** | **Description** |
| MIN\_VALUE | returns the largest minimum value. |
| MAX\_VALUE | returns the largest maximum value. |
| POSITIVE\_INFINITY | returns positive infinity, overflow value. |
| NEGATIVE\_INFINITY | returns negative infinity, overflow value. |
| NaN | represents "Not a Number" value. |

## JavaScript Number Methods

Let's see the list of JavaScript number methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [isFinite()](https://www.javatpoint.com/javascript-number-isfinite-method) | It determines whether the given value is a finite number. |
| [isInteger()](https://www.javatpoint.com/javascript-number-isinteger-method) | It determines whether the given value is an integer. |
| [parseFloat()](https://www.javatpoint.com/javascript-number-parsefloat-method) | It converts the given string into a floating point number. |
| [parseInt()](https://www.javatpoint.com/javascript-number-parseint-method) | It converts the given string into an integer number. |
| [toExponential()](https://www.javatpoint.com/javascript-number-toexponential-method) | It returns the string that represents exponential notation of the given number. |
| [toFixed()](https://www.javatpoint.com/javascript-number-tofixed-method) | It returns the string that represents a number with exact digits after a decimal point. |
| [toPrecision()](https://www.javatpoint.com/javascript-number-toprecision-method) | It returns the string representing a number of specified precision. |
| [toString()](https://www.javatpoint.com/javascript-number-tostring-method) | It returns the given number in the form of string. |

# **JavaScript Boolean**

**JavaScript Boolean** is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor as given below.

Boolean b=new Boolean(value);  [The default value of JavaScript Boolean object is false.]

## JavaScript Boolean Example

**<script>**

document.write(10**<20**);//true

document.write(10**<5**);//false

**</script>**

## JavaScript Boolean Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| constructor | returns the reference of Boolean function that created Boolean object. |
| prototype | enables you to add properties and methods in Boolean prototype. |

## JavaScript Boolean Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| toSource() | returns the source of Boolean object as a string. |
| toString() | converts Boolean into String. |
| valueOf() | converts other type into Boolean. |

# **Browser Object Model**

The **Browser Object Model** (BOM) is used to interact with the browser.

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

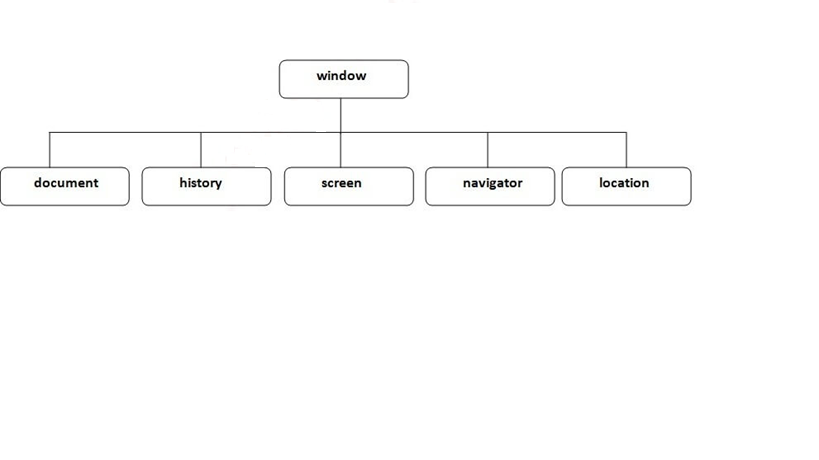
window.alert("hello iHub");

is same as:

alert("hello iHub");

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

#### **Note: The document object represents an html document. It forms DOM (Document Object Model).**

**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. |

#### **Example of alert() in javascript**

It displays alert dialog box. It has message and ok button.

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

function msg(){

 alert("Hello Alert Box");

}

**</script>**

**<input** type="button" value="click" onclick="msg()"**/>**

#### **Example of confirm() in javascript**

It displays the confirm dialog box. It has message with ok and cancel buttons.

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

function msg(){

var v= confirm("Are u sure?");

if(v==true){

alert("ok");

}

else{

alert("cancel");

}

}

**</script>**

**<input** type="button" value="delete record" onclick="msg()"**/>**

#### **Example of prompt() in javascript**

It displays prompt dialog box for input. It has message and textfield.

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

function msg(){

var v= prompt("Who are you?");

alert("I am "+v);

}

**</script>**

**<input** type="button" value="click" onclick="msg()"**/>**

#### **Example of open() in javascript**

It displays the content in a new window.

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

function msg(){

open("http://www.iHubTalent.com");

}

**</script>**

**<input** type="button" value="iHub" onclick="msg()"**/>**

#### **Example of setTimeout() in javascript**

It performs its task after the given milliseconds.

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

function msg(){

setTimeout(

function(){

alert("Welcome to iHub after 2 seconds")

},2000);

}

**</script>**

**<input** type="button" value="click" onclick="msg()"**/>**

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

window.history

Or,

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.

history.back();//for previous page

history.forward();//for next page

history.go(2);//for next 2nd page

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:

window.navigator

Or,

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.

<html>

<body>

<h2>JavaScript Navigator Object</h2>

<script>

document.writeln("<br/>navigator.appCodeName: "+navigator.appCodeName);

document.writeln("<br/>navigator.appName: "+navigator.appName);

document.writeln("<br/>navigator.appVersion: "+navigator.appVersion);

document.writeln("<br/>navigator.cookieEnabled: "+navigator.cookieEnabled);

document.writeln("<br/>navigator.language: "+navigator.language);

document.writeln("<br/>navigator.userAgent: "+navigator.userAgent);

document.writeln("<br/>navigator.platform: "+navigator.platform);

document.writeln("<br/>navigator.onLine: "+navigator.onLine);

</script>

</body>

</html>

# **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 screen object is the window property, so it can be accessed by:

window.screen

Or,

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.

<html>

<body>

<script>

document.writeln("<br/>screen.width: "+screen.width);

document.writeln("<br/>screen.height: "+screen.height);

document.writeln("<br/>screen.availWidth: "+screen.availWidth);

document.writeln("<br/>screen.availHeight: "+screen.availHeight);

document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);

document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);

</script>

</body>

</html>

# **Document Object Model**

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

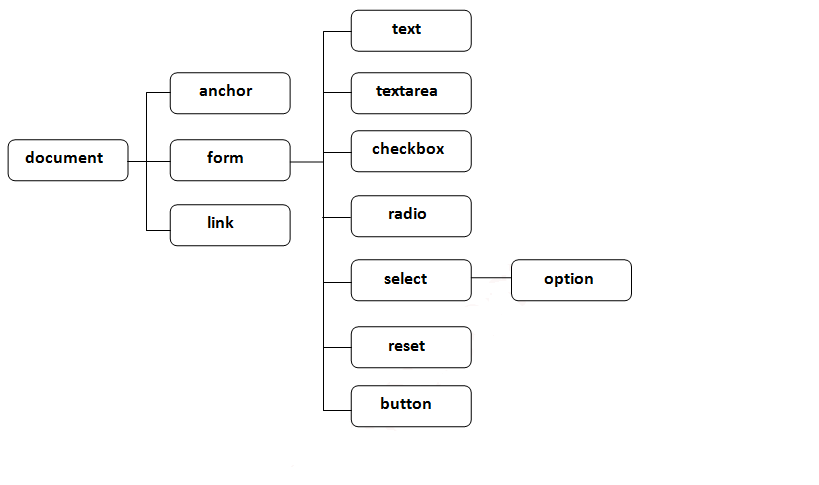
As mentioned earlier, it is the object of window. So

“window.document” Is same as “document”

According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

## Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



## 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 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. |
| getElementsByClassName() | returns all the elements having the given class name. |

### **Accessing field value by document object**

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

**form1** is the name of the form.

**name** is the attribute name of the input text.

**value** is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

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

function printvalue(){

var name=document.form1.name.value;

alert("Welcome: "+name);

}

**</script>**

**<form** name="form1"**>**

Enter Name:**<input** type="text" name="name"**/>**

**<input** type="button" onclick="printvalue()" value="print name"**/>**

**</form>**

# **document.getElementById() method:**

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

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

function getcube(){

var number=document.getElementById("number").value;

alert(number\*number\*number);

}

**</script>**

**<form>**

Enter No:**<input** type="text" id="number" name="number"**/><br/>**

**<input** type="button" value="cube" onclick="getcube()"**/>**

**</form>**

# **Document.GetElementsByClassName():**

The getElementsByClassName() method is used for selecting or getting the elements through their class name value. This DOM method returns an array-like object that consists of all the elements having the specified classname. On calling the getElementsByClassName() method on any particular element, it will search the whole document and will return only those elements which match the specified or given class name.

### **Syntax**

var ele=document.getELementsByClassName('name');

Here, name is the mandatory argument to be passed. It is the string that specifies either a single classname or multiple class names to match.

## Example of getElementsByClassName() Method

Let's look at some examples to know and understand the practical implementation of the method.

**Example**

It is a simple class implementation that returns an array-like object on invoking the variable x.

**<html>**

**<head>** **<h5>**DOM Methods **</h5>** **</head>**

**<body>**

**<div** class="Class"**>**

This is a simple class implementation

**</div>**

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

var x=document.getElementsByClassName('Class');

document.write("On calling x, it will return an arrsy-like object: **<br>**"+x);

**</script>**

**</body>**

**</html>**

Similarly, we can implement the getElementsByClassName() method for returning collections of elements for multiple classes.

### **Difference between getElementsByClassName(), querySelector() and querySelectorAll() Methods**

**getElementsByClassName():** It matches the elements with the specified class name, and returns a set of the matched elements. The returned elements are live HTML collection of elements. These live elements can be further updated if any changes are made in the Document Object Model.

**querySelector():** It returns only a single element that matches the specified classname. If it does not find any matching element, it returns null.

The main point to understand is that all the above-described methods return either one element or a list, but the getELementsByClassName() method serves the **dynamic** updation, and the other two methods serve for the **static**.

# **document.getElementsByName() method:**

The **document.getElementsByName()** method returns all the element of specified name.

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

document.getElementsByName("name")

Here, name is required.

### **Example of document.getElementsByName() method**

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

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

function totalelements()

{

var allgenders=document.getElementsByName("gender");

alert("Total Genders:"+allgenders.length);

}

**</script>**

**<form>**

Male:**<input** type="radio" name="gender" value="male"**>**

Female:**<input** type="radio" name="gender" value="female"**>**

**<input** type="button" onclick="totalelements()" value="Total Genders"**>**

**</form>**

# **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")

Here, name is required.

### **Example of document.getElementsByTagName() method**

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

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

function countpara(){

var totalpara=document.getElementsByTagName("p");

alert("total p tag length is: "+totalpara.length);

}

**</script>**

**<p>**This is a pragraph**</p>**

**<p>**Here we are going to count total number of paragraphs by getElementByTagName() method.**</p>**

**<p>**Let's see the simple example**</p>**

**<button** onclick="countpara()"**>**count paragraph**</button>**

### **Another example of document.getElementsByTagName() method**

In this example, we going to count total number of h2 and h3 tags used in the document.

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

function counth2(){

var totalh2=document.getElementsByTagName("h2");

alert("total h2 tags are: "+totalh2.length);

}

function counth3(){

var totalh3=document.getElementsByTagName("h3");

alert("total h3 tags are: "+totalh3.length);

}

**</script>**

**<h2>**This is h2 tag**</h2>**

**<h2>**This is h2 tag**</h2>**

**<h3>**This is h3 tag**</h3>**

**<h3>**This is h3 tag**</h3>**

**<h3>**This is h3 tag**</h3>**

**<button** onclick="counth2()"**>**count h2**</button>**

**<button** onclick="counth3()"**>**count h3**</button>**

# **Javascript - 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.

<html>

<body>

<script type="text/javascript" >

function showcommentform() {

var data="Name:<br><input type='text' name='name'><br>Comment:<br><textarea rows='5' cols='50'></textarea><br><input type='submit' value='comment'>";

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

}

</script>

<form name="myForm">

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

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

</form>

</body> </html>

### **Show/Hide Comment Form Example using innerHTML**

<!DOCTYPE html**>**

**<html>**

**<head>**

**<title>**First JS**</title>**

**<script>**

var flag=true;

function commentform(){

var cform="**<form** action='Comment'**>**Enter Name:**<br><input** type='text' name='name'**/><br/>**

Enter Email:**<br><input** type='email' name='email'**/><br>**Enter Comment:**<br/>**

**<textarea** rows='5' cols='70'**></textarea><br><input** type='submit' value='Post Comment'**/></form>**";

if(flag){

document.getElementById("mylocation").innerHTML=cform;

flag=false;

}else{

document.getElementById("mylocation").innerHTML="";

flag=true;

}

}

**</script>**

**</head>**

**<body>**

**<button** onclick="commentform()"**>**Comment**</button>**

**<div** id="mylocation"**></div>**

**</body>**

**</html>**

# **Javascript - innerText**

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

## JavascriptinnerText Example

In this example, we are going to display the password strength when releases the key after press.

<!DOCTYPE html>

<html>

<body>

<script type="text/javascript" >

function validate() {

var msg;

if(document.myForm.userPass.value.length>5){

msg="good";

}

else{

msg="poor";

}

document.getElementById('mylocation').innerText=msg;

}

</script>

<form name="myForm">

<input type="password" value="" name="userPass" onkeyup="validate()">

Strength:<span id="mylocation">no strength</span>

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

# **JavaScript Form Validation**

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

## JavaScript Form Validation Example

In this example, we are going to validate the name and password. The name can’t be empty and password can’t be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

<html>

<body>

<script>

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;

if (name==null || name==""){

alert("Name can't be blank");

return false;

}else if(password.length<6){

alert("Password must be at least 6 characters long.");

return false;

}

}

</script>

<form name="myform" method="post" action="http://www.iHubTalent.com/javascriptpages/valid.jsp" onsubmit="return validateform()" >

Name: <input type="text" name="name"><br/>

Password: <input type="password" name="password"><br/>

<input type="submit" value="register">

</form>

</body>

</html>

## JavaScript Retype Password Validation

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript">

function matchpass(){

var firstpassword=document.f1.password.value;

var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){

return true;

}

else{

alert("password must be same!");

return false;

}

}

</script>

</head>

<body>

<form name="f1" action=[http://www.iHubTalent.com/javascriptpages/valid.jsp](http://www.javatpoint.com/javascriptpages/valid.jsp)onsubmit="return matchpass()">

Password:<input type="password" name="password" /><br/>

Re-enter Password:<input type="password" name="password2"/><br/>

<input type="submit">

</form>

</body>

</html>

## JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

<!DOCTYPE html>

<html>

<head>

<script>

function validate(){

var num=document.myform.num.value;

if (isNaN(num)){

document.getElementById("numloc").innerHTML="Enter Numeric value only";

return false;

}else{

return true;

}

}

</script>

</head>

<body>

<form name="myform" action=[http://www.iHubTalent.com/javascriptpages/valid.jsp](http://www.javatpoint.com/javascriptpages/valid.jsp)onsubmit="return validate()" >

Number: <input type="text" name="num"><span id="numloc"></span><br/>

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

</form>

</body>

</html>

## JavaScript validation with image

Let’s see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

<html>

<body>

<script type="text/javascript">

function validate(){

var name=document.f1.name.value;

var passwordlength=document.f1.password.value.length;

var status=false;

if(name==""){

document.getElementById("namelocation").innerHTML=

" <imgsrc=”images/welcome.jpg” /> Please enter your name";

status=false;

}else{

document.getElementById("namelocation").innerHTML=" <imgsrc=’images/welcome.jpg ‘/>";

status=true;

}

if(passwordlength<6){

document.getElementById("passwordlocation").innerHTML=

" <imgsrc=’images/welcome.jpg’ /> Password must be greater than 6";

status=false;

}else{

document.getElementById("passwordlocation").innerHTML=" <imgsrc=’images/welcome.jpg '/>";

}

return status;

}

</script>

<form name="f1" action=[http://www.iHubTalent.com/javascriptpages/valid.jsp](http://www.javatpoint.com/javascriptpages/valid.jsp)onsubmit="return validate()">

<table>

<tr><td>Name:</td><td><input type="text" name="name"/>

<span id="namelocation" style="color:red"></span></td></tr>

<tr><td>Password:</td><td><input type="password" name="password"/>

<span id="passwordlocation" style="color:red"></span></td></tr>

<tr><td colspan="2"><input type="submit" value="register"/></td></tr>

</table>

</form>

</body>

</html>

JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

* email id must contain the @ and . character
* There must be at least one character before and after the @.
* There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

<html>

<body>

<script>

function validateemail()

{

var x=document.myform.email.value;

var atposition=x.indexOf("@");

var dotposition=x.lastIndexOf(".");

if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){

alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);

return false;

}

}

</script>

<body>

<form name="myform" method="post" action="http://www.iHubTalent.com/javascriptpages/valid.jsp" onsubmit="return validateemail();">

Email: <input type="text" name="email"><br/>

<input type="submit" value="register">

</form>

</body>

</html>

## JavaScript OOPs

# **JavaScript Classes**

In JavaScript, classes are the special type of functions. We can define the class just like function declarations and function expressions.

The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.

The class syntax contains two components:

* Class declarations
* Class expressions

## Class Declarations

A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.

### **Class Declarations Example**

### Let's see a simple example of declaring the class.

<!DOCTYPE html>

<html>

<body>

<script>

//Declaring class

class Employee{

//Initializing an object

constructor(id,name){

this.id=id;

this.name=name;

}

//Declaring method

detail(){

document.writeln(this.id+" "+this.name+"<br>")

}

}

//passing object to a variable

var e1=new Employee(101,"Martin Roy");

var e2=new Employee(102,"Duke William");

e1.detail(); //calling method

e2.detail();

</script>

</body></html>

### **Class Declarations Example: Hoisting**

Unlike function declaration, the class declaration is not a part of JavaScript hoisting. So, it is required to declare the class before invoking it.

Let's see an example.

<!DOCTYPE html>

<html>

<body>

<script>

//Here, we are invoking the class before declaring it.

var e1=new Employee(101,"Martin Roy");

var e2=new Employee(102,"Duke William");

e1.detail(); //calling method

e2.detail();

//Declaring class

class Employee{

//Initializing an object

constructor(id,name){

this.id=id;

this.name=name;

}

detail(){

document.writeln(this.id+" "+this.name+"<br>")

}

}

</script>

</body>

</html>

### **Class Declarations Example: Re-declaring Class**

A class can be declared once only. If we try to declare class more than one time, it throws an error.

Let's see an example.

<!DOCTYPE html>

<html>

<body>

<script>

//Declaring class

class Employee{

//Initializing an object

constructor(id,name){

this.id=id;

this.name=name;

}

detail(){

document.writeln(this.id+" "+this.name+"<br>")

}

}

//passing object to a variable

var e1=new Employee(101,"Martin Roy");

var e2=new Employee(102,"Duke William");

e1.detail(); //calling method

e2.detail();

//Re-declaring class

class Employee {}

</script>

</body></html>

## Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class. So, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However, this will not be possible with class declaration.

### **Unnamed Class Expression**

The class can be expressed without assigning any name to it.

Let's see an example.

<!DOCTYPE html>

<html>

<body>

<script>

var emp = class {

constructor(id, name) {

this.id = id;

this.name = name;

}

};

document.writeln(emp.name);

</script>

</body> </html>

### **Class Expression Example: Re-declaring Class**

Unlike class declaration, the class expression allows us to re-declare the same class. So, if we try to declare the class more than one time, it throws an error.

<!DOCTYPE html>

<html>

<body>

<script>

//Declaring class

var emp=class {

//Initializing an object

constructor(id,name){

this.id=id;

this.name=name;

}

//Declaring method

detail(){

document.writeln(this.id+" "+this.name+"<br>")

}

}

//passing object to a variable

var e1=new emp(101,"Martin Roy");

var e2=new emp(102,"Duke William");

e1.detail(); //calling method

e2.detail();

//Re-declaring class

var emp=class {

//Initializing an object

constructor(id,name){

this.id=id;

this.name=name;

}

detail(){

document.writeln(this.id+" "+this.name+"<br>")

}

}

//passing object to a variable

var e1=new emp(103,"James Bella");

var e2=new emp(104,"Nick Johnson");

e1.detail(); //calling method

e2.detail();

</script>

</body>

</html>

### **Named Class Expression Example**

We can express the class with the particular name. Here, the scope of the class name is up to the class body. The class is retrieved using class.name property.

<!DOCTYPE html>

<html>

<body>

<script>

var emp = class Employee {

constructor(id, name) {

this.id = id;

this.name = name;

}

};

document.writeln(emp.name);

/\*document.writeln(Employee.name);

Error occurs on console:

"ReferenceError: Employee is not defined

\*/

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

# **JavaScript Prototype Object**

JavaScript is a prototype-based language that facilitates the objects to acquire properties and features from one another. Here, each object contains a prototype object.

In JavaScript, whenever a function is created the prototype property is added to that function automatically. This property is a prototype object that holds a constructor property.

## Syntax:

ClassName.prototype.methodName

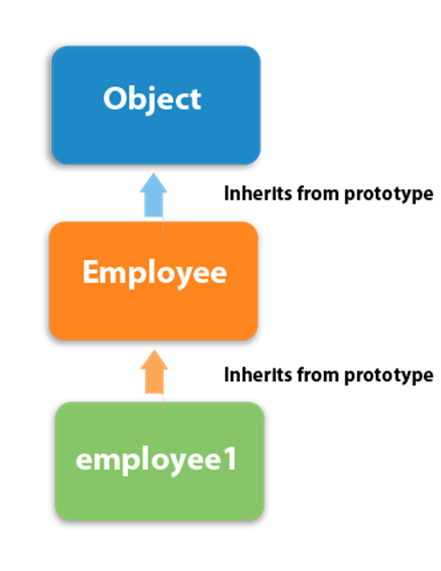
## What is the requirement of a prototype object?

Whenever an object is created in JavaScript, its corresponding functions are loaded into memory. So, a new copy of the function is created on each object creation.

In a prototype-based approach, all the objects share the same function. This ignores the requirement of creating a new copy of function for each object. Thus, the functions are loaded once into the memory.

## Prototype Chaining

In JavaScript, each object contains a prototype object that acquires properties and methods from it. Again an object's prototype object may contain a prototype object that also acquires properties and methods, and so on. It can be seen as prototype chaining.



### **JavaScript Prototype Object Example 1**

Let's see an example to add a new method to the constructor function.

<!DOCTYPE html>

<html>

<body>

<script>

function Employee(firstName,lastName){

this.firstName=firstName;

this.lastName=lastName;

}

Employee.prototype.fullName=function(){

return this.firstName+" "+this.lastName;

}

var employee1=new Employee("Martin","Roy");

var employee2=new Employee("Duke", "William");

document.writeln(employee1.fullName()+"<br>");

document.writeln(employee2.fullName());

</script>

</body>

</html>

### **Example 2**

Let's see an example to add a new property to the constructor function.

<!DOCTYPE html>

<html>

<body>

<script>

function Employee(firstName,lastName){

this.firstName=firstName;

this.lastName=lastName;

}

Employee.prototype.company="iHub"

var employee1=new Employee("Martin","Roy");

var employee2=new Employee("Duke", "William");

document.writeln(employee1.firstName+" "+employee1.lastName+" "+employee1.company+"<br>");

document.writeln(employee2.firstName+" "+employee2.lastName+" "+employee2.company);

</script>

</body>

</html>

# **JavaScript Constructor Method**

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

## Points to remember

* The constructor keyword is used to declare a constructor method.
* The class can contain one constructor method only.
* JavaScript allows us to use parent class constructor through super keyword.

### **Constructor Method Example**

<!DOCTYPE html>

<html>

<body>

<script>

class Employee {

constructor() {

this.id=101;

this.name = "Martin Roy";

}

}

var emp = new Employee();

document.writeln(emp.id+" "+emp.name);

</script>

</body>

</html>

### **Constructor Method Example: super keyword**

The super keyword is used to call the parent class constructor. Let's see an example.

<!DOCTYPE html>

<html>

<body>

<script>

class CompanyName{

constructor(){

this.company="iHub";

}

}

class Employee extends CompanyName {

constructor(id,name) {

super();

this.id=id;

this.name=name;

}

}

var emp = new Employee(1,"John");

document.writeln(emp.id+" "+emp.name+" "+emp.company);

</script>

</body>

</html>

#### **Note - If we didn't specify any constructor method, JavaScript use default constructor method.**

# **JavaScript static Method**

The JavaScript provides static methods that belong to the class instead of an instance of that class. So, an instance is not required to call the static method. These methods are called directly on the class itself.

## Points to remember

* The static keyword is used to declare a static method.
* The static method can be of any name.
* A class can contain more than one static method.
* If we declare more than one static method with a similar name, the JavaScript always invokes the last one.
* The static method can be used to create utility functions.
* We can use this keyword to call a static method within another static method.
* We cannot use this keyword directly to call a static method within the non-static method. In such case, we can call the static method either using the class name or as the property of the constructor.

### **JavaScript static Method Example 1**

<!DOCTYPE html>

<html>

<body>

<script>

class Test{

static display(){

return "static method is invoked"

}

}

document.writeln(Test.display());

</script>

</body></html>

### **Example 2**

<!DOCTYPE html>

<html>

<body>

<script>

class Test{

static display1(){

return "static method is invoked"

}

static display2(){

return "static method is invoked again"

}

}

document.writeln(Test.display1()+"<br>");

document.writeln(Test.display2());

</script>

</body>

</html>

### **Example 3**

Let's see an example to invoke more than one static method with similar names.

<!DOCTYPE html>

<html>

<body>

<script>

class Test{

static display(){

return "static method is invoked"

}

static display(){

return "static method is invoked again"

}

}

document.writeln(Test.display());

</script>

</body>

</html>

### **Example 4**

Let's see an example to invoke a static method within the constructor.

<!DOCTYPE html>

<html>

<body>

<script>

class Test {

constructor() {

document.writeln(Test.display()+"<br>");

document.writeln(this.constructor.display());

}

static display() {

return "static method is invoked"

}

}

var t=new Test();

</script>

</body>

</html>

### **Example 5**

Let's see an example to invoke a static method within the non-static method.

<!DOCTYPE html>

<html>

<body>

<script>

class Test {

static display() {

return "static method is invoked"

}

show() {

document.writeln(Test.display()+"<br>");

}

}

var t=new Test();

t.show();

</script>

</body>

</html>

# **JavaScript Encapsulation**

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

* Use var keyword to make data members private.
* Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

**Read/Write** - Here, we use setter methods to write the data and getter methods read that data.

**Read Only** - In this case, we use getter methods only.

**Write Only** - In this case, we use setter methods only.

### **JavaScript Encapsulation Example**

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

**<script>**

class Student{

     constructor(){

        var name;

        var marks;

     }

         getName(){

           return this.name;

         }

       setName(name){

         this.name=name;

       }

       getMarks(){

         return this.marks;

       }

     setMarks(marks){

       this.marks=marks;

    }

}

    var stud=new Student();

     stud.setName("John");

      stud.setMarks(80);

      document.writeln(stud.getName()+" "+stud.getMarks());

**</script>**

### **JavaScript Encapsulation Example: Validate**

In this example, we validate the marks of the student.

<!DOCTYPE html>

<html>

<body>

<script>

class Student{

constructor(){

var name;

var marks;

}

getName(){

return this.name;

}

setName(name){

this.name=name;

}

getMarks(){

return this.marks;

}

setMarks(marks){

if(marks<0||marks>100){

alert("Invalid Marks");

}

else{

this.marks=marks;

}

}

}

var stud=new Student();

stud.setName("John");

stud.setMarks(110);//alert() invokes

document.writeln(stud.getName()+" "+stud.getMarks());

</script>

</body>

</html>

### **JavaScript Encapsulation Example: Prototype-based approach:**

Here, we perform prototype-based encapsulation.

<!DOCTYPE html>

<html>

<body>

<script>

function Student(name,marks){

var s\_name=name;

var s\_marks=marks;

Object.defineProperty(this,"name",{

get:function(){

return s\_name;

},

set:function(s\_name){

this.s\_name=s\_name;

}

});

Object.defineProperty(this,"marks",{

get:function(){

return s\_marks;

},

set:function(s\_marks){

this.s\_marks=s\_marks;

}

});

}

var stud=new Student("John",80);

document.writeln(stud.name+" "+stud.marks);

</script>

</body>

</html>

# **JavaScript Inheritance**

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.

The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.

## Points to remember

* It maintains an IS-A relationship.
* The extends keyword is used in class expressions or class declarations.
* Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
* We can also use a prototype-based approach to achieve inheritance.

### **JavaScript extends Example: inbuilt object**

In this example, we extends **Date** object to display today's date.

<!DOCTYPE html>

<html>

<body>

<script>

class Moment extends Date {

constructor() {

super();

}

}

var m=new Moment();

document.writeln("Current date:")

document.writeln(m.getDate()+"-"+(m.getMonth()+1)+"-"+m.getFullYear());

</script>

</body>

</html>

Let's see one more example to display the year value from the given date.

<!DOCTYPE html>

<html>

<body>

<script>

class Moment extends Date {

constructor(year) {

super(year);

}

}

var m=new Moment("August 15, 1947 20:22:10");

document.writeln("Year value:")

document.writeln(m.getFullYear());

</script>

</body>

</html>

### **JavaScript extends Example: Custom class**

In this example, we declare sub-class that extends the properties of its parent class.

<!DOCTYPE html>

<html>

<body>

<script>

class Bike{

constructor(){

this.company="Honda";

}

}

class Vehicle extends Bike {

constructor(name,price) {

super();

this.name=name;

this.price=price;

}

}

var v = new Vehicle("Shine","70000");

document.writeln(v.company+" "+v.name+" "+v.price);

</script>

</body>

</html>

### **JavaScript extends Example: a Prototype-based approach**

Here, we perform prototype-based inheritance. In this approach, there is no need to use class and extends keywords.

**<script>**

//Constructor function

function Bike(company){

     this.company=company;

}

Bike.prototype.getCompany=function(){

   return this.company;

}

//Another constructor function

function Vehicle(name,price) {

  this.name=name;

   this.price=price;

}

var bike = new Bike("Honda");

Vehicle.prototype=bike; //Now Bike treats as a parent of Vehicle.

var vehicle=new Vehicle("Shine",70000);

document.writeln(vehicle.getCompany()+" "+vehicle.name+" "+vehicle.price);

**</script>**

# **JavaScript Polymorphism**

The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects. As JavaScript is not a type-safe language, we can pass any type of data members with the methods.

### **JavaScript Polymorphism Example 1**

Let's see an example where a child class object invokes the parent class method.

<!DOCTYPE html>

<html>

<body>

<script>

class A{

display(){

document.writeln("A is invoked");

}

}

class B extends A{}

var b=new B();

b.display();

</script>

</body>

</html>

### **Example 2**

Let's see an example where a child and parent class contains the same method. Here, the object of child class invokes both classes method.

<!DOCTYPE html>

<html>

<body>

<script>

class A{

display(){

document.writeln("A is invoked<br>");

}

}

class B extends A{

display(){

document.writeln("B is invoked");

}

}

var a=[new A(), new B()]

a.forEach(function(msg){

msg.display();

});

</script>

</body>

</html>

### **Example 3**

Let's see the same example with prototype-based approach.

<!DOCTYPE html>

<html>

<body>

<script>

function A(){ }

A.prototype.display=function(){

return "A is invoked";

}

function B(){ }

B.prototype=Object.create(A.prototype);

var a=[new A(), new B()]

a.forEach(function(msg){

document.writeln(msg.display()+"<br>");

});

<script>

</body>

</html>

# **JavaScript Abstraction**

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

## Points to remember

* We cannot create an instance of Abstract Class.
* It reduces the duplication of code.

## JavaScript Abstraction Example

### **Example 1**

Let's check whether we can create an instance of Abstract class or not.

**<script>**

//Creating a constructor function

function Vehicle(){

     this.vehicleName= vehicleName;

     throw new Error("You cannot create an instance of Abstract class");

}

Vehicle.prototype.display=function(){

     return this.vehicleName;

}

var vehicle=new Vehicle();

**</script>**

### **Example 2**

Let's see an example to achieve abstraction.

<!DOCTYPE html>

<html>

<body>

<script>

//Creating a constructor function

function Vehicle(){

this.vehicleName="vehicleName";

throw new Error("You cannot create an instance of Abstract Class");

}

Vehicle.prototype.display=function(){

return "Vehicle is: "+this.vehicleName;

}

//Creating a constructor function

function Bike(vehicleName){

this.vehicleName=vehicleName;

}

//Creating object without using the function constructor

Bike.prototype=Object.create(Vehicle.prototype);

var bike=new Bike("Honda");

document.writeln(bike.display());

</script>

</body>

</html>

### **Example 3**

In this example, we use instanceof operator to test whether the object refers to the corresponding class.

<!DOCTYPE html>

<html>

<body>

<script>

//Creating a constructor function

function Vehicle(){

this.vehicleName=vehicleName;

throw new Error("You cannot create an instance of Abstract class");

}

//Creating a constructor function

function Bike(vehicleName){

this.vehicleName=vehicleName;

}

Bike.prototype=Object.create(Vehicle.prototype);

var bike=new Bike("Honda");

document.writeln(bike instanceof Vehicle);

document.writeln(bike instanceof Bike);

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

### **Example 4**

<!DOCTYPEhtml>

<html>

    <body>

        <script>

            classEmployee {

                constructor() {

                    if(this.constructor == Employee){

                        thrownewError(" Object of Abstract Class cannot be created");

                    }

                }

                display(){

                    thrownewError("Abstract Method has no implementation");

                }

            }

            classManagerextendsEmployee {

                display(){

                    //super.display();

                    console.log("I am a Manager");

                }

            }

            //var emp = new Employee;

            varmang=newManager();

            mang.display();

        </script>

    </body>

</html>

## JavaScript Cookies

# **JavaScript Cookies**

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.

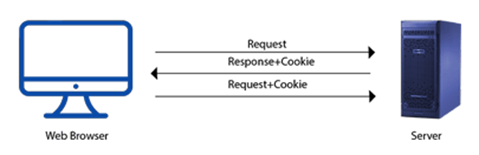
A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

## How Cookies Works?

* When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
* So, to recognize the old user, we need to add the cookie with the response from the server.

browser at the client-side.

* Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



## How to create a Cookie in JavaScript?

In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

The following syntax is used to create a cookie:

document.cookie="name=value";

## JavaScript Cookie Example

### **Example 1**

<!DOCTYPE html**>**

**<html>**

**<head>**  **</head>**

**<body>**

**<input** type="button" value="setCookie" onclick="setCookie()"**>**

**<input** type="button" value="getCookie" onclick="getCookie()"**>**

**<script>**

    function setCookie(){

         document.cookie="username=Duke Martin";

    }

    function getCookie(){

         if(document.cookie.length!=0){

         alert(document.cookie);

         }

         else{

         alert("Cookie not available");

         }

    }

**</script>**

**</body>**

**</html>**

### **Example 2**

Here, we display the cookie's name-value pair separately.

<!DOCTYPE html**>**

**<html>**

**<head>**  **</head>**

**<body>**

**<input** type="button" value="setCookie" onclick="setCookie()"**>**

**<input** type="button" value="getCookie" onclick="getCookie()"**>**

**<script>**

     function setCookie(){

         document.cookie="username=Duke Martin";

    }

     function getCookie(){

         if(document.cookie.length!=0){

             var array=document.cookie.split("=");

         alert("Name="+array[0]+" "+"Value="+array[1]);

         }

        else{

         alert("Cookie not available");

         }

     }

**</script>**

**</body>**

**</html>**

### **Example 3**

In this example, we provide choices of color and pass the selected color value to the cookie. Now, cookie stores the last choice of a user in a browser. So, on reloading the web page, the user's last choice will be shown on the screen.

<!DOCTYPE html**>**

**<html>**

**<head>** **</head>**

**<body>**

**<select** id="color" onchange="display()"**>**

**<option** value="Select Color"**>**Select Color**</option>**

**<option** value="yellow"**>**Yellow**</option>**

**<option** value="green"**>**Green**</option>**

**<option** value="red"**>**Red**</option>**

**</select>**

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

                 function display(){

                     var value = document.getElementById("color").value;

                     if (value != "Select Color"){

                         document.bgColor = value;

                         document.cookie = "color=" + value;

                     }

                 }

                 window.onload = function (){

                     if (document.cookie.length != 0){

                         var array = document.cookie.split("=");

                         document.getElementById("color").value = array[1];

                         document.bgColor = array[1];

                     }

                 }

**</script>**

**</body>**

**</html>**

# **Cookie Attributes**

JavaScript provides some optional attributes that enhance the functionality of cookies. Here, is the list of some attributes with their description.

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| expires | It maintains the state of a cookie up to the specified date and time. |
| max-age | It maintains the state of a cookie up to the specified time. Here, time is given in seconds. |
| path | It expands the scope of the cookie to all the pages of a website. |
| domain | It is used to specify the domain for which the cookie is valid. |

## Cookie expires attribute

The cookie expires attribute provides one of the ways to create a persistent cookie. Here, a date and time are declared that represents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Let's see an example of cookie expires attribute.

<!DOCTYPE html**>**

**<html>**

**<head>**  **</head>**

**<body>**

**<input** type="button" value="setCookie" onclick="setCookie()"**>**

**<input** type="button" value="getCookie" onclick="getCookie()"**>**

**<script>**

     function setCookie(){

document.cookie="username=Duke Martin;expires=Sun, 20 Aug 2030 12:00:00 UTC";

     }

     function getCookie(){

        if(document.cookie.length!=0){

            var array=document.cookie.split("=");

         alert("Name="+array[0]+" "+"Value="+array[1]);

         }

         else{

         alert("Cookie not available");

         }

    }

**</script>**

**</body>**

**</html>**

## Cookie max-age attribute

The cookie max-age attribute provides another way to create a persistent cookie. Here, time is declared in seconds. A cookie is valid up to the declared time only.

Let's see an example of cookie max-age attribute.

<!DOCTYPE html**>**

**<html>**

**<head>**  **</head>**

**<body>**

**<input** type="button" value="setCookie" onclick="setCookie()"**>**

**<input** type="button" value="getCookie" onclick="getCookie()"**>**

**<script>**

     function setCookie(){

document.cookie="username=Duke Martin;max-age=" + (60 \* 60 \* 24 \* 365) + ";"

     }

     function getCookie(){

         if(document.cookie.length!=0){

             var array=document.cookie.split("=");

         alert("Name="+array[0]+" "+"Value="+array[1]);

         }

        else{

         alert("Cookie not available");

         }

     }

**</script>**

**</body>**

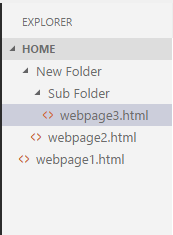
**</html>**

## Cookie path attribute

If a cookie is created for a webpage, by default, it is valid only for the current directory and sub-directory. JavaScript provides a path attribute to expand the scope of cookie up to all the pages of a website.

## Cookie path attribute Example

Let's understand the path attribute with the help of an example.



Here, if we create a cookie for webpage2.html, it is valid only for itself and its sub-directory (i.e., webpage3.html). It is not valid for webpage1.html file.

In this example, we use path attribute to enhance the visibility of cookies up to all the pages. Here, you all just need to do is to maintain the above directory structure and put the below program in all three web pages. Now, the cookie is valid for each web page.

<!DOCTYPE html**>**

**<html>**

**<head>**  **</head>**

**<body>**

**<input** type="button" value="setCookie" onclick="setCookie()"**>**

**<input** type="button" value="getCookie" onclick="getCookie()"**>**

**<script>**

     function setCookie(){

document.cookie="username=Duke Martin;max-age=" + (60 \* 60 \* 24 \* 365) + ";path=/;"

    }

     function getCookie(){

         if(document.cookie.length!=0){

             var array=document.cookie.split("=");

        alert("Name="+array[0]+" "+"Value="+array[1]);

         }

         else{

         alert("Cookie not available");

         }

     }

**</script>**

**</body>**

**</html>**

Cookie domain attribute

A JavaScript domain attribute specifies the domain for which the cookie is valid. Let's suppose if we provide any domain name to the attribute such like:

domain=iHubTalent.com

Here, the cookie is valid for the given domain and all its sub-domains.

However, if we provide any sub-domain to the attribute such like:

domain=training.iHubTalent.com

Here, the cookie is valid only for the given sub-domain. So, it's a better approach to provide domain name instead of sub-domain.

# **Cookie with multiple Name-Value pairs**

In JavaScript, a cookie can contain only a single name-value pair. However, to store more than one name-value pair, we can use the following approach: -

* Serialize the custom object in a JSON string, parse it and then store in a cookie.
* For each name-value pair, use a separate cookie.

## Examples to Store Name-Value pair in a Cookie

### **Example 1**

Let's see an example to check whether a cookie contains more than one name-value pair.

<!DOCTYPE html>

<html>

<head></head>

<body>

Name: <input type="text" id="name"><br>

Email: <input type="email" id="email"><br>

Course: <input type="text" id="course"><br>

<input type="button" value="Set Cookie" onclick="setCookie()">

<input type="button" value="Get Cookie" onclick="getCookie()">

<script>

function setCookie(){

//Declaring 3 key-value pairs

Var info="Name="+ document.getElementById("name").value+";Email="+document.getElementById("email").value+";Course="+document.getElementById("course").value;

//Providing all 3 key-value pairs to a single cookie

document.cookie=info;

}

function getCookie(){

if(document.cookie.length!=0){

//Invoking key-value pair stored in a cookie

alert(document.cookie);

}

else{

alert("Cookie not available")

}

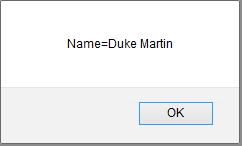
}

</script>

</body>

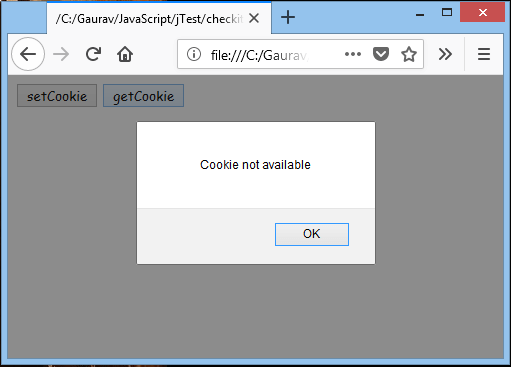
</html>

On clicking **Get Cookie** button, the below dialog box appears.



Here, we can see that only a single name-value is displayed.

However, if you click, **Get Cookie** without filling the form, the below dialog box appears.



### **Example 2**

Let's see an example to store different name-value pairs in a cookie using JSON.

<!DOCTYPE html>

<html>

<head></head>

<body>

Name: <input type="text" id="name"><br>

Email: <input type="email" id="email"><br>

Course: <input type="text" id="course"><br>

<input type="button" value="Set Cookie" onclick="setCookie()">

<input type="button" value="Get Cookie" onclick="getCookie()">

<script>

function setCookie(){

var obj = {};//Creating custom object

obj.name = document.getElementById("name").value;

obj.email = document.getElementById("email").value;

obj.course = document.getElementById("course").value;

//Converting JavaScript object to JSON string

var jsonString = JSON.stringify(obj);

document.cookie = jsonString;

}

function getCookie(){

if( document.cookie.length!=0){

//Parsing JSON string to JSON object

var obj = JSON.parse(document.cookie);

alert("Name="+obj.name+" "+"Email="+obj.email+" "+"Course="+obj.course);

}

else{

alert("Cookie not available");

}

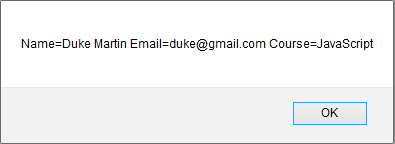
}

</script>

</body>

</html>

On clicking **Get Cookie** button, the below dialog box appears.



Here, we can see that all the stored name-value pairs are displayed.

### **Example 3**

Let's see an example to store each name-value pair in a different cookie.

<!DOCTYPE html>

<html>

<head></head>

<body>

Name: <input type="text" id="name"><br>

Email: <input type="email" id="email"><br>

Course: <input type="text" id="course"><br>

<input type="button" value="Set Cookie" onclick="setCookie()">

<input type="button" value="Get Cookie" onclick="getCookie()">

<script>

function setCookie(){

document.cookie = "name=" + document.getElementById("name").value;

document.cookie = "email=" + document.getElementById("email").value;

document.cookie = "course=" + document.getElementById("course").value;

}

function getCookie(){

if (document.cookie.length != 0){

alert("Name="+document.getElementById("name").value+" Email="+document.getElementById("email").value+" Course="+document.getElementById("course").value);

}

else{

alert("Cookie not available");

}

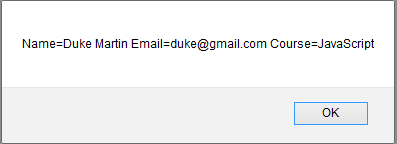
}

</script>

</body>

</html>

On clicking **Get Cookie** button, the below dialog box appears.



Here, also we can see that all the stored name-value pairs are displayed.

# **Deleting a Cookie in JavaScript**

In the previous section, we learned the different ways to set and update a cookie in JavaScript. Apart from that, JavaScript also allows us to delete a cookie. Here, we see all the possible ways to delete a cookie.

## Different ways to delete a Cookie

These are the following ways to delete a cookie:

* A cookie can be deleted by using expire attribute.
* A cookie can also be deleted by using max-age attribute.
* We can delete a cookie explicitly, by using a web browser.

## Examples to delete a Cookie

### **Example 1**

In this example, we use expire attribute to delete a cookie by providing expiry date (i.e. any past date) to it.

<!DOCTYPE html**>**

**<html>**

**<head>**

**</head>**

**<body>**

**<input** type="button" value="Set Cookie" onclick="setCookie()"**>**

**<input** type="button" value="Get Cookie" onclick="getCookie()"**>**

**<script>**

function setCookie(){

document.cookie="name=Martin Roy; expires=Sun, 20 Aug 2000 12:00:00 UTC";

     }

function getCookie(){

     if(document.cookie.length!=0){

     alert(document.cookie);

     }

     else{

         alert("Cookie not avaliable");

     }

}

**</script>**

**</body>**

**</html>**

### **Example 2**

In this example, we use **max-age** attribute to delete a cookie by providing zero or negative number (that represents seconds) to it.

<!DOCTYPE html**>**

**<html>**

**<head>**

**</head>**

**<body>**

**<input** type="button" value="Set Cookie" onclick="setCookie()"**>**

**<input** type="button" value="Get Cookie" onclick="getCookie()"**>**

**<script>**

function setCookie(){

     document.cookie="name=Martin Roy;max-age=0";

}

function getCookie(){

     if(document.cookie.length!=0){

     alert(document.cookie);

     }

     else{

         alert("Cookie not avaliable");

     }

}

**</script>**

**</body>**

**</html>**

### **Example 3**

Let's see an example to set, get and delete multiple cookies.

<!DOCTYPE html**>**

**<html>**

**<head>** **</head>**

**<body>**

**<input** type="button" value="Set Cookie1" onclick="setCookie1()"**>**

**<input** type="button" value="Get Cookie1" onclick="getCookie1()"**>**

**<input** type="button" value="Delete Cookie1" onclick="deleteCookie1()"**><br>**

**<input** type="button" value="Set Cookie2" onclick="setCookie2()"**>**

**<input** type="button" value="Get Cookie2" onclick="getCookie2()"**>**

**<input** type="button" value="Delete Cookie2" onclick="deleteCookie2()"**><br>**

**<input** type="button" value="Display all cookies" onclick="displayCookie()"**>**

**<script>**

function setCookie1() {

     document.cookie="name=Martin Roy";

      cookie1=  document.cookie;

}

function setCookie2() {

     document.cookie="name=Duke William";

      cookie2=  document.cookie;

}

function getCookie1()  {

    if(cookie1.length!=0{

     alert(cookie1);

    }

    else {

        alert("Cookie not available");

    }

}

function getCookie2() {

    if(cookie2.length!=0) {

     alert(cookie2);

    }

    else{

        alert("Cookie not available");

    }

}

function deleteCookie1() {

     document.cookie=cookie1+";max-age=0";

     cookie1=document.cookie;

     alert("Cookie1 is deleted");

}

function deleteCookie2()  {

     document.cookie=cookie2+";max-age=0";

     cookie2=document.cookie;

   alert("Cookie2 is deleted");

}

function displayCookie() {

if(cookie1!=0&&cookie2!=0{

     alert(cookie1+" "+cookie2);

}

else if(cookie1!=0) {

     alert(cookie1);

}

else if(cookie2!=0) {

     alert(cookie2);

}

else{

     alert("Cookie not available");

}

}

**</script>**

**</body>**

**</html>**

### **Example 4**

Let's see an example to delete a cookie explicitly.

<!DOCTYPE html**>**

**<html>**

**<head>** **</head>**

**<body>**

**<input** type="button" value="Set Cookie" onclick="setCookie()"**>**

**<input** type="button" value="Get Cookie" onclick="getCookie()"**>**

**<script>**

function setCookie()  {

     document.cookie="name=Martin Roy";

}

function getCookie() {

     if(document.cookie.length!=0{

     alert(document.cookie);

     }

     else  {

         alert("Cookie not avaliable");

     }

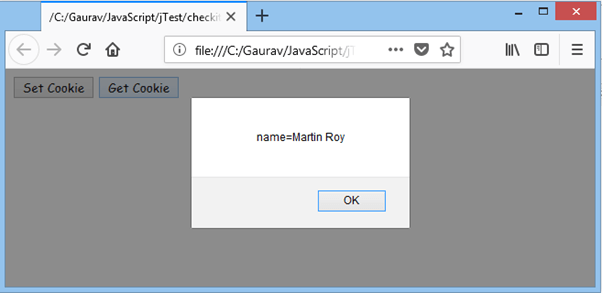
}

**</script>**

**</body>**

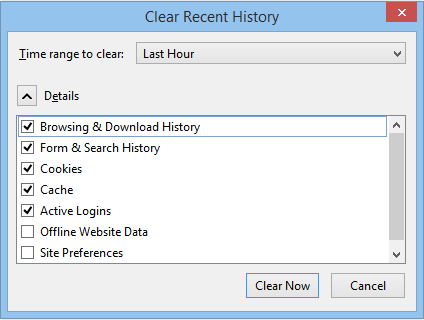
**</html>**

After clicking **Set Cookie** once, whenever we click **Get Cookie**, the cookies key and value is displayed on the screen.



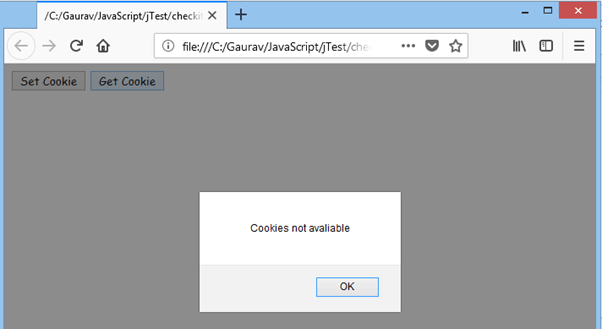
To delete a cookie explicitly, follow the following steps:

* Open Mozilla Firefox.
* Click **Open menu - Library - History - Clear Recent History - Details**.



* Here we can see a **Cookies** checkbox which is already marked. Now, click **Clear Now** to delete the cookies explicitly.

Now, on clicking **Get Cookie**, the below dialog box appears.



Here, we can see that the cookies are deleted.

# **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](https://www.javatpoint.com/html-tutorial), 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 iHub Institute");

}

</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 iHub");

}

</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>

<body onload="window.alert('Page successfully loaded');">

<script>

document.write("The page is loaded successfully");

</script>

</body>

</html>

# **JavaScript addEventListener()**

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

### **Syntax**

element.addEventListener(event, function, useCapture);

Although it has three parameters, the parameters **event** and **function** are widely used. The third parameter is optional to define. The values of this function are defined as follows.

### **Parameter Values**

**event:** It is a required parameter. It can be defined as a string that specifies the event's name.

#### **Note: Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".**

**function:** It is also a required parameter. It is a JavaScript function which responds to the event occur.

**useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

Let's see some of the illustrations of using the addEventListener() method.

### **Example**

It is a simple example of using the addEventListener() method. We have to click the given HTML button to see the effect.

<!DOCTYPE html>

<html>

<body>

<p> Example of the addEventListener() method. </p>

<p> Click the following button to see the effect. </p>

<button id = "btn"> Click me </button>

<p id = "para"></p>

<script>

document.getElementById("btn").addEventListener("click", fun);

function fun() {

document.getElementById("para").innerHTML = "Hello World" + "<br>" + "Welcome to the iHubTalent.com";

}

</script>

</body>

</html>

### **Example**

In this example, we are adding multiple events to the same element.

<!DOCTYPE html>

<html>

<body>

<p> This is an example of adding multiple events to the same element. </p>

<p> Click the following button to see the effect. </p>

<button id = "btn"> Click me </button>

<p id = "para"></p>

<p id = "para1"></p>

<script>

function fun() {

alert("Welcome to the iHubtalent.com");

}

function fun1() {

document.getElementById("para").innerHTML = "This is second function";

}

function fun2() {

document.getElementById("para1").innerHTML = "This is third function";

}

var mybtn = document.getElementById("btn");

mybtn.addEventListener("click", fun);

mybtn.addEventListener("click", fun1);

mybtn.addEventListener("click", fun2);

</script>

</body>

</html>

### **Example**

In this example, we are adding multiple events of a different type to the same element.

<!DOCTYPE html>

<html>

<body>

<p> This is an example of adding multiple events of different type to the same element. </p>

<p> Click the following button to see the effect. </p>

<button id = "btn"> Click me </button>

<p id = "para"></p>

<script>

function fun() {

btn.style.width = "50px";

btn.style.height = "50px";

btn.style.background = "yellow";

btn.style.color = "blue";

}

function fun1() {

document.getElementById("para").innerHTML = "This is second function";

}

function fun2() {

btn.style.width = "";

btn.style.height = "";

btn.style.background = "";

btn.style.color = "";

}

var mybtn = document.getElementById("btn");

mybtn.addEventListener("mouseover", fun);

mybtn.addEventListener("click", fun1);

mybtn.addEventListener("mouseout", fun2);

</script>

</body>

</html>

## Event Bubbling or Event Capturing

Now, we understand the use of the third parameter of JavaScript's addEventListener(), i.e., **useCapture.**

In HTML DOM, **Bubbling** and **Capturing** are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in **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);

We can specify the propagation using the **useCapture** parameter. When it is set to false (which is its default value), then the event uses bubbling propagation, and when it is set to true, there is the capturing propagation.

We can understand the bubbling and capturing using an illustration.

### **Example**

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element.

When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called bubbling.

But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called capturing.

<!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**

The **onclick** event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.

Using JavaScript, this event can be dynamically added to any element. It supports all HTML elements except [<html>](https://www.javatpoint.com/html-html-tag)**,**[<head>](https://www.javatpoint.com/html-head)**,**[<title>](https://www.javatpoint.com/html-title)**,**[<style>](https://www.javatpoint.com/html-style)**,**[<script>](https://www.javatpoint.com/html-script-tag)**,**[<base>](https://www.javatpoint.com/html-base-tag)**,**[<iframe>](https://www.javatpoint.com/html-iframes)**,**[<bdo>](https://www.javatpoint.com/html-bdo-tag)**,**[<br>](https://www.javatpoint.com/html-br-tag)**,**[<meta>](https://www.javatpoint.com/html-meta-tag)**,** and [<param>](https://www.javatpoint.com/html-param-tag). 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.

### **Syntax**

Now, we see the syntax of using the **onclick** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

### **In HTML**

**<element** onclick = "fun()"**>**

### **In JavaScript**

object.onclick = function() { myScript };

### **In JavaScript by using the addEventListener() method**

object.addEventListener("click", myScript);

Let's see how to use **onclick** event by using some illustrations. Now, we will see the examples of using the **onclick** event in HTML, and in JavaScript.

### **Example1 - Using onclick attribute in HTML**

In this example, we are using the [HTML **onclick**](https://www.javatpoint.com/html-button-onclick) attribute and assigning a JavaScript's function to it. When the user clicks the given button, the corresponding function will get executed, and an alert dialog box will be displayed on the screen.

<!DOCTYPE html>

<html>

<head>

<script>

function fun() {

alert("Welcome to the iHubTalent.com");

}

</script>

</head>

<body>

<h3> This is an example of using onclick attribute in HTML. </h3>

<p> Click the following button to see the effect. </p>

<button onclick = "fun()">Click me</button>

</body>

</html>

### **Example2 - Using JavaScript**

In this example, we are using JavaScript's **onclick** event. Here we are using the **onclick** event with the paragraph element.

When the user clicks on the paragraph element, the corresponding function will get executed, and the text of the paragraph gets changed. On clicking the **<p>** element, the background color, size, border, and color of the text will also get change.

<!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 iHubtalent.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>

### **Example3 - Using addEventListener() method**

In this example, we are using JavaScript's **addEventListener()** method to attach a **click** event to the paragraph element. When the user clicks the paragraph element, the text of the paragraph gets changed.

On clicking the paragraph, the background color and font-size of elements will also change.

<!DOCTYPE html>

<html>

<body>

<h3> This is an example of using click 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 iHubTalent.com";

document.getElementsByTagName("body")[0].style.color = "blue";

document.getElementsByTagName("body")[0].style.backgroundColor = "lightgreen";

document.getElementsByTagName("body")[0].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.

### **Syntax**

Now, we see the syntax of creating double click event in HTML and in javascript (without using **addEventListener()** method or by using the **addEventListener()** method).

### **In HTML**

**<element** ondblclick = "fun()"**>**

### **In JavaScript**

object.ondblclick = function() { myScript };

### **In JavaScript by using the addEventListener() method**

object.addEventListener("dblclick", myScript);

Let's see some of the illustrations to understand the double click event.

### **Example - Using ondblclick attribute in HTML**

In this example, we are creating the double click event using the HTML **ondblclick** attribute.

<!DOCTYPE html>

<html>

<body>

<h1 id = "heading" ondblclick = "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 iHubTalent.com ";

}

</script>

</body>

</html>

### **Example - Using JavaScript**

<!DOCTYPE html>

<html>

<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 iHubTalent.com ";

}

</script>

</body>

</html>

### **Example - Using JavaScript's addEventListener() method**

<!DOCTYPE html>

<html>

<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>. </p>

<script>

document.getElementById("heading").addEventListener("dblclick", fun);

function fun() {

document.getElementById("heading").innerHTML = " Welcome to the iHubTalent.com ";

}

</script>

</body>

</html>

# **JavaScript onload**

In JavaScript, this event can apply to launch a particular function when the page is fully displayed. It can also be used to verify the type and version of the visitor's browser. We can check what cookies a page uses by using the **onload** attribute.

In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

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.

The difference between the **document.onload** and **window.onload** is: **document.onload** triggers before the loading of images and other external content. It is fired before the **window.onload**. While the **window.onload** triggers when the entire page loads, including CSS files, script files, images, etc.

### **Syntax**

window.onload = fun()

Let's understand this event by using some examples.

### **Example1**

In this example, there is a div element with a height of 200px and a width of 200px. Here, we are using the **window.onload()** to change the background color, width, and height of the **div** element after loading the web page.

The background color is set to **'red'**, and width and height are set to **300px** each.

<!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.backgroundColor = "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>

### **Example2**

In this example, we are implementing a simple animation by using the properties of the DOM object and functions of javascript. We use the JavaScript function getElementById() for getting the DOM object and then assign that object into a global variable.

<html>

<head>

<script type = "text/javascript">

var img = null;

function init(){

img = document.getElementById('myimg');

img.style.position = 'relative';

img.style.left = '50px';

}

function moveRight(){

img.style.left = parseInt(

img.style.left) + 100 + 'px';

}

window.onload = init;

</script>

</head>

<body>

<form>

<img id = "myimg" src = "train1.png" />

<center>

<p>Click the below button to move the image right</p>

<input type = "button" value = "Click Me" onclick = "moveRight();" />

</center>

</form>

</body>

</html>

### **Example3**

It is a simple example of using the HTML **onload** attribute with the function defined in JavaScript. In this example, the **alert()** function gets called whenever the document refresh.

<!DOCTYPE html>

<html>

<head>

<script>

function fun() {

alert("Hello World!!, Welcome to the iHubTalent.com");

}

</script>

</head>

<body onload = "fun()">

<h1> Example of the HTML onload attribute </h1>

<p> Try to refresh the document to see the effect. </p>

</body>

</html>

# **JavaScript onresize event**

The **onresize** event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use the JavaScript's **window.outerWidth** and **window.outerHeight**events. We can also use the JavaScript's properties such as **innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight** to get the size of an element.

In HTML, we can use the **onresize** attribute and assign a JavaScript function to it. We can also use the [JavaScript's **addEventListener()**](https://www.javatpoint.com/javascript-addeventlistener) method and pass a **resize** event to it for greater flexibility.

### **Syntax**

Now, we see the syntax of using the **onresize** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

### **In HTML**

**<element** onresize = "fun()"**>**

### **In JavaScript**

object.onresize = function() { myScript };

### **In JavaScript by using the addEventListener() method**

object.addEventListener("resize", myScript);

Let's see some of the illustrations to understand the **onresize** event.

### **Example**

In this example, we are using the HTML **onresize** attribute. Here, we are using the **window.outerWidth**and **window.outerHeight** events of JavaScript to get the height and width of the window.

When the user resizes the window, the updated width and height of the window will be displayed on the screen. It will also display how many times the user tried to resize the window. When we change the height of the window, the updated height will change accordingly. Similarly, when we change the width of the window, the updated width will change accordingly.

<!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**

In this example, we are using JavaScript's **onresize** event.

<!DOCTYPE html>

<html>

<body>

<h3> This is an example of using JavaScript's 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 <span id = "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.getElementById("para").innerHTML = res;

var res1 = i += 1;

document.getElementById("s1").innerHTML = res1;

}

</script>

</body>

</html>

### **Example - Using addEventListener() method**

In this example, we are using JavaScript's **addEventListener()** method.

<!DOCTYPE html>

<html>

<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() {

var res = "Width = " + window.outerWidth + "<br>" + "Height = " + window.outerHeight;

document.getElementById("para").innerHTML = res;

var res1 = i += 1;

document.getElementById("s1").innerHTML = res1;

}

</script>

</body>

</html>

# **Exception Handling in JavaScript**

An exception signifies the presence of an abnormal condition which requires special operable techniques. In programming terms, an exception is the anomalous code that breaks the normal flow of the code. Such exceptions require specialized programming constructs for its execution.

## What is Exception Handling

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. **For example**, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

**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…catch block. If an error is present, the catch block will execute, else only 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 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()**.

## Exception Handling Statements

There are following statements that handle if any exception occurs:

* throw statements
* try…catch statements
* try…catch…finally statements.

These exception handling statements are discussed in the next section.

# **JavaScript try…catch**

A try…catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try…catch statements.

Let's discuss each block of statement individually:

**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:** This 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 built-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.

#### **Note: catch {} statement executes only after the execution of the try {} statement. Also, one try block can contain one or more catch blocks.**

### **Syntax:**

try{

expression; } //code to be written.

catch(error){

expression; } // code for handling the error.

### **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 statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

### **Syntax:**

throw exception;

### **try…catch…throw syntax**

try{

throw exception; // user can define their own exception

}

catch(error){

expression; }  // code for handling exception.

The exception can be a string, number, object, or boolean value.

### **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>

With the help of throw statement, users can create their own errors.

## try…catch…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, will definitely execute. It does not care for the output too.

### **Syntax:**

try{

expression;

}

catch(error){

expression;

}

finally{

expression; } //Executable code

### **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>

Therefore, we can also use try/catch/throw/finally keyword together for handling complex code.

## JavaScript Misc

# **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.

## JavaScript this Keyword Example

Let's see a simple example of this keyword.

<script>

var address=

{

company:"iHub",

city:"Noida",

state:"UP",

fullAddress:function() {

**return** **this**.company+" "+**this**.city+" "+**this**.state;

}

};

var fetch=address.fullAddress();

document.writeln(fetch);

</script>

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 function. Here, this keyword refers to the window object.

<script>

var website="iHub";

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:"iHub",

    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(**this**.lang+ " is a popular programming language.");

   }

};

lang\_name(obj.language);

lang\_name(obj.language.bind(obj));

</script>

# **JavaScript Debugging**

Sometimes a code may contain certain mistakes. Being a scripting language, JavaScript didn't show any error message in a browser. But these mistakes can affect the output.

The best practice to find out the error is to debug the code. The code can be debugged easily by using web browsers like Google Chrome, Mozilla Firebox.

## JavaScript Debugging Example

Here, we will find out errors using built-in web browser debugger. To perform debugging, we can use any of the following approaches:

* Using console.log() method
* Using debugger keyword

### **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.

Let's see the simple example to print the result on console.

<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.

JavaScript provides **debugger** keyword to set the breakpoint through the code itself. The **debugger** stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line.

<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 is applicable only for declaration not initialization. It is required to initialize the variables and functions before using their values.

## JavaScript Hoisting Example

Here, we will use the variable and function before declaring them.

### **JavaScript Variable Hoisting**

Let's see the simple example of variable hoisting.

**<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.

#### **Note - The "use strict"; expression can only be placed as the first statement in a script or in a function.**

### **Example 1**

Let's see the example without using strict mode.

<script>

x=10;

console.log(x);

</script>

Let's see the same example by enabling the strict mode.

<script>

"use strict";

x=10;

console.log(x);

</script>

### **Example 2**

<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 (when required). Promise in javascript is used for managing and tackling asynchronous operations.

## Need for JavaScript Promise

Till now, we learned about events and callback functions for handling the data. But, its scope is limited. It is because events were not able to manage and operate asynchronous operations. Thus, Promise is the simplest and better approach for handling asynchronous operations efficiently.

There are two possible differences between Promise and Event Handlers:

1. A Promise can never fail or succeed twice or more. This can happen only once.
2. A Promise can neither switch from success to failure, or failure to success. If a Promise has either succeeded or failed, and after sometime, if any success/failure callback is added, the correct callback will be invoked, no matter the event happened earlier.

## 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 successfully.
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 asynchronous 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, i.e., 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. The methods list is:

|  |  |
| --- | --- |
| **Method Name** | **Summary** |
| Promise.resolve(promise) | This method returns promise only if promise.constructor==Promise. |
| Promise.resolve(thenable) | Makes a new promise from thenable containing then(). |
| Promise.resolve(obj) | Makes a promise resolved for an object. |
| Promise.reject(obj) | Makes a promise rejection for the object. |
| Promise.all(array) | Makes a promise resolved when each item in an array is fulfilled or rejects when items in the array are not fulfilled. |
| 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(fromResolve){

document.write("Promise is"+fromResolve);

}).catch(function(fromReject){

document.write("Promise is "+fromReject);

});

</script>

</body>

</html>

In the above Promise implementation, the Promise constructor takes an argument that callbacks the function. This callback function takes two arguments, i.e.,

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 as the constructors.

Here, with the help of those methods, we will learn to compare dates.

Basically, there are different ways by 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(){

var d1=new Date('2020-01-23'); //yyyy-mm-dd

var d2=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

<html>

<head> Comparing Date and time</br></head>

<body>

<script>

var d1=new Date("Apr 17, 2019 12:10:10"); //mm dd, yyyyhh:mm:ss

var d2=new Date("Dec 1, 2019 12:10:30"); //mm dd, yyyyhh:mm:ss

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 in terms of both time and date");

}

else{

document.write("Both date and time are same and equal");

}

</script>

</body>

</html>

**Example2:** Comparing same dates with disimilar timings

<html>

<head> Comparing same date but different time</br></head>

<body>

<script>

var d1=new Date("2018-01-10, 12:10:10"); //yyyy-mm-dd hh:mm:ss

var d2=new Date("2018-01-10, 12:10:50"); //yyyy-mm-dd hh:mm:ss

if(d1>d2){

document.write("False, d1 & d2 dates are same but d2 time is greater than d1 time");

}

else if(d1<d2){

document.write("True, d2 time is greater than d1 time.");

}

else{

document.write("Both date and time are same and equal");

}

</script>

</body>

</html>

## Comparing date with getTime()

A better approach to make comparison between dates is to use **getTime()** function. This function lets converting date into numeric value to directly compare them.

**Example1:** 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>

**Example2:** Comparing two different dates with different timings.

<html>

<head> Comparing Dates</br></head>

<body>

<script>

var d1=new Date("2019-10-10, 10:10:10");

var d2=new Date("2019-11-02, 14:19:05");

if(d1.getTime()<d2.getTime()){

document.write("True, d1 date and time are smaller than d2 date and time");

}

else if(d1.getTime()>d2.getTime()){

document.write("False, d2 date and time are greater than d1");

}

else{

document.write("True, d1 and d2 have same time and date");

}

</script>

</body>

</html>

## Changing Date Format

We can also change or set the format through JavaScript code. The function getFullYear(), GetMonth(), and getDate() allows to set the format of date accordingly.

**Example1:** Changing the date format to 'yyyy-mm-dd'.

<html>

<head><h3>Changing date format</h3></br></head>

<body>

<script>

var current\_date=new Date(); //fetches current date

var set\_to=current\_date.getFullYear()+"-"+(current\_date.getMonth()+1)+"-"+current\_date.getDate();

document.write("The format followed is yyyy-dd-mm: "+set\_to);

</script>

</body>

</html>

**Example2:** Changing the datetime format to 'yyyy-dd-mm hh:mm:ss'.

<html>

<head><h3>Changing date format</h3></br></head>

<body>

<script>

var current\_datetime=new Date(); //fetches current date and time

var set\_to=current\_datetime.getFullYear()+"-"+(current\_datetime.getMonth()+1)+"-"+current\_datetime.getDate()+" "+current\_datetime.getHours()+":"+current\_datetime.getMinutes()+":"+current\_datetime.getSeconds();

document.write("The format followed is yyyy-dd-mm hh:mm:ss : "+set\_to);

</script>

</body>

</html>

# **JavaScript array.length property**

The length property returns the number of elements in an array in the form of a 32-bit unsigned integer. We can also say that the **length** property returns a number that represents the number of array elements. The return value is always larger than the highest array index.

The **length** property can also be used to set the number of elements in an array. We have to use the assignment operator in conjunction 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](https://www.javatpoint.com/jquery-tutorial). In [JavaScript](https://www.javatpoint.com/javascript-tutorial), it is invalid to use **array.size()** method so, we use **array.length** property to calculate the size of an array.

### **Syntax**

**The following syntax is used to return the length of an array**

array.length

**The following syntax is used to set the length of an array**

array.length = number

For better understanding, let's see some of the illustrations of using **array.length** property.

### **Example1**

It is a simple example to understand how to calculate the length of an array using the **array.length** property.

<html>

<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>

### **Example2**

In this example, we are setting the length of an array by using the **array.length** property. Initially, the array contains two elements, so at the beginning, the length is 2. Then we increase the length of the array to 9.

In the output, the values of the array are separated by commas. After increasing the length, the array contains two defined and seven undefined values separated by a comma. Then we insert five array elements and print them. Now, the array contains seven defined and two undefined values.

<html>

<body>

<h3> Here, we are setting the length of an array. </h3>

<script>

var arr = [100, 200];

document.write(" Before setting the length, the array elements are: " + arr);

arr.length = 9;

document.write("<br><br> After setting the length, the array elements are: " + arr);

// It will print [ 1, 2, <7 undefined items> ]

arr[2] = 300;

arr[3] = 400;

arr[4] = 500;

arr[5] = 600;

document.write("<br><br> After inserting some array elements: " + arr);

</script>

</body>

</html>

### **Example3**

In this example, the index of the array is non-numeric. Here, the array contains five elements with the non-numeric index. We are applying the length property on the given array to see the effect. Now let's see how the **array.length** property works on the non-numeric index of the array.

<html>

<body>

<h3>There are five array elements but the index of the array is non numeric. </h3>

<script>

var arr = new Array();

arr['a'] = 100;

arr['b'] = 200;

arr['c'] = 300;

arr['d'] = 400;

arr['e'] = 500;

document.write("The length of array is: " + arr.length);

</script>

</body>

</html>

We can also use the length property to find out the number of words in the string. Let's understand it with an example.

### **Example4**

In this example, we are using the length property to display the number of words present in the string. Here, we are creating an array and use the **split()** function for the array elements. We are splitting the string from the whitespace (**" "**) character.

If we direct apply the length property on the string, then it gives us the number of characters in the string. But in this example, we will understand how to calculate the number of words in the string.

<html>

<body>

<script>

var str = "Welcome to the ihubtalent.com";

var arr = new Array();

arr = str.split(" ");

document.write(" The given string is: " + str);

document.write("<br><br> Number Of Words: "+ arr.length);

document.write("<br><br> Number of characters in the string: " + str.length);

</script>

</body>

</html>

# **JavaScript alert()**

The **alert()** method in JavaScript is used to display a virtual alert box. It is mostly used to give a warning message to the users. It displays an alert dialog box that consists of some specified message (which is optional) and an OK button. When the dialog box pops up, we have to click "OK" to proceed.

The alert dialog box takes the focus and forces the user to read the specified message. So, we should avoid overusing this method because it stops the user from accessing the other parts of the webpage until the box is closed.

We can understand the usefulness of the alert method using an example. Suppose we have to fill a form for an identity card. It asks about the date of birth for the eligibility criteria of the identity card. If the age is 18 years or above, then the process will continue. Otherwise, it will show a warning message that the age is below 18 years. This warning message is the 'Alert Box'.

Another example is suppose a user is required to fill the form in which some mandatory fields are required to enter some text, but the user forgets to provide the input. As the part of the validation, we can use the alert dialog box to show a warning message related to fill the textfield.

Rather than showing the warnings or errors, the alert dialog box can be used for normal messages such as '**welcome back', 'Hello XYZ'**, etc.

### **Syntax**

alert(message)

### **Values**

**message:** It is an optional string that specifies the text to display in the alert box. It consists of the information that we want to show to the users.

Let's see some examples of the JavaScript alert() method.

### **Example1**

In this example, there is a simple alert dialog box with a message and an OK button. Here, there is an HTML button which is used for displaying the alert box. We are using the **onclick** attribute and call the **fun()** function where the **alert()** is defined.

<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>

### **Example2**

In this example, there is an alert dialog box with a message and an OK button. Here, we are using the line-breaks in the message of the alert box. The line breaks are defined by using the **'\n'**. The line breaks make the message readable and clear. We have to click the given button to see the effect.

<html>

<head>

<script type = "text/javascript">

function fun() {

alert (" Hello World \n Welcome to the ihubtalent.com \n 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>

### **Example**

In this example, there is an alert dialog box with a message and an OK button. Here, the alert box displays the URL of the corresponding page. The [URL](https://www.javatpoint.com/url-full-form) is defined by using the **alert(location.hostname);** statement.

<html>

<head>

<script type = "text/javascript">

function fun() {

alert(location.hostname);

}

</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.

There are some limitations of using the **eval()** function, such as the **eval()** function is not recommended to use because of the security reasons. It is not suggested to use because it is slower and makes code unreadable.

### **Syntax**

eval(string)

### **Values**

It accepts a single parameter, which is defined as follows.

**string:** It represents a JavaScript expression, single statement, or the sequence of statements. It can be a variable, statement, or a JavaScript expression.

Let's understand the JavaScript **eval()** function by using illustrations.

### **Example1**

It is a simple example of evaluating an expression using the **eval()** function. In this example, there are some variables. We are applying the eval() function on variables a, b, and c to calculate the sum, multiplication, and subtraction.

<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>

</html>

### **Example2**

In this example, we are calling a function using the **eval()** function. Here there is a function **fun()** having two arguments and returns the multiplication of both parameters.

We are calling the function in the **eval()** function, and storing the result in the **res** variable.

<html>

<head>

<script>

var res;

function fun(a, b){

return a \* b;

}

eval("res = fun(50, 50);");

document.write(res);

</script>

</head>

</html>

### **Example3 - Evaluate the string with JavaScript statements**

In this example, we are using the **eval()** function to evaluate the string with JavaScript statements. Here, there is string **str**, having JavaScript conditional **if-else** statement. We are matching the value of the variable **'x'**, if the value of **x** is **0**, the output will be **'SUNDAY'**, else the output will be **'MONDAY'**.

<html>

<head>

<script>

var x = 0;

var str = "if(x == 0) {'SUNDAY'} else 'MONDAY';";

document.write('The output is : ', eval(str));

</script>

</head>

</html>

### **Example - convert string to JavaScript Objects**

In this example, we are converting the string to JSON object. Here, the string **str** contains the data as opposed to the code. We have to switch the data to JSON that allows the string to use a subset of JavaScript syntax for representing the data.

Here, we are using the object **obj** to represent the data.

<html>

<head>

<script>

var str = '({"fname" : "Harry", "lname" : "Rickman"})';

var obj = eval(str);

document.write(obj.fname + " " + obj.lname);

</script>

</head>

</html>

# **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.

### **Syntax**

targetElement.closest(selectors);

In the above syntax, **selectors** is a string containing a selector (like **p:hover**, etc.) used to find a node.

Let's understand this method by using some illustrations.

### **Example1**

In this example, there are three div elements and a heading on which we are applying the **closest()** method. Here, the selectors that we are using are the **id** selector, **descendant** selector, **child** selector, and **:not** selector.

<!DOCTYPE html>

<html>

<body>

<div id = "div1"> This is the first div element.

<h3 id = "h"> This is a heading inside the div. </h3>

<div id = "div2"> This is the div inside the div element.

<div id="div3">This is the div element inside the second div element. </div>

</div>

</div>

<script>

var val1 = document.getElementById("div3");

var o1 = val1.closest("#div1");

var o2 = val1.closest("div div");

var o3 = val1.closest("div > div");

var o4 = val1.closest(":not(#div3)");

console.log(o1);

console.log(o2);

console.log(o3);

console.log(o4);

</script>

</body>

</html>

### **Example2**

This is another example of using [JavaScript](https://www.javatpoint.com/javascript-tutorial)'s **closest()** method.

<!DOCTYPE html>

<html>

<body>

<div id = "div1"> This is the div element.

<p id = "p1"> This is the paragraph element inside the div element.

<h3 id = "h"> This is the child of the paragraph element.

<p id = "p2"> This is the child of heading element of the paragraph element. </p>

</h3>

</p>

</div>

<script>

var val1 = document.getElementById("p2");

var o1 = val1.closest("p");

var o2 = val1.closest("h3");

var o3 = val1.closest("div");

console.log(o1);

console.log(o2);

console.log(o3);

</script>

</body>

</html>

# **JavaScript continue statement**

There is full control to handle loop statements in JavaScript. Sometimes, a situation occurs when we require to skip some code of the loop and move to the next iteration. It can be achieved by using JavaScript's **continue** statement.

The continue statement 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, while loop,** and **do-while loop**. When it is used in **a while** loop, then it jumps back to the condition. If 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 to the conditional expression, and if the condition is true, then the next iteration will be started; otherwise, the control exits the loop.

### **Syntax**

continue;

OR

continue[label];  // Using the label reference

It can be used with or without the label reference. The **label** is an identifier name for a statement. It is optional.

Let's understand the **continue** statement using some examples.

### **Example1**

In this example, we are using the **continue** statement in the **for** loop. Here the iteration of the loop begins with 1 and ends at 7. There is a conditional statement that checks when the iteration reaches at 4. When it is reached to 4, the iteration is skipped due to the **continue** statement and moves to the next iteration.

<!DOCTYPE html>

<html>

<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>

### **Example2**

In this example, we are using the **continue** statement in the **while** loop. Here, we are defining an array **'rainbow'**. The iteration of the loop begins with 0 and ends at the length of the array. There is a conditional statement using the OR (||) operator, which checks when the iteration reaches to the values 'Magenta' and 'Skyblue". When it is reached to the appropriate values, the iteration is skipped due to the continue statement and moves to the next iteration.

<!DOCTYPE html>

<html>

<body>

<h1> Example of the JavaScript Continue Statement </h1>

<h3> You can see that the arrray values "Magenta" and "Skyblue" are skipped. </h3>

<script>

var rainbow = ["Violet", "Indigo", "Magenta", "Blue", "Skyblue", "Green", "Yellow", "Orange", "Red"];

var i = 0;

var res = "";

while(i<rainbow.length){

if (rainbow[i] == "Magenta" || rainbow[i] == "Skyblue") {

i++;

continue;

}

res = "";

res += rainbow[i] + "<br>";

i++;

document.write(res);

}

</script>

</body> </html>

### **Example3**

In this example, we are using a label with the continue statement. There is a nested for loop in which the outer loop is labeled as **'label1'** and the inner loop is labeled as **'label2'**.

<!DOCTYPE html>

<html>

<body>

<p> This is an example of the continue statement with the label </p>

<p id="para"></p>

<script>

var res = "";

var i, j;

label1: // This loop is labeled as "label1"

for (i = 1; i<=5; i++) {

res += "<br>" + "i = " + i + ", j = ";

label2: // This loop is labeled as "label2"

for (j = 1; j <= 4; j++) {

if (j == 2) {

continue label2;

}

document.getElementById("para").innerHTML = res += j + " ";

}

}

</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.

It is different from the **getAttributeNode()** method. The **getAttributeNode()** method returns the attribute as an Attr object.

### **Syntax**

element.getAttribute(attributename)

### **Parameter Values**

**attributename:** It is the required parameter. It is the name of the attribute we want to get the value from.

### **Example1**

In this example, there are two **div** elements with id **div1** and **div2**, each having **style** attribute. We are getting the value of [**style**](https://www.javatpoint.com/html-style) attribute by using the **getAttribute()** method.We have to click the given button to get the value of the **style** attribute of given div elements.

<!DOCTYPE html>

<html>

<body>

<h1>Example of the getAttribute() Method </h1>

<div id = "div1" style = "background-color: yellow; font-size: 25px; color: red; border: 2px solid red;">This is first div element.</div><br>

<div id = "div2" style = "background-color: lightblue; font-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>

### **Example2**

We can also get the value of **onclick** attribute of the button element. In this example, we are extracting the value of **onclick** attribute and the value of **href** attribute. There is an anchor element with the **href** attribute; we are getting this attribute's value using the **getAttribute()** method.

<!DOCTYPE html>

<html>

<body>

<h1>Welcome to the iHub.com</h1>

<h2>Example of the getAttribute() Method</h2>

<div id = "div1" style = "background-color: yellow; font-size: 25px; color: red; border: 2px solid red;">This is the div element.</div><br>

<a href = "http://www.ihubtalent.com/" id = "link">iHubtalent.com </a><br><br>

<button onclick = "fun()" id = "btn">Click me!</button>

<p id = "p"></p>

<p id = "p1"></p>

<script>

function fun() {

var val = document.getElementById("btn").getAttribute("onclick");

document.getElementById("p").innerHTML = val;

var val1 = document.getElementById("link").getAttribute("href");

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 an 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.

### **Syntax**

The general syntax to hide an element using **style.hidden** and **style.visibility** is given as follows.

Using **style.hidden**

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

Using **style.visibility**

document.getElementById("element").style.visibility = "none";

Now, let's see some examples to understand the hiding of elements in javascript.

### **Example1**

In this example, we will see how to remove elements by using JavaScript's **style.display** property. Here, there is a **div** element and a paragraph element that gets hide on clicking the given HTML button. We have to click the **'Click me!'** button to see the effect.

<!DOCTYPE html>

<html>

<body>

<h1>Welcome to the ihubtalent.com</h1>

<h2>Example of the JavaScript's style.display property</h2>

<div id = "div" style = "background-color: yellow; font-size: 25px; color: red; border: 2px 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>

### **Example2**

In this example, we will see how to hide elements by using JavaScript's **style.visibliity** property. Here, a div element and a paragraph element get hidden, but their space is still allocated.

We have to click the **'Click me!'** button to see the effect.

<!DOCTYPE html>

<html>

<body>

<h1>Welcome to the ihubtalent.com</h1>

<h2>Example of the JavaScript's style.visibility property</h2>

<div id = "div" style = "background-color: yellow; font-size: 25px; color: red; border: 2px 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.visibility = "hidden";

document.getElementById("p").style.visibility = "hidden";

}

</script>

</body>

</html>

### **Example3**

In this example, we are using both **style.display** and **style.visibility** JavaScript properties to see the difference between both of them. Here, there is a **div** element and **<h3>** heading element on which we are applying the properties. We are hiding the **div** element by applying the **style.display** property, and hiding the **<h3>** element by applying the **style.visibility** property.

We have to click the **'Click me!'** button to see the effect.

<!DOCTYPE html>

<html>

<body>

<h1>Welcome to the ihubtalent.com</h1>

<h2>Using both style.visibility and style.display properties</h2>

<div id = "div" style = "background-color: yellow; font-size: 25px; color: red; border: 2px solid red;">This is the div element.</div>

<p> This is a paragraph element. </p>

<h3 id = "heading"> This is the heading after the paragraph element. </h3>

<button onclick = "fun()" id = "btn">Click me!</button>

<script>

function fun() {

document.getElementById("div").style.visibility = "hidden";

document.getElementById("heading").style.display = "none";

}

</script>

</body>

</html>

# **JavaScript prompt() dialog 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** prefix. 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".

The prompt box takes the focus and forces the user to read the specified message. So, it should avoid overusing this method because it stops the user from accessing the other parts of the webpage until the box is closed.

### **Syntax**

prompt(message, default)

### **Values**

The parameter values of this function are defined as follows.

**message:** It is an optional parameter. It is the text displays to the user. We can omit this value if we don't require to show anything in the prompt.

**default:** It is also an optional parameter. It is a string that contains the default value displayed in the textbox.Let's see some examples of the JavaScript prompt() method.

### **Example1**

In this example, there is a simple prompt box with a message and two buttons (OK and Cancel). Here, there is an HTML button which is used for displaying the prompt box. We are using the **onclick** attribute and call the **fun()** function where the **prompt()** is defined.

<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>

### **Example2**

It is another example of using the **prompt()** method.

<!DOCTYPE html>

<html>

<head>

<title>JavaScript prompt() method</title>

<script>

function fun() {

var a = prompt("Enter some text", "the iHubTalent.com");

if (a != null) {

document.getElementById("para").innerHTML = "Welcome to " + a;

}

}

</script>

</head>

<body style = "text-align: center;">

<h1 style = "color: red;">Hello World</h1>

<h2>Example of the JavaScript prompt() method</h2>

<button onclick = "fun()">Click me</button>

<p id = "para"></p>

</body>

</html>

### **Example3**

In this example, there is a prompt box with a message and buttons. Here, we are using the line-breaks in the message of the box. The line breaks are defined by using the **'\n'**. The line breaks make the message readable and clear. We have to click the given button to see the effect.

<html>

<head>

<script type = "text/javascript">

function fun() {

prompt(" Hello World \n Welcome to the iHubtalent.com \n This is a prompt 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 removeAttribute() method**

This 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.

### **Syntax**

element.removeAttribute(attributename)

### **Parameter Values**

**attributename:** It is the required parameter that specifies the attribute's name to remove from the element. If the attribute doesn't exist, the method doesn't create any error.

Let us understand it by using some examples.

### **Example1**

In this example, there are two paragraph elements with id **para,** and **para1** belongs to the same class **jtp**. Here, we are removing the **class** attribute of these paragraph elements. We have to click the given HTML button to see the effect.

<!DOCTYPE html>

<html>

<head>

<title>The removeAttribute Method</title>

<style>

.jtp {

color: red;

background-color: yellow;

}

</style>

</head>

<body>

<h1>Welcome to the iHub.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>

### **Example2**

In this example, there are two div elements with id **div1** and **div2**. We are applying the **style** attribute to these div elements.

Here, we are removing the **style** attribute of these div elements. We have to click the given HTML button to see the effect.

<!DOCTYPE html>

<html>

<head>

<title>The removeAttribute Method</title>

<style>

.jtp {

color: red;

background-color: yellow;

}

</style>

</head>

<body>

<h1>Welcome to the iHub.com</h1>

<h2>Example of the removeAttribute() Method</h2>

<div id = "div1" style = "background-color: yellow; font-size: 25px; color: red; border: 2px solid red;">This is first div element.</div><br>

<div id = "div2" style = "background-color: lightblue; font-size: 25px; color: blue; border: 2px solid blue;">This is second div element.</div><br>

<button onclick = "fun()">Click me!</button>

<script>

function fun() {

document.getElementById("div1").removeAttribute("style");

document.getElementById("div2").removeAttribute("style");

}

</script>

</body>

</html>

Similarly, we can use the **removeAttribute()** method to remove the **target** attribute, **align** attribute, **readonly** attribute, and many more.

# **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 values to default. It does not require any parameter values and also does not return any value.

### **Syntax**

formElement.reset()

### **Example**

To illustrate the use of **reset()** method in JavaScript, we are creating a simple HTML document into which we have created a form with the **id = "myForm"**. In this form, there are four text fields: **"First Name", "Last Name", "Age",** and **"E-mail Id"**. There are two buttons that are **"Submit"**, and **"Reset data"**. When we click the **Reset data** button, it calls the function **fun(),** where we have defined the JavaScript's **reset()** method.

In the function **fun()**, we are first taking the reference of the form required to reset, and then we are applying the **reset()** method over it. Now, let's see the code for the same.

<!DOCTYPE html>

<html>

<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 unreachable.

We can return primitive values (such as Boolean, number, string, etc.) and Object types (such as functions, objects, arrays, etc.) 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.

### **Syntax**

return expression;

The **expression** in the above syntax is the value returned to the function caller. It is optional. If the **expression** is not specified, the function returns **undefined**.

It is not allowed to use a line terminator between the **return** keyword and value. We can understand it by using the following lines. Suppose we are writing the **return** statement as follows:

return

x + y;

Then, it will be transformed into -

return;

x + y;

There is the automatic insertion of the semicolon after the **return** statement. The code written after the **return** statement (**x + y;**) will be considered as the **unreachable code**.

We can use parentheses to prevent this problem. It can be written as -

return (

x + y;

);

### **Example1**

This is a simple example of using the **return** statement. Here, we are returning the result of the product of two numbers and returned back the value to the function caller.

The variable **res** is the function caller; it is calling the function **fun()** and passing two integers as the arguments of the function. The result will be stored in the **res** variable. In the output, the value **360** is the product of arguments **12** and **30**.

<!DOCTYPE html>

<html>

<body>

<h3> Example of the JavaScript's return statement </h3>

<script>

var res = fun(12, 30);

function fun(x, y){

return x \* y;

}

document.write(res);

</script>

</body>

</html>

### **Example2**

Here, we are interrupting a function using the **return** statement. The function stops executing immediately when the **return** statement is called.

There is an infinite **while** loop and variable **i,** which is initialized to 1. The loop continues until the value of **i** reached to **4**. When the variable's value will be 4, the loop stops its execution because of the **return** statement. The statement after the loop will never get executed.

Here, the **return** statement is without using the **expression**, so it returns **undefined.**

<!DOCTYPE html>

<html>

<body>

<h2> Welcome to the iHub.com </h2>

<h3> Example of the JavaScript's return statement </h3>

<script>

var x = fun();

function fun() {

var i = 1;

while(i) {

document.write(i + '<br>');

if (i == 4) {

return;

}

document.write(i + '<br>');

i++;

}

}

</script>

</body>

</html>

Now, we will see how to return multiple values using the **return** statement. Usually, the JavaScript functions returns a single value, but we can return multiple values by using the **array** or **object**. To return multiple values, we can pack the values as the object's properties or array elements.

### **Example3 - Returning multiple values using Array**

In this example, we are returning multiple values by using the **Array**. Here, we are using the **ES6 Array destructuring** syntax to unpack the values of array.

<!DOCTYPE html>

<html>

<body>

<h1> Welcome to the iHub.com </h1>

<h3> This is an example of returning multiple values using array </h3>

<script>

function getData() {

let fname = 'John',

lname = 'Rickman',

age = '25',

occupation = 'Private Employee';

return [fname, lname, age, occupation];

}

const [fname, lname, age, occupation] = getData();

document.write("Name = " + fname + " " + lname + "<br>");

document.write("Age = " + age + "<br>");

document.write("Occupation = " + occupation);

</script>

</body>

</html>

### **Example4 - Returning multiple values using object**

In this example, we are returning multiple values by using the **Object**. Here, we are using the **ES6 Object destructuring** syntax to unpack the values of the object.

<!DOCTYPE html>

<html>

<body>

<h1> Welcome to the iHub.com </h1>

<h3> This is an example of returning multiple values using object </h3>

<script>

function getData() {

let fname = 'John',

lname = 'Rickman',

age = '25',

occupation = 'Private Employee';

return {

fname,

lname,

age,

occupation

};

}

let {fname, lname, age, occupation} = getData();

document.write("Name = " + fname + " " + lname + "<br>");

document.write("Age = " + age + "<br>");

document.write("Occupation = " + occupation);

</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. It 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.

### **Syntax**

string.split(separator, limit)

The function arguments are discussed as follows.

**separator:** It is an optional parameter. It can be a regular expression or a simple string. It specifies the point where the split should take place.

If it has multiple characters, then the sequence of the entire character must be found to split.

If the separator is not present in the given string, or if it is omitted, then the entire string becomes a single array element. In these cases, the returned array contains a single element consisting of the entire string.

If the separator is present at the beginning or the end of the string, then it still has the effect of splitting. The returned array consists of an empty string of zero length that appears at the beginning or the last position of the returned array.

**limit:** It is also an optional parameter. It is a non-negative integer that specifies the number of limits. It defines the higher limit on the number of splits to be found in the given string. If it is given, it splits the string at each occurrence of the specified **separator**. It stops when the limit entries have been placed in the array.

An array can contain fewer entries than the given limit. It happens when the end of the string is reached before the limit is reached.

Let's understand the **split()** method using some examples.

### **Example1**

In this example, the **split()** function splits the string **str** wherever the whitespace (" ") occurs and returns an array of strings. Here, we are using the limit argument and providing the value of the **limit** argument to **3**.

<!DOCTYPE html>

<html>

<head>

<script>

var str = 'Welcome to the ihubtalent.com'

var arr = str.split(" ", 3);

document.write(arr);

</script>

</head>

</html>

### **Example2**

In this example, we are using the letter **'t'** as the separator of the given string. The **split()** function will make an array of strings by splitting the given string at each occurrence of the letter **'t'.**

Here, we are not specifying the **limit** argument.

<!DOCTYPE html>

<html>

<head>

<script>

var str = 'Welcome to the ihubtalent.com'

var arr = str.split("t");

document.write(arr);

</script>

</head>

</html>

### **Example3**

In this example, we are omitting the **separator** parameter. In the output, we can see that the returned array contains a single element consists of the given string.

<!DOCTYPE html>

<html>

<head>

<script>

var str = 'Welcome to the iHub.com'

var arr = str.split();

document.write(arr);

</script>

</head>

</html>

### **Example4**

Here, we are defining the **limit** parameter and using the letter **'o'** as the separator of the given string. The **split()** function will create an array of strings by splitting the given string wherever the given letter **'o'** occurs until the limit is reached.

The limit parameter limits the number of splits to 2 because we are defining **2** as the value of the limit. In the output, we can see that the splitting only happens twice.

<!DOCTYPE html>

<html>

<head>

<script>

var str = 'Welcome to the iHub.com'

var arr = str.split("o", 2);

document.write(arr);

</script>

</head>

</html>

# **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.

### **Syntax**

There are following two ways of using the typeof operator.

typeof operand

or

typeof (operand)

### **Values**

**operand:** It is an expression that represents the object or primitive whose type is to be returned.

The possible return 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" |

Let's understand this operator by using some examples.

### **Example1**

In this example, the operands are of number type. The **typeof** operator will print the **"number"** as the type of the operand, whether the operand is a negative interger, floating-point number, infinity, NaN, zero, or any integer.

<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>

</html>

### **Example2**

In this example, the operands are of string type. The **typeof** operator will print the **"string"** as the type of the operand, whether the operand is an empty string, collection of characters, number written in quotes.

<html>

<head>

<script>

document.write(typeof ("") + "<br>"); // results: "string"

document.write(typeof ("ihubtalent.com") + "<br>"); // results: "string"

document.write(typeof ("20") + "<br>"); // results: "string"

document.write(typeof (typeof 1) + "<br>"); // results: "string"

document.write(typeof String(12)); // wrapping in String function will results: "string"

</script>

</head>

</html>

### **Example3**

In this example, the operands are of Boolean type. The **typeof** operator will print the **"boolean",** as the type of the operand, if the operand is **true**, or **false**.

<html>

<head>

<script>

document.write(typeof (true) + "<br>"); // results: "boolean"

document.write(typeof (false) + "<br>"); // results: "boolean"

document.write(typeof Boolean(20) + "<br>"); // wrapping in Boolean function will results: "boolean"

</script>

</head>

</html>

### **Example4**

In this example, the operands are of undefined type. The **typeof** operator will print the **"undefined"** as the type of the operand. Here, the type of **Null** is **undefined**, it is because it is written as **Null** instead of **null**. If we write it as **null**, the type of it will be **object.**

<html>

<head>

<script>

document.write(typeof Null + "<br>"); // results: "undefined"

document.write(typeof undefined + "<br>"); // results: "undefined"

document.write(typeof a + "<br>"); // results: "undefined"

</script>

</head>

</html>

### **Example5**

In this example, the operands are of **Object** and **Function** type. The **typeof** operator will print the **"object"** and **"function",** according to the type of the operand.

<html>

<head>

<script>

document.write(typeof null + "<br>"); // results: "object"

document.write(typeof [1, 2, 'hello'] + "<br>"); // results: "object"

document.write(typeof {a: 'hello'} + "<br>"); // results: "object"

document.write(typeof [1, 2, 3, 4] + "<br>"); // results: "object"

document.write(typeof function(){} + "<br>"); // results: "function"

document.write(typeof class hello{} + "<br>"); // results: "function"

</script>

</head>

</html>

# **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.

It is the only operator in JavaScript that takes three operands. The working of this operator is the same as the **if-else** conditional statement. We can say that it is the shortcut of the **if-else**.

This operator includes three operands: a condition followed by a question mark (?) sign, and two expressions separated by the colon (:). The first expression is executed when the condition is true, and the second expression is executed when the condition is false.

### **Syntax**

var a = (condition) ? expr1 : expr2;

In the above syntax, **condition, expr1,** and **expr2** are the three operands used in the ternary operator. The value is assigned to the variable **'a'** based on the provided condition. The condition is evaluated as a Boolean value; based on its result, the operator assigns the result to the variable. It assigns the first expression on the true value of the condition and assigns the second expression on the false value of the condition.

Let's see an example of using the **ternary** operator in JavaScript.

### **Example**

This is a simple example of checking whether the number is odd or even using the ternary operator. The result will display using the **alert()** dialog box.

<!DOCTYPE html>

<html>

<head>

<script>

let a = 358;

let val = ( a % 2 == 0) ? 'Even Number' : 'Odd Number';

alert(val);

</script>

</head>

<body>

<h1> Welcome to the iHub.com </h1>

<h3> This is an example of ternary operator. </h3>

</body>

</html>

# **JavaScript reload() method**

In JavaScript, 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.

### **Syntax**

location.reload()

This method can have optional parameters **true** and **false**. The **true** keyword force to reload the page from the server, while the **false** keyword reloads the page from the cache.

The **false** is the default parameter of this method, so if we omitted the parameter's value, the **reload()** method reloads the page from the cache. It means that the **object.reload()** is same as the **object.reload(false)**.

Let's see an example of using the **location.reload()** method.

### **Example**

Here, the function **fun()** contains the **location.reload()** method. We are calling the function **fun()** using the **onclick** attribute of the **button** element. So, we have to click the given HTML **'Reload'** button to see the effect. After clicking the button, the page will reload.

<!DOCTYPE html>

<html>

<head>

<title>location.reload() method</title>

<script>

function fun() {

location.reload();

}

</script>

</head>

<body>

<h1> Welcome to the iHub.com </h1>

<h2> This is 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>

In the above example, instead of calling the **fun()** function, we can also attach the **location.reload()** method to the button markup. It can be done as given below:

**<button** onclick = "location.reload()"**>** Reload **</button>**

Using the above syntax, we do not require creating any [JavaScript](https://www.javatpoint.com/javascript-tutorial) function to reload the page.

# **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.

Although we can add the **style** attribute using the **setAttribute()** method, but it is recommended not to use this method for styling. For adding styles, we can use the properties of the style object that will effectively change the style. It can be clear with the following code.

**Incorrect way**

It is recommended not to use it to change the style.

element.setAttribute("style", "background-color: blue;");

**Correct way**

The correct way to change the style is given below.

element.setAttribute.backgroundColor = "blue";

To get the value of an attribute, we can use the **getAttribute()** method, and to remove a specific attribute from an element, we can use the **removeAtrribute()** method.

If we are adding a Boolean attribute such as **disabled**, then whatever the value it has, it is always considered as **true**. If we require to set the value of the Boolean attribute to **false**, we have to remove the entire attribute using the **removeAttribute()** method.

### **Syntax**

element.setAttribute(attributeName, attributeValue)

The arguments of this method are not optional. Both parameters must be included when using this method. The parameter values of this method are defined as follows.

### **Parameter Values**

**attributeName:** It is the name of the attribute that we want to add to an element. It cannot be left empty; i.e., it is not optional.

**attributeValue:** It is the value of the attribute that we are adding to an element. It is also not an optional value.

Let's understand how to use **setAttribute()** method by using some illustrations.

### **Example1**

In this example, we are adding a **href** attribute with a value of **"https://www.iHubTalent.com/"** to the **<a>** tag with **id = "link"**.

<html>

<head>

<title> JavaScript setAttribute() method </title>

<script>

function fun() {

document.getElementById("link").setAttribute("href", "https://www.iHubTalent.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">ihubtalent.com </a>

<p> Click the follwing button to see the effect. </p>

<button onclick = "fun()"> Add attribute </button>

</body>

</html>

### **Example2**

In this example we are updating the value of a existing attribute using the **setAttribute()** method. Here, we are converting a textfield to a button by changing the value of **type** attribute from **text** to **button**.

We have to click the specified button to see the effect.

<html>

<head>

<title> JavaScript setAttribute() method </title>

<script>

function fun() {

document.getElementById("change").setAttribute("type", "button");

}

</script>

</head>

<body style = "text-align: center;">

<h2> It is an example to update an attribute's value using the setAttribute() method. </h2>

<input id = "change" type = "text" value = "iHub"/>

<p> Click the follwing button to see the effect. </p>

<button onclick = "fun()"> Change </button>

</body>

</html>

### **Example3**

Here, we are adding a Boolean attribute **disabled** to disable the specified button. If we set the value of the **disabled** attribute to an empty string, then it is automatically sets to true which causes the button to be disabled.

<html>

<head>

<title> JavaScript setAttribute() method </title>

<script>

function fun() {

document.getElementById("btn").setAttribute("disabled", "");

}

</script>

</head>

<body style = "text-align: center;">

<h2> Example of the setAttribute() method. </h2>

<p> Click the following button to see the effect </p>

<button onclick = "fun()" id = "btn"> Click me </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 **clearInterval()** method is called. This method returns a numeric value or a non-zero number that identifies the created timer.

Unlike the **setTimeout()** method, the **setInterval()** method invokes the function multiple times. This method can be written with or without the **window** prefix.

The commonly used syntax of **setInterval()** method is given below:

### **Syntax**

window.setInterval(function, milliseconds);

### **Parameter values**

This method takes two parameter values **function** and **milliseconds** that are defined as follows.

**function:** It is the function containing the block of code that will be executed.

**milliseconds:** This parameter represents the length of the time interval between each execution. The interval is in milliseconds. It defines how often the code will be executed. If its value is less than 10, the value 10 is used.

## How to stop the execution?

We can use the **clearInterval()** method to stop the execution of the function specified in **setInterval()** method. The value returned by the **setInterval()** method can be used as the argument of **clearInterval()** method to cancel the timeout.

Let's understand the use of **setInterval()** method by using some illustrations.

### **Example1**

This is a simple example of using the **setInterval()** method. Here, an alert dialog box displays at an interval of 3 seconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

**<html>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setInterval() 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 iHubTalent.com ");

}

**</script>**

**</body>**

**</html>**

### **Example2**

Here, the background color will change on every 200 milliseconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

<html>

<body>

<h1> Hello World :) :) </h1>

<h3> This is an example of using the setInterval() method </h3>

<p> Here, the background color changes on every 200 milliseconds. </p>

<script>

var var1 = setInterval(color, 200);

function color() {

var var2 = document.body;

var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";

}

</script>

</body>

</html>

### **Example3**

In the above example, we have not used any method to stop the toggling between the colors. Here, we are using the **clearInterval()** method to stop the toggling of colors in the previous example.

We have to click the specified **stop** button to see the effect.

<html>

<body>

<h1> Hello World :) :) </h1>

<h3> This is an example of using the setInterval() method </h3>

<p> Here, the background color changes on every 200 milliseconds. </p>

<button onclick = "stop()"> Stop </button>

<script>

var var1 = setInterval(color, 200);

function color() {

var var2 = document.body;

var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";

}

function stop() {

clearInterval(var1);

}

</script>

</body>

</html>

The color of the background will start changing after 200 milliseconds. On clicking the specified **stop** button, the toggling between the colors will be stopped on the corresponding background color.

# **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.

Unlike the **setInterval()** method, the **setTimeout()** method executes the function only once. This method can be written with or without the **window** prefix.

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.

The commonly used syntax of the **setTimeout()** method is given below.

### **Syntax**

window.setTimeout(function, milliseconds);

### **Parameter values**

This method takes two parameter values **function** and **milliseconds** that are defined as follows.

**function:** It is the function containing the block of code that will be executed.

**milliseconds:** This parameter represents the time-interval after which the execution of the function takes place. The interval is in milliseconds. Its default value is 0. It defines how often the code will be executed. If it is not specified, the value **0** is used.

Let's understand the use of **setTimeout()** method by using some illustrations.

### **Example1**

This is a simple example of using the **setTimeout()** method. Here, an alert dialog box will display at an interval of two seconds. We are not using any method to prevent the execution of the function specified in **setTimeout()** method. So the **setTimeout()** method executes the specified function only once, after the given time interval.

<html>

<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 iHubtalent.com ");

}

</script>

</body>

</html>

### **Example2**

It is another example of using the **setTimeout()** method. Here, a new tab opens after a time interval of two seconds and gets close after two seconds of opening. We are using the **window.open()** method to open a new tab and **window.close()** method to close the opened tab.

Because we are not using any method to prevent the execution of the function specified in **setTimeout()** method. So the function gets execute only once, after the given time interval.

<html>

<body>

<h1> Hello World :) :) </h1>

<h3> This is an example of using the setTimeout() method </h3>

<p> Here, a new tab will open after 2 seconds and close after 2 seconds. </p>

<script>

var a = setTimeout(fun1, 2000);

function fun1(){

var win1 = window.open();

win1.document.write(" <h2> Welcome to the iHubtalent.com </h2>");

setTimeout(function(){win1.close()}, 2000);

}

</script>

</body>

</html>

### **Example3**

In the above examples, we have not used any method to prevent the execution of function specified in **setTimeout()**. Here, we are using the **clearTimeout()** method to stop the function's execution.

We have to click the given **stop** button before two seconds to see the effect.

<html>

<body>

<h3> This is an example of using the setTimeout() method </h3>

<p> Click the following button before 2 seconds to see the effect. </p>

<button onclick = "stop()"> Stop </button>

<script>

var a = setTimeout(fun1, 2000);

function fun1(){

var win1 = window.open();

win1.document.write(" <h2> Welcome to the iHubTalent.com </h2>");

setTimeout(function(){win1.close()}, 2000);

}

function stop() {

clearTimeout(a);

}

</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.

### **Syntax**

The following syntax represents the **includes()** method:

string.includes(searchValue, start);

### **Parameter values**

The parameter values of this method are defined as follows:

**searchValue:** It is a required parameter. It is the substring to search for.

**start:** It is an optional parameter. It represents the position where to start the searching in the string. Its default value is 0. When it is omitted, the search will begin from the initial position of the string, i.e., from **0**.

Let's understand the **includes()** method using some examples.

### **Example1**

It is a simple example to determine whether the given string contains the specified substring. Here, we are declaring a variable **str** and assigning a string value **'Welcome to the iHubtalent.com'** to it. Then we use the **includes()** method for determining whether the given substring ('**to**') is present or not.

Here, we are not defining the position to start the search. So, the search will begin from the initial position of the string.

<!DOCTYPE html>

<html>

<body>

<h3>Example of using the JavaScript's string includes() method. </h3>

<script>

let str = "Welcome to the iHubTalent.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>

### **Example2**

In this example, we are determining whether the **includes()** method is case-sensitive or not. The given string is **'Welcome to the iHubTalent.com'**. We are searching for the substring **'TO'** in the given string.

Although the word **'to'** is present in the given string, but the method is case-sensitive so, it will return the Boolean value **false**.

<!DOCTYPE html>

<html>

<body>

<h3> Example of using the JavaScript?s string includes() method. </h3>

<p> Here, we are searching for the substring <b> 'TO' </b> in the given string. </p>

<script>

let str = "Welcome to the iHubTalent.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>

### **Example3**

In this example, we are defining the position to start the search. So, the searching will start from the specified position.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<h3>** Example of using the JavaScript string includes() method. **</h3>**

**<script>**

let str = "Welcome to the iHubTalent.com";

document.write(" **<b>** The given string is: **</b>**", str);

document.write("**<br>**");

let res = str.includes('the', 10);

document.write(" **<b>** The result of str.includes('the', 10) is : **</b>** ", res);

**</script>**

**</body>**

**</html>**

# **Calculate current week number in JavaScript**

Sometimes we need to calculate the current week number or the week number for a given date. This problem can be solved using the JavaScript programming language. JavaScript offers several date functions, such as getDays(), getMonth(), getTime(), to solve date-related tasks. Along with that, the math functions Math.floor() and Math.ceil() also help to calculate the week number.

**Situation**

In this approach, we will assign a number to each day of the week, e.g., 1 for Sunday, 2 for Monday, 3 for Tuesday, and so on. Similarly, we will assign the number to other days in the week. Note that weekday starts with Sunday and ends with Saturday.

Let's suppose today is Monday and the week number is 1. So, if we calculate the week number after 25 days, the week number will be 4th.

According to the week number calculation:

25 days = 7 + 7 + 7 + 4 days

The following formula mentioned below is used to calculate the week number after p days:

Week Number = least integer [ p / 7 ]

We can use this concept in our JavaScript example to calculate the week number. We will calculate the week number in two ways:

* Calculate the week number of the current date
* Calculate the week number for a predefined date

We will discuss both the methods in detail with examples. Along with that, we will calculate the weekdays dynamically by taking date input from the user using a dynamic [HTML](https://www.javatpoint.com/html-tutorial) form.

* Calculate the week number by taking date input dynamic HTML form

### **Approach 1: Calculate Week Number of current date**

In this approach, we will find the week number of current date means the date will be taken from the system. It is a static way to calculate the week number. Follow each step for calculating weekdays:

1. Define a date variable **todaydate** and **oneJan** in JavaScript.
2. Initialize the todaydate variable by creating the date object using **new Date()**, which will take system date by default.
3. Initialize another variable oneJan by creating date object using new Date() but this time **getFullYear()** method inside it.
4. The **getFullYear()** function will return the year of current date along with first date of the year, e.g., 1 Jan 2020.
5. After getting the current date and year, now calculate the number of days using the **Math.floor()** method. So, calculate the difference of current date and current year date and divide it by total milliseconds in a day (1000\*60\*60\*24).
6. Now, add 1 and numberofdays calculated in previous result to the day of current date and divide them by 7, i.e., (this.getDay() + 1 + numberofdays) / 7. Don't forget to put this whole calculation of step 6 inside **Math.ceil()** method.
7. Finally, display the calculated weekday on the web using **document.write()**.

Now, we will convert these steps into actual implementation. See the code below:

<html>

<body>

<script>

//define a date object variable that will take the current system date

todaydate = new Date();

//find the year of the current date

var oneJan= new Date(todaydate.getFullYear(), 0, 1);

// calculating number of days in given year before a given date

var numberOfDays= Math.floor((todaydate - oneJan) / (24 \* 60 \* 60 \* 1000));

// adding 1 since to current date and returns value starting from 0

var result = Math.ceil(( todaydate.getDay() + 1 + numberOfDays) / 7);

//display the calculated result

document.write("Week Numbers of current date (" + todaydate +

") is: <br>" + result);

</script>

</body>

</html>

### **Approach 2: Calculate the week number for a predefined date**

This approach will help to calculate the week number by providing a date in code. Follow the below step:

1. Define a date variable in JavaScript, i.e., date1
2. Initialize it by creating the date objects using **new Date()** and provide date inside it.
3. After defining the date, calculate the time difference between them by subtracting one date from another date using **date2.getTime() - date1.getTime();**
4. Now next step is to calculate the days between the dates. So, we will divide the time difference of both dates by the milliseconds in a day, i.e., 1000\*60\*60\*24
5. At last, print the result calculated using the **document.write()** method.

Now, we will convert these steps into actual implementation. See the code below:

<html>

<body>

<script>

//define a date object variable with date inside it

var date1 = new Date("11/27/2019");

//find the year of the entered date

var oneJan= new Date(date1.getFullYear(), 0, 1);

// calculating number of days in given year before the given date

var numberOfDays= Math.floor((date1 - oneJan) / (24 \* 60 \* 60 \* 1000));

// adding 1 since to current date and returns value starting from 0

var result = Math.ceil(( date1.getDay() + 1 + numberOfDays) / 7);

//display the calculated result

document.write("Week Numbers of date (" + date1 +

") is: <br>" + result);

</script>

</body>

</html>

## Calculate Week Number by entering a date using HTML form

In this approach, we will find the week number by providing a date using a dynamic HTML form. The user can choose the date from the calendar and enter to input field in HTML form. It is a dynamic way for calculating the week number in which a user can provide the input by itself on the web instead of providing the dates input in code by the programmer.

<html>

<head>

<title> Calculate week number by user input </title>

<script>

function weekNumber() {

//define a variable and fetch the input from HTML form

var dateinput = document.getElementById("dateInput1").value;

//create a date object variable to store the date values

var date1 = new Date(dateinput);

//find the year of the current date

var oneJan= new Date(date1.getFullYear(), 0, 1);

// calculating number of days in given year before the given date

var numberOfDays= Math.floor((date1 - oneJan) / (24 \* 60 \* 60 \* 1000));

// adding 1 since to current date and returns value starting from 0

var result = Math.ceil(( date1.getDay() + 1 + numberOfDays) / 7);

//return the calculated result and display it

return document.getElementById("result").innerHTML = "Week number of given date is: " + result;

}

</script>

</head>

<body>

<h2 style="color: 32A80F" align="Center">iHubTalent: Calculate Week Number using user input <br><br></h2>

<p align="Center"><b> Enter date   </b>

<input type="date" id="dateInput1"><br><br>

<button onclick="weekNumber()">

Calculate Week Number

</button></p>

<h3 style="color:32A80F" id="result" align="center"></h3>

</body>

</html>

# **Calculate days between two dates in JavaScript**

Sometimes we need to calculate the number of days between two dates. This can be done using the JavaScript programming language. JavaScript provides a [math function](https://www.javatpoint.com/javascript-math) [Math.floor() method](https://www.javatpoint.com/javascript-math-floor-method) to calculate the days between two dates. In JavaScript, we have to use the date object for any calculation to define the date using **new Date()**.

To learn more about JavaScript in detail step by step follow our [JavaScript tutorial](https://www.javatpoint.com/javascript-tutorial).

We will calculate the number of days in two ways:

* Calculate days by entering two dates
* Calculate days from current dates

We will discuss both the methods in detail with examples. Along with that, we will calculate the days remaining in Christmas day from today (date will be taken from the system).

* Find the number of days remaining in Christmas from the current date

### **Approach 1: Calculate the days by entering two dates**

This approach will help to calculate the days between two dates by providing two different dates in code. Follow the below step:

1. Define two date variables in JavaScript
2. Initialize them by creating the date objects using **new Date()**
3. After defining the dates, calculate the time difference between them by subtracting one date from another date using **date2.getTime() - date1.getTime();**
4. Now next step is to calculate the days between the dates. So, we will divide the time difference of both dates by the milliseconds in a day, i.e., 1000\*60\*60\*24
5. At last, print the result calculated using the **document.write()** method.

Now, we will convert these steps into actual implementation. See the code below:

<html>

<head>

<title>Calculate days by entering two dates</title>

</head>

<body>

<script>

var date1, date2;

//define two date object variables with dates inside it

date1 = new Date("07/15/2015");

date2 = new Date("11/28/2016");

//calculate time difference

var time\_difference = date2.getTime() - date1.getTime();

//calculate days difference by dividing total milliseconds in a day

var days\_difference = time\_difference / (1000 \* 60 \* 60 \* 24);

document.write("Number of days between dates <br>" +

date1 + " and <br>" + date2 + " are: <br>"

+ days\_difference + " days");

</script>

</body>

</html>

## Calculate days using dynamic HTML form

It is a dynamic JavaScript example in which a user can provide the input by itself on web instead of providing the dates input in code by the programmer. The user will dynamically choose the dates from the calendar and enter the two dates in input field and calculate the number of days between two dates by clicking on a button. See the code below:

<html>

<head>

<title> Calculate days between dates </title>

<script>

function daysDifference() {

//define two variables and fetch the input from HTML form

var dateI1 = document.getElementById("dateInput1").value;

var dateI2 = document.getElementById("dateInput2").value;

//define two date object variables to store the date values

var date1 = new Date(dateI1);

var date2 = new Date(dateI2);

//calculate time difference

var time\_difference = date2.getTime() - date1.getTime();

//calculate days difference by dividing total milliseconds in a day

var result = time\_difference / (1000 \* 60 \* 60 \* 24);

return document.getElementById("result").innerHTML =

result + " days between both dates. ";

}

</script>

</head>

<body>

<h2 style="color: 32A80F" align="Center">

ihub : Calculate days between dates

<br><br></h2>

<p align="Center"><b> Enter date1   </b>

<input type="date" id="dateInput1">

<br><br>

<p align="Center"><b> Enter date2   </b>

<input type="date" id="dateInput2">

<br><br>

<button onclick="daysDifference()">

Calculate number of days

</button>

</p>

<h3 style="color:32A80F" id="result" align="center"></h3>

</body>

</html>

### **Approach 2: Calculate the days from current date**

In this approach, we will use the math function Math.floor() to calculate the days. Follow the below step:

1. Define two date variables, i.e., date1 and date2
2. Initialize the date1 variables by creating the date objects using **new Date()**, which will take system date by default.
3. Initialize the date2 variables by creating the date objects using **new Date()** and provide a date in it.
4. Now use the Math.abs() method to calculate the total seconds between two dates. So, divide the difference of dates by the milliseconds in one second **Math.abs(date2-date1) / 1000;**
5. Now next step is to calculate the number of days using the **Math.floor()** method. So, divide the calculated result from previous steps (total seconds between two dates) by 60\*60\*24
6. At last, print the result calculated using the **document.write()** method.

Now, we will convert these steps into actual implementation. See the code below:

<html>

<head>

<title>Calculate days between the dates using default system dates</title>

</head>

<body>

<script>

var date1, date2;

//define two date object variables with dates inside it

date1 = new Date();

date2 = new Date ("Dec 15, 2021, 21:45:10");

//calculate total number of seconds between two dates

var total\_seconds = Math.abs(date2 - date1) / 1000;

//calculate days difference by dividing total seconds in a day

var days\_difference = Math.floor (total\_seconds / (60 \* 60 \* 24));

document.write("Number of days between dates from current date <br>" +

date1 + " and <br>" + date2 + " are: <br>"

+ days\_difference + " days");

</script>

</body>

</html>

### **Example 3: Calculate the days remaining in Christmas day from the current date**

We have one more example in which we will calculate the days till Christmas Day means for a specific date. Follow the below steps:

1. Define the current date (system date) by creating a date object using the **new Date()**.
2. Define one more date using new Date() class method and fetch the Christmas date by **date.getFullYear()** method in it. The date.getFullYear() method will return the year (0-11 months in JavaScript).
3. In case Christmas has been already passed away this year, it will return the number of days remaining in Christmas of next year from the current date.
4. Now, its time to use Math.round() function to calculate the result in milliseconds and then convert this result into days. So, divide the **Math.round(Christmas() - present\_date.getTime())** by the total milliseconds in a day, i.e., 1000\*60\*60\*24.
5. Remove the decimal value from the result and display it to the web using **document.write()** method.

Now, convert these steps into actual implement to calculate the days remaining in Christmas. The implementation is as easy as the steps are. See the below code:

<html>

<head>

<title>Calculate days remaining in Christmas</title>

</head>

<body>

<script>

//declare two variables in JavaScript

var current\_date, christmas\_day;

//total milliseconds in one day

var one\_day\_ms = 1000 \* 60 \* 60 \* 24;

//set the current date in the variable

current\_date = new Date();

//set the Christmas date in another variable where 0-11 is month in JavaScript

christmas\_day= new Date(current\_date.getFullYear(), 11, 25);

//calculate next year Christmas if this year?s date is already passed away

if (current\_date.getMonth() == 11 &&current\_date.getdate() > 25)

{

christmas\_day.setFullYear(christmas\_day.getFullYear() + 1);

}

//calculate result in milliseconds and convert it into days

var res = Math.round(christmas\_day.getTime() - current\_date.getTime()) / (one\_day\_ms);

//remove the decimal point from the calculated result

var FinalResult = res.toFixed (0);

//display the final result on web

document.write("Number of days remaining in Christmas day: <br>" +

current\_date + " and <br>" + christmas\_day + " are: <br>" +FinalResult + " days");

</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, i.e., 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 with which the trim() method will be used.

#### **Note - The trim() method does not change the original string; it just removes the leading and trailing whitespace character.**

### **Syntax**

The syntax of trim() method is as follows:

str.trim()

Here, str is a String class of object that will contain the string to be trimmed.

### **Parameters**

The trim() method does not have any arguments or parameters inside it.

### **Return Value**

The string.trim() function returns the string after removing whitespace character from that string from the beginning and end of the string.

### **Examples**

Below are some examples, which are using trim() function to remove the elements from it. In these examples, you will see how to use this JavaScript function. So, here are a few examples -

**Example 1**

In this example, we will pass a string containing whitespace at both ends.

<html>

<body>

<script>

function func\_trim() {

//original string with whitespace in beginning and end

var str = " iHub tutorial website ";

//string trimmed using trim()

var trimmedstr = str.trim();

document.write(trimmedstr);

}

func\_trim();

</script>

</body>

</html>

**Example 2**

In this example, we will pass a string containing whitespace only at the end.

<html>

<body>

<script>

function func\_trim() {

//original string with whitespace at the end

var str = "iHub tutorial website ";

//string trimmed using trim()

var trimmedstr = str.trim();

document.write(trimmedstr);

}

func\_trim();

</script>

</body>

</html>

**Example 3**

In this example, we will pass a string containing whitespace only in the beginning.

<html>

<body>

<script>

function func\_trim() {

//original string with whitespace in beginning

var str = " iHub tutorial website";

//string trimmed using trim()

var trimmedstr = str.trim();

document.write(trimmedstr);

}

func\_trim();

</script>

</body>

</html>

JavaScript offers two more functions similar to the trim() method. They also used to trim the string but only from the one end, either left or right. Remember that both the functions do not change the original string; they just remove the whitespaces.

* trimLeft()
* trimRight()

### **trimLeft()**

The **trimLeft()** method removes the whitespace only from the left of the string. To understand better, we can say that it removes whitespace only from the start and returns the string back without any leading whitespace character.

**Syntax**

str.trimLeft()

**Example 1**

<html>

<body>

<script>

function func\_trim() {

//original string with whitespace in beginning

var str = " iHub tutorial website ";

//string trimmed using trimLeft()

var trimmedstr = str.trimLeft();

document.write(trimmedstr);

}

func\_trim();

</script>

</body>

</html>

### **trimRight()**

On the other hand, the **trimRight()** method removes the whitespace only from the right of the string. To understand better, we can say that it removes whitespace only from the end and returns the string back without any trailing whitespace character.

**Syntax**

str.trimRight()

**Example 1**

<html>

<body>

<script>

function func\_trim() {

//original string with whitespace from the end

var str = " iHub tutorial website ";

//string trimmed using trimRight()

var trimmedstr = str.trimRight();

document.write(trimmedstr);

}

func\_trim();

</script>

</body>

</html>

# **JavaScript timer**

In JavaScript, a timer is created to execute a task or any function at a particular time. Basically, the timer is used to delay the execution of the program or to execute the JavaScript code in a regular time interval. With the help of timer, we can delay the execution of the code. So, the code does not complete it's execution at the same time when an event triggers or page loads.

The best example of the timer is advertisement banners on websites, which change after every 2-3 seconds. These advertising banners are changed at a regular interval on the websites like Amazon. We set a time interval to change them. In this chapter, we will show you how to create a timer.

JavaScript offers two timer functions **setInterval()** and **setTimeout()**, which helps to delay in execution of code and also allows to perform one or more operations repeatedly. We will discuss both the timer functions in detail as well as their examples.

## Examples

Below are some examples of JavaScript timer using these functions.

### **setTimeout()**

The setTimeout() function helps the users to delay the execution of code. The setTimeout() method accepts two parameters in which one is a user-defined function, and another is the time parameter to delay the execution. The time parameter holds the time in milliseconds (1 second = 1000 milliseconds), which is optional to pass.

The basic syntax of setTimeout() is:

setTimeout(function, milliseconds)

We will use the setTimeout() function to delay the printing of message for 3 seconds. The message will display on the web after 3 seconds of code execution rather than immediately. Now, let's look at the code below to see how it works:

**Execution of code after a delay**

<html>

<body>

<script>

function delayFunction() {

//display the message on web after 3 seconds on calling delayFunction

document.write('<h3> Welcome to iHub<h3>');

}

</script>

<h4> Example of delay the execution of function <h4>

<!?button for calling of user-defined delayFunction having 3 seconds of delay -->

<button onclick = "setTimeout(delayFunction, 3000)"> Click Here </button>

</body>

</html>

### **setInterval()**

The setInterval method is a bit similar to the setTimeout() function. It executes the specified function repeatedly after a time interval. Or we can simply say that a function is executed repeatedly after a specific amount of time provided by the user in this function. **For example -** Display updated time in every five seconds.

The basic syntax of setInterval() is:

setInterval(function, milliseconds)

Similar to setTimeout() method, it also accepts two parameters in which one is a user-defined function, and another is a time-interval parameter to wait before executing the function. The time-interval parameter holds the amount of time in milliseconds (1 second = 1000 milliseconds), which is optional to pass. Now, see the code below how this function works:

**Execution of code at a regular interval**

<html>

<body>

<script>

function waitAndshow() {

//define a date and time variable

var systemdate = new Date();

//display the updated time after every 4 seconds

document.getElementById("clock").innerHTML = systemdate.toLocaleTimeString();

}

//define time interval and call user-defined waitAndshow function

setInterval(waitAndshow, 4000);

</script>

<h3> Updated time will show in every 4 seconds </h3>

<h3> The current time on your computer is: <br>

<span id="clock"></span></h3>

</body>

</html>

### **Example**

One more example of setInterval() methods for displaying a message string after every 4 seconds continuously.

<html>

<body>

<script>

function waitAndshow() {

//display the message on web on calling delayFunction

document.write('<h3> Welcome to iHub<h3>');

}

</script>

<h3> A string will show in every 4 seconds </h3>

<!-- call user-defined delayFunction after 4 seconds -->

<button onclick = "setInterval(waitAndshow, 4000)"> Click Here </button>

</body>

</html>

You have seen how to create timer or set time interval. Sometimes, we need to cancel these timers. JavaScript offers the in-built function to clear the timer, which are as follows:

## Cancel or Stop the timer

JavaScript offers two functions **clearTimeout()** and **clearInterval()** to cancel or stop the timer and halt the execution of code. The setTimeout() and setInterval() both return a unique IDs. These IDs are used by the clearTimeout() and clearInterval() to clear the timer and stop the code execution beforehand. They both take only one parameter, i.e., ID.

**Example**

In this example, we will use clearTimeout() to clear the timer set by with setTimeout() function. Look at the example how clearInterval() work with setInterval().

### **Disable the regular interval**

<html>

<body>

<script>

function waitAndshow() {

//define a date and time variable

var systemdate = new Date();

//display the updated time after every 4 seconds

document.getElementById("clock").innerHTML = systemdate.toLocaleTimeString();

}

//function to disable the time interval

function stopClock() {

clearInterval(intervalID);

}

//define time interval and call user-defined waitAndshow function

var intervalID = setInterval(waitAndshow, 3000);

</script>

<p>Current system time: <span id="clock"></span></p>

<!-- button to stop showing time in a regular interval -->

<button onclick = "stopClock();" > Stop Clock </button>

</body>

</html>

# **Remove elements from array in JavaScript**

An array is a variable used to store one or more elements of the same data type. Basically, it stores multiple elements of the same type. Sometimes we need to remove these elements from an array. JavaScript offers several built-in array methods to add or remove the elements from an array easily. Using these methods, you can remove an element from start, end, or as well as from a specific index.

These JavaScript array methods are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **pop()** | This method removes the elements from the end of the array. |
| **shift()** | Like the pop() method, it also removes the elements but from the start of the array. |
| **filter()** | The filter() method removes the elements from an array in a programmatically way. |
| **splice()** | This method removes the elements from a specific index. |

All the above methods are array functions offered by JavaScript. These methods are discussed below in detail with examples.

## Remove elements from the end of the array - pop()

JavaScript provides the pop() method to remove the elements from the end of the array. It removes the last element of the array and returns the removed element. When an element removes from the array, the length of the array is reduced by 1. See the code and output below to understand:

### **Example 1**

<html>

<body>

<script>

function removeLastElement() {

var shoeBrand = ["Nike", " Adidas", " Sparks", " RedTape"];

document.write("Elements in array before removing: <br>" + shoeBrand + "<br><br>");

// Removing last element from the array

var poppedElement = shoeBrand.pop();

document.write("Removed element from array: " + poppedElement + "<br><br>");

//display remaining elements present in array after removing

document.write("Elements present in array: <br>" + shoeBrand);

}

removeLastElement();

</script>

</body>

</html>

### **Example 2**

By putting the above code in a loop (for, while, or do-while), we can delete all elements one by one from the end of the array. See how it will work:

<html>

<body>

<script>

function removeElement() {

var shoeBrand = ["Nike", " Adidas", " Sparks", " RedTape"];

//initial length of the array

document.write("Elements in array before removing: <br>" + shoeBrand + "<br><br>");

document.write("Array length before removing elements is:" + shoeBrand.length + "<br><br>");

while (shoeBrand.length) {

//store removed element in a variable

var poppedElement = shoeBrand.pop();

//display removed element

document.write("Removed element from array: " + poppedElement + " <br>");

}

//Length of the array after removing all elements

document.write("<br> Array length after removing elements is:" + shoeBrand.length);

}

removeElement();

</script>

</body>

</html>

## Remove elements from the start of the array - shift()

JavaScript provides the shift() method, which is used to remove the element from the start of the array. It removes the first element from an array and returns the removed element. When an element removes from the array, the array length is reduced by 1. See the code and output below how this function works:

### **Example 1**

<html>

<body>

<script>

function removeFirstElement() {

var shoeBrand = ["Nike", " Adidas", " Sparks", " RedTape"];

document.write("Elements in array before removing: <br>" + shoeBrand + "<br><br>");

// Removing first element from the array

var poppedElement = shoeBrand.shift();

document.write("Removed element from array: " + poppedElement + "<br><br>");

//display remaining elements present in array after removing

document.write("Elements present in array: <br>" + shoeBrand);

}

removeFirstElement();

</script>

</body>

</html>

### **Example 2**

Like the pop() method, we can delete all elements one by one from the start of the array by putting the above code in a loop (for, while, or do-while). In this example, we will put this code in a while loop. See how it will work:

<html>

<body>

<script>

function removeElement() {

var shoeBrand = ["Nike", " Adidas", " Sparks", " RedTape"];

//initial length of the array

document.write("Elements in array before removing: <br>" + shoeBrand + "<br><br>");

document.write("Array length before removing elements is:" + shoeBrand.length + "<br><br>");

while (shoeBrand.length) {

//store removed element in a variable

var poppedElement = shoeBrand.shift();

//display removed element

document.write("Removed element from array: " + poppedElement + " <br>");

}

//Length of the array after removing all elements

document.write("<br> Array length after removing elements is:" + shoeBrand.length);

}

removeElement();

</script>

</body>

</html>

## Remove elements from a specific index in an array - splice()

To remove the element from a specific index position, the splice() method is used. It removes the element from a specific position and returns that removed element. It also allows the users to remove one or more elements from the array.

The splice() method accepts mainly two arguments: initial index position and number of items to be removed. Array index count starts from 0, i.e., a[0]. When the elements remove from an array, the array length is reduced. See the below example and its output how the splice() function works:

### **Example 1**

In this example, we will delete three elements, starts from index 1, i.e., a[1] to a[3].

<html>

<body>

<script>

function removeElement() {

var shoeBrand = ["Nike", " Adidas", " Sparks", " RedTape", " Bata"];

document.write("Elements in array before removing: <br>" + shoeBrand + "<br><br>");

// Removing first element from the array

var poppedElement = shoeBrand.splice(1, 3);

document.write("Removed element from array: " + poppedElement + "<br><br>");

//display remaining elements present in array after removing

document.write("Elements present in array: <br>" + shoeBrand);

}

removeElement();

</script>

</body>

</html>

### **Example 2**

In this example, we will put the above code inside a for loop to remove all occurrences of a specific element from the array. It will trace the whole array and remove the matching element one by one from the array.

<html>

<body>

<script>

function removeElement() {

var clothingBrand = ["Gucci", " Chanel", "Gucci", " Zara"];

// for loop to trace the whole array

for (var i = 0; i<clothingBrand.length; i++) {

//Match the specific element in array

if (clothingBrand[i] === "Gucci") {

//remove the matched element from array

var delEle = clothingBrand.splice(i, 1);

document.write("<br> Removed element: " + delEle);

document.write("<br> Remaining elements: " + clothingBrand);

document.write("<br>"); }

}

}

removeElement();

</script>

</body>

</html>

You can even remove all elements from the array. See the below code:

**<script>**

    var clothingBrand = ["Gucci", " Chanel", " Calvin Klein", " Zara"];

    document.write("Elements in array: " + clothingBrand);

    //remove all elements

    clothingBrand.splice(0, clothingBrand.length);

    document.write("**<br>** Remaining elements: " + clothingBrand);

**</script>**

## Remove elements from the array using filter()

This method basically removes the element based on the given condition provided by the user. It removes the elements and creates a new array of remaining elements. See the code and output below how it works:

### **Example 1**

In this example, we will check the even-odd values in an array and filter them. The filter() method will check for the even values and return to add them to the modified array. The odd values will remove from the array, and only modified array will be displayed.

<html>

<body>

<script>

function isEven( value ) {

if(value%2 == 0)

return value;

}

//initialize the array named ary

var ary = [43, 243, 56, 24, 1021, 348].filter( isEven );

document.write("Even elements in array: " + ary);

</script>

</body>

</html>

### **Remove elements using delete operator**

Apart from all these functions, JavaScript offers a **delete** operator. It helps to remove the element from a specific index position in an array. This operator is used with array name and index number, which you want to remove, e.g., **delete arrayname[3]**. It returns true after successfully removing an element.

The **delete** operator helps to remove specific index element directly from the array. Now, with the help of an example, let us see how this **delete** operator works:

### **Example**

<html>

<body>

<script>

//declare and initialize an array

var clothingBrand = ["Gucci", " Calvin Klein", " Chanel", " Zara"];

document.write("Elements in array: " + clothingBrand);

//delete element of index 1 from clothingBrand array

var result = delete clothingBrand[1];

//if returned value is true, element is deleted successfully

document.write("<br> Removed successfully: " + result + "<br>");

document.write("Remaining elements in array: " + clothingBrand);

</script>

</body>

</html>

### **Remove elements using clear and reset operator**

JavaScript provides the **clear** and **reset** operator to remove the elements from the array. Usually, they do not delete the array elements; they just shift them to another array and clear the original array.

Now, with the help of an example, let us see how it works:

### **Example 1**

<html>

<body>

<script>

//declare and initialize an array

var originalArray = ["Gucci", " Calvin Klein", " Chanel", " Zara"];

document.write("Initially elements in array: " + originalArray);

//declare one more array to keep the elements of original array

var newArray = originalArray

//clear the initially declared array

originalArray = [ ]

//display element of original and new array after removing

document.write("<br><br> Array after removing elements: " + originalArray);

document.write("<br><br> Elements in new array: " + newArray);

</script>

</body>

</html>

### **Example 2**

Other than this, we can remove all elements of the array by setting its length to 0. See the example below:

<html>

<body>

<script>

//declare and initialize an array

var array1 = ["Gucci", " Calvin Klein", " Chanel", " Zara"];

document.write("Initially elements in array: " + array1);

//set length of array to 0

array1.length = 0;

//display element of original and new array after removing

document.write("<br><br> Array after removing elements: " + array1);

</script>

</body>

</html>

# **JavaScript localStorage**

**LocalStorage** is a data storage type of web storage. This allows the JavaScript sites and apps to store and access the data without any expiration date. This means that the data will always be persisted and will not expire. So, data stored in the browser will be available even after closing the browser window.

In short, all we can say is that the localStorage holds the data with no expiry date, which is available to the user even after closing the browser window. It is useful in various ways, such as remembering the shopping cart data or user login on any website.

In the past days, cookies were the only option to remember this type of temporary and local information, but now we have localStorage as well. Local storage comes with a higher storage limit than cookies (5MB vs 4MB). It also does not get sent with every HTTP request. So, it is a better choice now for client-side storage. Some essential points of localStorage need to be noted:

* localStorage is not secure to store sensitive data and can be accessed using any code. So, it is quite insecure.
* It is an advantage of localStorage over cookies that it can store more data than cookies. You can store 5MB data on the browser using localStorage.
* localStorage stores the information only on browser instead in database. Thereby the localStorage is not a substitute for a server-based database.
* localStorage is synchronous, which means that each operation executes one after another.

## localStorage Methods

The localStorage offers some methods to use it. We will discuss all these localStorage methods with examples. Before that, a basic overview of these methods are as follows:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| setItem() | This method is used to add the data through key and value to localStorage. |
| getItem() | It is used to fetch or retrieve the value from the storage using the key. |
| removeItem() | It removes an item from storage by using the key. |
| clear() | It is used to gets clear all the storage. |

Each of these methods is used with localStorage keyword connecting with dot(.) character. **For Example:** localStorage.setItem().

Remember that localStorage property is read-only.

Following some codes given, which are used to add, retrieve, remove, and clear the data in localStorage. Use them in your code accordingly whenever needed. You need a key-value pair to add some data in localStorage. So, let key is city and its value is Noida, **i.e.,** key: value = city: Noida.

**Add data**

To add the data in localStorage, both key and value are required to pass in setItem() function.

localStorage.setItem("city", "Noida");

**Retrieve data**

It requires only the key to retrieve the data from storage and a [JavaScript](https://www.javatpoint.com/javascript-tutorial) variable to store the returned data.

const res = localStorage.getItem("city");

**Remove data**

It also requires only the key to remove the value attached with it.

localStorage.removeItem("city");

**Clear localStorage**

It is a simple clear() function of localStorage, which is used to remove all the localStorage data:

localStorage.clear()

## Limitation of localStorage

As the localStorage allows to store temporary, local data, which remains even after closing the browser window, but it also has few limitations. Below are some limitations of localStorage are given:

* Do not store sensitive information like username and password in localStorage.
* localStorage has no data protection and can be accessed using any code. So, it is quite insecure.
* You can store only maximum 5MB data on the browser using localStorage.
* localStorage stores the information only on browser not in server-based database.
* localStorage is synchronous, which means that each operation executes one after another.

## Advantage of localStorage

The localStorage has come with several advantages. First and essential advantage of localStorage is that it can store temporary but useful data in the browser, which remains even after the browser window closed. Below is a list of some advantages:

* The data collected by localStorage is stored in the browser. You can store 5 MB data in the browser.
* There is no expiry date of data stored by localStorage.
* You can remove all the localStorage item by a single line code, i.e., **clear()**.
* The localStorage data persist even after closing the browser window, like items in shopping cart.
* It also has advantages over cookies because it can store more data than cookies.

### **Browser compatibility**

The localStorage has specified in HTML 5, which is supported by several browsers, like Chrome. Below is a list of different browsers and their versions that supports JavaScript localStorage.

## JavaScript code to check browser compatibility

With the help of below code example, you can check the browser compatibility. Use this code in your every localStorage program to check the browser compatibility before adding or deleting something from localStorage.

**<script>**

// Code to check browser support

if (typeof(Storage) !== "undefined") {

    //browser support localStorage

} else {

   //browser does not support localStorage

}

**</script>**

### **Example**

It is a basic example of adding a key and value to localStorage and retrieving back by the key. See the code below how localStorage methods work:

<html>

<body>

<div id="result"></div>

<script>

// Check browser support

if (typeof(Storage) !== "undefined") {

// Store an item to localStorage

localStorage.setItem("firstname", "Alen");

// Retrieve the added item

document.getElementById("result").innerHTML = localStorage.getItem("firstname");

} else {

//display this message if browser does not support localStorage

document.getElementById("result").innerHTML = "Sorry, your browser does not support Web Storage.";

}

</script>

</body>

</html>

## More Examples

It is an example to count the button clicks means that it will count how many times a user clicks the button. In this example, we will create two buttons, one for counting the user clicks and another for clear the that clicks data.

<html>

<head>

<script>

//function to count the button clicks

function clickCounting() {

if(typeof(Storage) !== "undefined") {

if (localStorage.clickcount) {

localStorage.clickcount = Number(localStorage.clickcount)+1;

} else {

localStorage.clickcount = 1;

}

document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s).";

}

//when the browser does not support

else {

document.getElementById("result").innerHTML = "Your browser does not support web storage.";

}

}

//function to clear the data stored by browser

function clearCounting() {

window.localStorage.clear();

}

</script>

</head>

<body>

<h3> Click the button to see the counter increase.</h3>

<p><button onclick="clickCounting()" type="button">Click to Count</button></p>

<div id="result"></div>

<h4> Now close the browser tab or browser window and execute the code again on the browser. <h4>

<h3>Note: The counter will start counting from where you leave and is not reset.</h3>

<p><button onclick="clearCounting()" type="button">Clear Count</button></p>

</body>

</html>

Clear all records

Clear() method of localStorage is used to clear the entire storage data. When this method invokes, it clears all the records for that domain from the storage. It does not contain any parameters. See the syntax to clear the localStorage:

window.localStorage.clear();

Or

localStorage.clear();

We will use this clear code in below example.

Check localStorage

On the JavaScript console, you can check what is in local storage by typing **localStorage** command on it. Even if there nothing in localStorage, it has length = 0 inside it.

***Command***

LocalStorage

# **JavaScript offsetHeight**

The **offsetHeight** is an HTML DOM property, which is used by JavaScript programming language. It returns the visible height of an element in pixels that includes the height of visible content, border, padding, and scrollbar if present. The offsetHeight is often used with offsetWidth property. The **offsetWidth** is one more property of HTML DOM, which is almost same as the offsetHeight. These properties are used by JavaScript to find the visible height and width of the HTML elements.

The offsetHeight is a combination of following HTML elements:

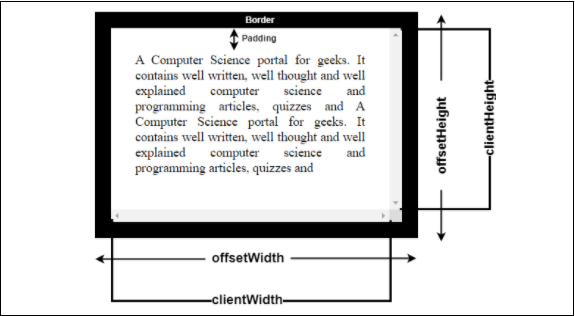
offsetHeight = height + border + padding + horizontal scrollbar

On the other hand, the offsetWidth includes the following elements:

offsetWidth = width + border + padding + vertical scrollbar

Remember one thing that offsetHeight and offsetWidth do not include margin, neither top margin nor bottom margin. These DOM properties are used by JavaScript programming language to calculate the dimension of HTML elements in pixels.

With the help of below diagram you can understand offsetHeight and offsetWidth much better:



### **Syntax**

Below is a simple syntax of offsetHeight:

element.offsetHeight

Here, element is a variable created in JavaScript to hold the [CSS](https://www.javatpoint.com/css-tutorial) properties values or HTML text paragraph.

### **Return Values**

The offsetHeight and offsetWidth return the calculated height and width of the HTML elements in pixels, respectively.

### **Examples**

Below is a list of some examples. With the help of which we will see how offsetHeight property is used and works.

### **Example 1**

**<html>**

**<head**

**<style>**

#JTP {

height: 120px;

width: 250px;

margin: 20px;

padding: 15px;

background-color: yellow;  }

**</style>**

**</head>**

**<script>**

function getInfo() {

var eleValue = document.getElementById("JTP");

var txt = "Height of the elements paragraph along with padding and border in pixels is: " + eleValue.offsetHeight + "px";

document.getElementById("sudo").innerHTML = txt;

}

**</script>**

**<body>**

**<h2>** HTML DOM offsetHeight Property example **</h2>**

**<div** id= "JTP"**>**

**<b>** A basic information about this div tab: **</b>**

**<p** id= "sudo"**>** **</p>**

**</div>**

**<button** type="JTP" onclick="getInfo()"**>** Submit **</button>**

**</body>**

**</html>**

### **Example 2**

In this example, we will calculate the offsetHeight for a paragraph provided in this example along with CSS styling. Remember that the offsetHeight will not include margin.

**<html>**

**<head>**

**<style>**

#PStyle {

height: 220px;

width: 320px;

margin: 20px;

padding: 15px;

background-color: pink;  }

**</style>**

**</head>**

**<script>**

function getInfo() {

var eleValue = document.getElementById("PStyle");

var txt = "Height of the elements paragraph along with padding and border in pixels is: " + eleValue.offsetHeight + "px";

document.getElementById("sudo").innerHTML = txt;

}

**</script>**

**<body>**

**<h3>** HTML DOM offsetHeight Property Example 2 **</h3>**

**<div** id= "PStyle"**>**

**<p>** In this example, we will calculate the offset height for this paragraph. We have also provided CSS styling to this paragraph. This offsetHeight will include the height of text, padding, border except margin taken by this paragraph. **</p>**

**<b>** OffsetHeight of this div tab paragraph: **</b>**

**<p** id= "sudo"**>** **</p>**

**</div>**

**<button** type= "button" onclick = "getInfo()"**>** Calculate offsetHeight **</button>**

**</body>**

**</html>**

### **Example 3 without CSS styling**

See another example of calculating the offsetHeight. We have not included any CSS style like height, width, margin, padding, etc., expect background color. So, the paragraph will be a simple paragraph with no styling.

**<html>**

**<head>**

**<title>**  HTML DOM offsetHeight Property example  **</title>**

**<style>**

#PStyle {

background-color: orange;

}

**</style>**

**</head>**

**<script>**

function getInfo() {

var eleValue = document.getElementById("PStyle");

var txt = "Height of the elements in paragraph calculated as pixels is: " + eleValue.offsetHeight + "px";

document.getElementById("sudo").innerHTML = txt;

}

**</script>**

**<body>**

**<h3>** HTML DOM offsetHeight Property Example 3 **</h3>**

**<div** id= "PStyle"**>**

**<p>** In this example, we will calculate the offset height of this given paragraph. We have jusr included background color in CSS styling not height, width, margin, or padding to this paragraph. So, the offsetHeight will be calculted for the height of text taken by this paragraph. **</p>**

**<b>** OffsetHeight of this div tab paragraph: **</b>**

**<p** id= "sudo"**>** **</p>**

**</div>**

**<button** type= "submit" onclick = "getInfo()"**>** Calculate offsetHeight **</button>**

**</body>**

**</html>**

### **Calculate both offsetHeight and offsetWidth**

In this example, we will calculate both **offsetHeight** and **offsetWidth** for a paragraph inside a div tab. So, you can understand how differently they calculated. Here, we will use CSS and pass the height, width, margin, padding, etc. for styling in this example.

**<html>**

**<head>**

**<title>**HTML DOM offsetHeight Property example  **</title>**

**<style>**

#PStyle {

height: 180px; width: 400px;  margin: 20px;  padding: 15px; background-color: lightblue;

}

**</style>**

**</head>**

**<script>**

function getInfo() {

var eleValue = document.getElementById("PStyle");

var txt1 = "OffsetHeight of the paragraph along with padding and border in pixels is: " + eleValue.offsetHeight + "px";

var txt2 = "OffsetWidth of the paragraph along with padding and border in pixels is: " + eleValue.offsetWidth + "px";

document.getElementById("sudo1").innerHTML = txt1;

document.getElementById("sudo2").innerHTML = txt2;

}

**</script>**

**<body>**

**<h2>** Calculation of offsetHeight and offsetWidth **</h2>**

**<div** id= "PStyle"**>**

**<b>** A basic information about this div tab: **</b>**

**<p** id= "sudo1"**>** **</p>**

**<p** id= "sudo2"**>** **</p>**

**</div>**

**<button** type="button" onclick="getInfo()"**>** Submit **</button>**

**</body>**

**</html>**

# **Confirm password validation in JavaScript**

In this chapter, we will discuss password validation using JavaScript. We need to validate a password every time whenever a user creates an account on any website or app. So, we have to verify a valid password as well as put the confirm password validation. For a valid password, the following parameters must be contained by it to be valid -

* A password should be alphanumeric.
* First letter of the password should be capital.
* Password must contain a special character (@, $, !, &, etc).
* Password length must be greater than 8 characters.
* One of the most important that the password fields should not be empty.

Whenever a user creates a password, there is always one more field of confirm password. It checks that the password entered by the user is same as this confirm password fields. To create a valid password, both the password and confirm password fields value must be matched.

First one, we will check for a valid password and then confirm password validation checks.

## Valid password Validation

In this example, we will check that the password created by the user is valid or not and match with all the parameter discussed above. See the code below for password verification.

<html>

<head><title> Verification of valid Password </title></head>

<script>

function verifyPassword() {

var pw = document.getElementById("pswd").value;

//check empty password field

if(pw == "") {

document.getElementById("message").innerHTML = "\*\*Fill the password please!";

return false;

}

//minimum password length validation

if(pw.length< 8) {

document.getElementById("message").innerHTML = "\*\*Password length must be atleast 8 characters";

return false;

}

//maximum length of password validation

if(pw.length> 15) {

document.getElementById("message").innerHTML = "\*\*Password length must not exceed 15 characters";

return false;

} else {

alert("Password is correct");

}

}

</script>

<body>

<center>

<h1 style="color:green">iHub</h1>

<h3> Verify valid password Example </h3>

<form onsubmit ="return verifyPassword()">

<!-- Enter Password -->

<td> Enter Password </td>

<input type = "password" id = "pswd" value = "">

<span id = "message" style="color:red"></span><br><br>

<!-- Click to verify valid password -->

<input type = "submit" value = "Submit">

<!-- Click to reset fields -->

<button type = "reset" value = "Reset" >Reset</button>

</form>

</center>

</body>

</html>

## Confirm Password Validation

In this example, we will validate the password by verifying both the password entered by the user are same. This process will be done at the client site using JavaScript before the form loading.

<html>

<head><title> Password Matching Validation </title></head>

<script>

function matchPassword() {

var pw1 = document.getElementById("pswd1");

var pw2 = document.getElementById("pswd2");

if(pw1 != pw2)

{

alert("Passwords did not match");

} else {

alert("Password created successfully");

}

}

</script>

<body>

<center>

<form>

<h1 style="color:green">iHub</h1>

<h3> Confirm password Validation Example </h3>

<td> Enter Password </td>

<input type = "password" name = "pswd1"><br><br>

<td> Confirm Password </td>

<input type = "password" name = "pswd2"><br><br>

<!?Click to validate confirm password -->

<button type = "submit" onclick="matchPassword()">Submit</button>

<button type = "reset" value = "Reset" >Reset</button>

</form>

</center>

</body>

</html>

A complete form with password validation

In the above examples, you have learned to verify a valid password and confirm password validation. Now, we will keep both the validations in a single form to complete the password validation process.

For this, we will create a simple basic signup form that will contain some fields, such as first name, last name, create password, and confirm password. The fields with a star (\*) are required fields in which the user must have to provide some value. We will put the following validation in this form to validate a password:

* Empty field validation
* Minimum password length validation, i.e., > 8
* Maximum password length validation, i.e., <15
* Confirm password validation

Apart from that, we have also put a **Reset** button to clear the field's data in the form. When you click on this **reset** button, all the data provided by the user in fields will get clear. Now, see the code below:

<html>

<script>

function validateForm() {

//collect form data in JavaScript variables

var pw1 = document.getElementById("pswd1").value;

var pw2 = document.getElementById("pswd2").value;

var name1 = document.getElementById("fname").value;

var name2 = document.getElementById("lname").value;

//check empty first name field

if(name1 == "") {

document.getElementById("blankMsg").innerHTML = "\*\*Fill the first name";

return false;

}

//character data validation

if(!isNaN(name1)){

document.getElementById("blankMsg").innerHTML = "\*\*Only characters are allowed";

return false;

}

//character data validation

if(!isNaN(name2)){

document.getElementById("charMsg").innerHTML = "\*\*Only characters are allowed";

return false;

}

//check empty password field

if(pw1 == "") {

document.getElementById("message1").innerHTML = "\*\*Fill the password please!";

return false;

}

//check empty confirm password field

if(pw2 == "") {

document.getElementById("message2").innerHTML = "\*\*Enter the password please!";

return false;

}

//minimum password length validation

if(pw1.length < 8) {

document.getElementById("message1").innerHTML = "\*\*Password length must be atleast 8 characters";

return false;

}

//maximum length of password validation

if(pw1.length > 15) {

document.getElementById("message1").innerHTML = "\*\*Password length must not exceed 15 characters";

return false;

}

if(pw1 != pw2) {

document.getElementById("message2").innerHTML = "\*\*Passwords are not same";

return false;

} else {

alert ("Your password created successfully");

document.write("JavaScript form has been submitted successfully");

}

}

</script>

<body>

<h1 style="color:green">iHub</h1>

<h3> Verify valid password Example </h3>

<form onsubmit ="return validateForm()">

<!-- Enter first name -->

<td> Full Name\* </td>

<input type = "text" id = "fname" value = "">

<span id = "blankMsg" style="color:red"></span><br><br>

<!-- Enter last name -->

<td> Last Name </td>

<input type = "text" id = "lname" value = "">

<span id = "charMsg" style="color:red"></span><br><br>

<!-- Create a new password -->

<td> Create Password\* </td>

<input type = "password" id = "pswd1" value = "">

<span id = "message1" style="color:red"></span><br><br>

<!?Enter confirm password -->

<td> Confirm Password\* </td>

<input type = "password" id = "pswd2" value = "">

<span id = "message2" style="color:red"></span><br><br>

<!-- Click to verify valid password -->

<input type = "submit" value = "Submit">

<!-- Click to reset fields -->

<button type = "reset" value = "Reset" >Reset</button>

</form>

</body>

</html>

# **Difference Between Static and Const in JavaScript**

We make use of both static and const variables in different languages. In this section, we will describe the difference points between both variables. Let's discuss.

### **What is a Static variable in JavaScript**

A static variable is a class property that is used in a class and not on the instance of the class. The variable is stored on the data segment area of the memory, and the same value is shared among every instance created in a class. To use a static variable, we use the static keyword. We can use the static keyword for making a static value, a static function, with classes, operators, properties and work as a utility function for the application or websites. The value of a static variable is set at the run time and is a kind of global value that can be used for the instance of the specified class.

### **What is a Const variable in JavaScript**

A const variable is a variable that has a fixed value and remains the same throughout the program. A property of the const variable is that we cannot change or modify its value throughout the program. It is because as soon as we make a const value, the compiler gets informed that the value is fixed and should be prevented from the programmer. Thus, whenever the programmer tries to modify a const value, an error gets displayed that the const value cannot be changed. To use a const variable, we use a 'const' keyword and input value.

## Static vs. Const in JavaScript

There are the following difference points which will let us understand the difference between the two:

|  |  |
| --- | --- |
| **Static** | **Const** |
| The static keyword is used for defining static properties and methods in a javascript class program. | The const keyword is used for defining constant value for a variable. |
| The static keyword can be accessed on the class definition only. In order to access the static keyword for non-static methods, one needs to invoke them using the class name. However, for calling a static method within another static method, we can make use of this keyword. | A const value can be accessed globally or locally, but a global constant can never be window object properties. |
| The static methods are the utility functions that are used for creating or cloning the objects. | The const variable is used for declaring a constant or fixed value whose value cannot be changed. |
| JavaScript static is labeled by a keyword known as the 'static' keyword. | JavaScript const is labeled by a keyword known as the 'const' keyword, where we declare a const variable and initialize it with a constant value. |
| JavaScript static can be used with classes and methods also. | JavaScript const can be used with objects and arrays also. |
| The value for a static variable can be reassigned. | The value for a const variable cannot be reassigned. However, we can re-declare the const variable in different block scope as it is allowed. |

Above are some difference points that will make us understand the working of both JavaScript keywords. Apart from these theoretical difference descriptions, let's have a look over an example of both through which we can understand the use and working of the static and const variable.

### **Using JavaScript Static**

Below is a practical implementation of using JavaScript static keyword within a class:

**<html>**

**<body>**

**<script>**

               class A {

                static staticMethod() {

                 return "Calling Static method.";

                }

               }

   document.write(A.staticMethod());

**</script>**

**</body>**

**</html>**

In the above program code, we can see that a static method is created within a class and when it is invoked, it calls the statement within the static method of the class.

### **Using Javascript const**

Below is a practical implementation of using JavaScript const keyword:

**<html>**

**<body>**

**<script>**

const value= 8;

 try {

         value= 10;

}

  catch (e) {

   document.write(e);

} //will display a TypeError

document.write(value);

**</script>**

**</body>**

**</html>**

# **How to Convert Comma Separated String into an Array in JavaScript**

A comma-separated valuation (CSV) file is a demarcated file format that uses a comma to isolate the values. The data record is comprised of one or more than one field, separated by the commas. The name root for this file format is the comma-separated file format, is the use of the comma as a field extractor.

You can convert the comma-separated string into an array by using the two following methods.

* **By using the split () method**
* **Append through the list and put track of each comma you find to generate a new sequence with different strings.**

## By using the split () method

The split () method is used to partition a sequence based on an extractor. This separator can be delineated as a comma to differentiate the string if a comma is encountered. This procedure returns an array of distinct strings.

**Syntax**

string.split(' , ')

**Example**

<!DOCTYPE html**>**

**<html**

**<body>**

**<h2** style="color: green"**>**        iHub  **</h2>**

**<b>**Conversion of comma separated string to array in JavaScript**</b>**

**<p>**Original string is  "Twenty, Thirty, Fourty, Fifty, Sixty"**</p>**

**<p>** Values of the Separated Array :**<span** class="output"**></span></p>**

**<button** onclick="separateString()"**>**    Remove Text   **</button>**

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

        function separateString() {

            originalString = "Twenty, Thirty, Fourty, Fifty, Sixty";

            separatedArray = originalString.split(', ');

            console.log(separatedArray);

            document.querySelector('.output').textContent =

            separatedArray;

}

**</script>**

**</body>**

**</html>**

## Append through the list and put track of each comma you find to generate a new sequence with different strings

This method helps you to iterate the characters of the string and analyze for the comma. The Previous index variable can be determined, which keeps records of the first character of the next string. The slice method is used to remove the part of the string between the prior index and the specific position of the comma found. This string is driven to a new array. This procedure is iterated for the entire length of the string. The final segment comprises all the separate strings.

**Syntax**

originalString = " Twenty, Thirty, Fourty, Fifty, Sixty ";

separatedArray = [];

let previousIndex = 0;                       // index of end of the last string

for(i = 0; i **<** **originalString.length**; i++)

{

  if (originalString[i] == ', ') {  // check the character for a comma

       // split the string from the last index to the comma

        separated = originalString.slice(previousIndex, i);

        separatedArray.push(separated);

      previousIndex = i + 1;              // update the index of the last string

    }

}

             // push the last string into the array

separatedArray.push(originalString.slice(previousIndex, i));

**Example**

<!DOCTYPE html**>**

**<html>**

**<body>**

**<h1** style="color: green"**>**iHub**</h1>**

**<b>**Conversion of comma separated stringto array in JavaScript**</b>**

**<p>**Original string is "Twenty, Thirty, Fourty, Fifty, Sixty"**</p>**

**<p>**Separated Array is: **<span** class="output"**></span>** **</p>**

**<button** onclick="separateString()"**>** Remove Text**</button>**

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

         function separateString() {

             originalString = "Twenty, Thirty, Fourty, Fifty, Sixty";

             separatedArray = [];

             let previousIndex = 0;       // index of end of the last string

             for (i = 0; i **<** **originalString.length**; i++){

        if (originalString[i] == ', '){       // check the character for a comma

separated =        // split the string from the last index to the comma

originalString.slice(previousIndex, i);

separatedArray.push(separated);

previousIndex = i + 1;                 // update the index of the last string

                 }

             }

        separatedArray.push(          // push the last string into the array

        originalString.slice(previousIndex, i));

        console.log(separatedArray);

        document.querySelector(

        '.output').textContent = separatedArray;

}

**</script>**

**</body>**

**</html>**

# **Calculate age using JavaScript**

JavaScript offers some built-in date and time functions, which helps to calculate the age from the date (Date of Birth). Using these JavaScript methods, you can easily find the age of anyone. For this, we require a date input from the user and the current system date. We need to track the following conditions in mind while calculating the difference between them:

* If the **current date** is less than the date (birthdate) entered by the user, that month will not be counted as the month is not completed. Otherwise, we will add the number of month days (30 or 31) to the current date to get the difference between them.
* If the **current month** is less than the birth month, the current year will not be counted. To get the month's difference, we will subtract by adding the total number of months (12) to the current month.
* Finally, we just need to subtract the date, month and year after satisfying the above two conditions.

Now, we will convert this process to actual implementation.

There are various ways to calculate the age from date of birth. We will discuss simple and easily understandable methods to calculate the age using JavaScript.

### **Example 1: Predefined date input**

<script>

var dob = new Date("06/24/2008");

//calculate month difference from current date in time

var month\_diff = Date.now() - dob.getTime();

//convert the calculated difference in date format

var age\_dt = new Date(month\_diff);

//extract year from date

var year = age\_dt.getUTCFullYear();

//now calculate the age of the user

var age = Math.abs(year - 1970);

//display the calculated age

document.write("Age of the date entered: " + age + " years");

</script>

### **Example 2: dynamic date input**

In this example, we will create an HTML form to take the date input from the user and then calculate the age using JavaScript. It will take dynamic input from the user. This HTML form will use the calendar to choose date input.

<html>

<head>

<script>

function ageCalculator() {

var userinput = document.getElementById("DOB").value;

var dob = new Date(userinput);

if(userinput==null || userinput=='') {

document.getElementById("message").innerHTML = "\*\*Choose a date please!";

return false;

} else {

//calculate month difference from current date in time

var month\_diff = Date.now() - dob.getTime();

//convert the calculated difference in date format

var age\_dt = new Date(month\_diff);

//extract year from date

var year = age\_dt.getUTCFullYear();

//now calculate the age of the user

var age = Math.abs(year - 1970);

//display the calculated age

return document.getElementById("result").innerHTML =

"Age is: " + age + " years. ";

}

}

</script>

</head>

<body>

<center>

<h2 style="color: 32A80F" align="center"> Calculate Age from DOB</h2>

<!-- Choose a date and enter in input field -->

<b> Enter Date of Birth: <input type=date id = DOB></b>

<span id = "message" style="color:red"></span><br><br>

<button type="submit" onclick="ageCalculator()">Calculate age </button><br><br>

<h3 style="color:32A80F" id="result" align="center"></h3>

</center>

</body>

</html>

### **Example 2: Calculate age in year, month and days**

This example will calculate and display the age in year, month, and days rather than only in years. **E.g.,** for a DOB 27 Dec 2015, the person will be 4 years, 9 months, and 23 days old.

<html>

<head>

<script>

function ageCalculator() {

//collect input from HTML form and convert into date format

var userinput = document.getElementById("DOB").value;

var dob = new Date(userinput);

//check user provide input or not

if(userinput==null || userinput==''){

document.getElementById("message").innerHTML = "\*\*Choose a date please!";

return false;

}

//execute if the user entered a date

else {

//extract the year, month, and date from user date input

var dobYear = dob.getYear();

var dobMonth = dob.getMonth();

var dobDate = dob.getDate();

//get the current date from the system

var now = new Date();

//extract the year, month, and date from current date

var currentYear = now.getYear();

var currentMonth = now.getMonth();

var currentDate = now.getDate();

//declare a variable to collect the age in year, month, and days

var age = {};

var ageString = "";

//get years

yearAge = currentYear - dobYear;

//get months

if (currentMonth>= dobMonth)

//get months when current month is greater

var monthAge = currentMonth - dobMonth;

else {

yearAge--;

var monthAge = 12 + currentMonth - dobMonth;

}

//get days

if (currentDate>= dobDate)

//get days when the current date is greater

var dateAge = currentDate - dobDate;

else {

monthAge--;

var dateAge = 31 + currentDate - dobDate;

if (monthAge< 0) {

monthAge = 11;

yearAge--;

}

}

//group the age in a single variable

age = {

years: yearAge,

months: monthAge,

days: dateAge

};

if ( (age.years> 0) && (age.months> 0) && (age.days> 0) )

ageString = age.years + " years, " + age.months + " months, and " + age.days + " days old.";

else if ( (age.years == 0) && (age.months == 0) && (age.days> 0) )

ageString = "Only " + age.days + " days old!";

//when current month and date is same as birth date and month

else if ( (age.years> 0) && (age.months == 0) && (age.days == 0) )

ageString = age.years + " years old. Happy Birthday!!";

else if ( (age.years> 0) && (age.months> 0) && (age.days == 0) )

ageString = age.years + " years and " + age.months + " months old.";

else if ( (age.years == 0) && (age.months> 0) && (age.days> 0) )

ageString = age.months + " months and " + age.days + " days old.";

else if ( (age.years> 0) && (age.months == 0) && (age.days> 0) )

ageString = age.years + " years, and" + age.days + " days old.";

else if ( (age.years == 0) && (age.months> 0) && (age.days == 0) )

ageString = age.months + " months old.";

//when current date is same as dob(date of birth)

else ageString = "Welcome to Earth! <br> It's first day on Earth!";

//display the calculated age

return document.getElementById("result").innerHTML = ageString;

}

}

</script>

</head>

<body>

<center>

<h2 style="color: 32A80F" align="center"> Calculate Age from Date of Birth <br><br></h2>

<b> Enter Date of Birth: <input type=date id = DOB></b>

<span id = "message" style="color:red"></span><br><br>

<button type="submit" onclick = "ageCalculator()"> Calculate age </button><br><br>

<h3 style="color:32A80F" id="result" align="center"></h3>

</center>

</body>

</html>

### **Example 4**

In this example, we are calculating the age by converting dates difference in milliseconds. It is also an easy way to calculate age.

<html>

<head>

<script>

function ageCalculator() {

//collect input from HTML form and convert into date format

var userinput = document.getElementById("DOB").value;

var dob = new Date(userinput);

//check user provide input or not

if(userinput==null || userinput==''){

document.getElementById("message").innerHTML = "\*\*Choose a date please!";

return false;

}

//execute if user entered a date

else {

//extract and collect only date from date-time string

var mdate = userinput.toString();

var dobYear = parseInt(mdate.substring(0,4), 10);

var dobMonth = parseInt(mdate.substring(5,7), 10);

var dobDate = parseInt(mdate.substring(8,10), 10);

//get the current date from system

var today = new Date();

//date string after broking

var birthday = new Date(dobYear, dobMonth-1, dobDate);

//calculate the difference of dates

var diffInMillisecond = today.valueOf() - birthday.valueOf();

//convert the difference in milliseconds and store in day and year variable

var year\_age = Math.floor(diffInMillisecond / 31536000000);

var day\_age = Math.floor((diffInMillisecond % 31536000000) / 86400000);

//when birth date and month is same as today's date

if ((today.getMonth() == birthday.getMonth()) && (today.getDate() == birthday.getDate())) {

alert("Happy Birthday!");

}

var month\_age = Math.floor(day\_age/30);

day\_age = day\_age % 30;

var tMnt= (month\_age + (year\_age\*12));

var tDays =(tMnt\*30) + day\_age;

//DOB is greater than today?s date, generate an error: Invalid date

if (dob>today) {

document.getElementById("result").innerHTML = ("Invalid date input - Please try again!");

}

else {

document.getElementById("result").innerHTML = year\_age + " years " + month\_age + " months " + day\_age + " days"

}

}

}

</script>

</head>

<body>

<center>

<h2 style="color: #008CBA" align="center"> Calculate Age from Date of Birth <br><br></h2>

<b> Enter Date of Birth: <input type=date id = DOB></b>

<span id = "message" style="color:red"></span><br><br>

<button type="submit" onclick = "ageCalculator()"> Calculate age </button><br><br>

<h3 style="color:#008CBA" id="result" align="center"></h3>

</center>

</body>

</html>

# **JavaScript label statement**

JavaScript label is a statement used to prefix a label as an identifier. You can specify the label by any name other than the reserved words. It is simply used with a colon (:) in code.

A label can be used with a **break** or **continue** statement to control the flow of the code more precisely. The label is applied to a block of code or a statement.

Using some examples, we will learn how to define and use the label statement in JavaScript.

### **Syntax**

label: statements

### **Parameters**

**label:** It is a JavaScript identifier. Define it by any name that is not a reserved keyword.

**Statements:** It is a JavaScript statement, where **break** is simply used with the labelled statement and **continue** with looping labelled statement.

### **Examples**

Let's understand the JavaScript label that how it works and helps to break or continue with the looping statement with the help of different examples.

### **Example: Label with for loop to break**

In this example, we will define two labels by the name **innerloop** and **outerloop**, which is used with for loop to break the execution of the loop for a specified condition.

<html>

<body>

<script>

var i, j;

//Execution of outerloop and innerloop using label

document.write("Entering the loop!<br /> ");

outerloop: // This is the label name for the below loop

for (i = 0; i< 5; i++) {

document.write("<b>Outerloop i: </b>" + i + "</br>");

innerloop: //another label

for (j = 0; j <= 4; j++) {

//when j is greater than 3, quit the innermost loop

if (j >3 ) {

document.write("<b> Break innermost loop when j>3 </b></br>");

break ;

}

// when i = 2, exit from innerloop

if (i == 2) {

document.write("<b> Break innerloop when i=2 </b></br>");

break innerloop;

}

// when i = 4, exit from outerloop too

if (i == 4) {

document.write("<b> Break outerloop when i=4 </b></br>");

break outerloop;

}

document.write("Innerloop execution j: " + j + " <br />");

}

}

document.write("Exit from all loops! </br> ");

</script>

</body>

</html>

### **Example: Label with for loop to continue**

In this example, we will again define two labels by the name **innerloop** and **outerloop**. But now they are used with for loop to continue the execution of the loop when the specified condition occurs.

<html>

<body>

<script>

var i,j;

//Execution of loops using outerloop and innerloop label

document.write("Entering the loop! </br> ");

outerloop: // This is the label name

for (i = 0; i< 4; i++) {

document.write("<b>Outerloop: </b>" + i + "</br>");

innerloop:

for (j = 0; j < 4; j++) {

if (i> 2) {

document.write("<b> Continue Innerloop when i>2 </b></br>");

continue innerloop;

}

if (j == 3) {

document.write("<b> Continue Outerloop when j=3 </b></br>");

continue outerloop;

}

document.write("Innerloop execution: " + j + "<br />");

}

}

document.write("Exit from all loops!<br /> ");

</script>

</body>

</html>

# **JavaScript String with quotes**

The strings are usually used to store and manipulate the text data. However, you can also store special characters and numeric data in strings as well. **E.g.,** address or email id, which contains multitype data.

Strings are created by putting data inside the quotes. JavaScript and other programming languages allow the users to keep the data either in double quotes (" ") or single quotes (' '). This chapter will show you how to print string with quotes.

### **For example**

|  |  |
| --- | --- |
| **Quotes** | **Example** |
| Single Quote (' ') | 'This is a string inside single quotes.' |
| Double Quote (" ") | "This is a string inside double quotes." |

These single quotes and double quotes do not print with string on the web browser. But sometimes we need to print the quotes with the string as well. Some words like it's, b'day, seven o'clock, can't, and etc. Firstly, see how a string simply display on the web.

### **Example: String without quotes**

<script>

var pat1 = 'A string inside single quote';

var pat2 = "A string inside double quote";

document.write(pat1 + "</br>");

document.write(pat2);

</script>

## Enclosing quotation marks

**You can use a backslash (\) with the particular word or string to escape the quotation mark.** Remember one thing; if you do not want to use the backslash (\), you have to use the quotation mark alternatively inside and outside of a string. This means that if you try to use a single quote inside a string, the outside quotes should be double quotes. Similarly, if you try to use a double quote inside a string, the outside quotes must be single quotes.

### **Example: Print quotes using backslash (\)**

<html>

<body>

<script>

var singleQ = 'It\'s nine o\' clock in the morning.';

var doubleQ = "Mukesh Ambani is \"the richest man\" of India.";

document.write(singleQ + "</br>");

document.write(doubleQ + "</br>");

</script>

</body>

</html>

### **Example: Print quotes using alternative String syntax**

In this example, we will use alternative quotation mark inside and outside of a string. This means the same thing can be done with it. See the below example, how it will be done:

<html>

<body>

<script>

var singleQ = "It's nine o' clock in the morning.";

var doubleQ = 'Always say "Thank you" when anyone helps you.';

document.write(singleQ + "</br>");

document.write(doubleQ + "</br>");

</script>

</body>

</html>

Other than all these methods, there is one more way to display the single quote and double quote on the web browser.

### **Example: Use apostrophe to print single quote**

In this example, we will use apostrophe in the middle of the single-quote string. See the below example, how it will be done:

<html>

<body>

<script>

var sq = 'It's an example of printing the single quote with string.';

document.write(sq);

</script>

</body>

</html>

### **Example: Use &quot to print double quote**

In JavaScript, you can use **&quot** with a string to display string with double quotation mark. With **&quot,** you can use any quote. See the below example:

<html>

<body>

<script>

var dq1 = "Always say " Thank you " when anyone helps you.";

var dq2 = 'Always say " Thank you " when anyone helps you.';

document.write(dq1 + "</br>");

document.write(dq2);

</script>

</body>

</html>

# **How to create dropdown list using JavaScript?**

Before starts creating a dropdown list, it is important to know what is a dropdown list. A dropdown list is a toggleable menu that allows the user to choose one option from multiple ones. The options in this list are defined in coding, which is associated with a function. When you click or choose this option, that function triggers and starts performing.

You have seen a dropdown list most of the time on registration forms to select the state or city from the dropdown menu. A dropdown list allows us to choose only one from the list of items. See the below screenshot how the dropdown list looks like-

### **Important points to create a dropdown list**

* The <select> tab is used with <option> tab to create the simple dropdown list in HTML. After that JavaScript helps to perform operation with this list.
* Other than this, you can use the container tab <div> to create the dropdown list. Add the dropdown items and links inside it. We will discuss each method with an example in this chapter.
* You can use any element such as <button>, <a>, or <p> to open the dropdown menu.

See the below examples to create a dropdown list using different methods.

### **Examples**

### **Simple dropdown list using <select> tab**

It is a simple example of creating a simple and easy dropdown list without using any complicated JavaScript code and CSS stylesheet.

<html>

<head><title>dropdown menu using select tab</title></head>

<script>

function favTutorial() {

var mylist = document.getElementById("myList");

document.getElementById("favourite").value = mylist.options[mylist.selectedIndex].text;

}

</script>

<body>

<form>

<b> Select youfavourite tutorial site using dropdown list </b>

<select id = "myList" onchange = "favTutorial()" >

<option> ---Choose tutorial--- </option>

<option> w3schools </option>

<option>iHub</option>

<option>tutorialspoint</option>

<option>geeksforgeeks</option>

</select>

<p> Your selected tutorial site is:

<input type = "text" id = "favourite" size = "20" </p>

</form>

</body></html>

A dropdown list can be created using some other ways; see some more below examples.

### **Dropdown list using button and div tab**

In this example, we will create a dropdown list with a button having a list of items as a dropdown menu.

<html>

<style>

/\* set the position of dropdown \*/

.dropdown {

position: relative;

display: inline-block;

}

/\* set the size and position of button on web \*/

.button {

padding: 10px 30px;

font-size: 15px;

}

/\* provide css to background of list items \*/

#list-items {

display: none;

position: absolute;

background-color: white;

min-width: 185px;

}

/\* provide css to list items \*/

#list-items a {

display: block;

font-size: 18px;

background-color: #ddd;

color: black;

text-decoration: none;

padding: 10px;

}

</style>

<script>

//show and hide dropdown list item on button click

function show\_hide() {

var click = document.getElementById("list-items");

if(click.style.display ==="none") {

click.style.display ="block";

} else {

click.style.display ="none";

}

}

</script>

<body>

<div class="dropdown">

<button onclick="show\_hide()" class="button">Choose Language</button>

<center>

<div id="list-items">

<a href="#"> Hindi </a>

<a href="#"> English </a>

<a href="#"> Spanish </a>

<a href="#"> Chinese </a>

<a href="#"> Japanese </a>

</div>

</center>

</div>

</body>

</html>

### **Multiple dropdown list Example**

In the above examples, we have created a single dropdown list. We will now create a form with multiple dropdown menu of various online technical subject tutorials lists like C, C++, PHP, MySQL, and Java, categorised into several categories. When the user clicks on a particular dropdown button, their respective dropdown list will open up to you.

<html>

<head>

<style>

.dropbtn {

background-color: green;

color: white;

padding: 14px;

font-size: 16px;

cursor: pointer;

}

.dropbtn:hover {

background-color: brown;

}

.dropdown {

position: relative;

display: inline-block;

}

.dropdown-content {

display: none;

position: absolute;

background-color: white;

min-width: 140px;

overflow: auto;

box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);

}

.dropdown-content a {

color: black;

padding: 12px 16px;

text-decoration: none;

display: block;

}

.dropdown a:hover {

background-color: #ddd;

}

.show {

display: block;

}

</style>

</head>

<body>

<h2>List of tutorials using Dropdown menu</h2>

<p>Click on the button to open the tutorial dropdown menu.</p>

<div class="dropdown">

<button onclick="programmingList()" class="dropbtn">Programming</button>

<div id="myDropdown1" class="dropdown-content">

<a href="#java" onclick="java()">Java</a>

<a href="#python" onclick="python()">Python</a>

<a href="#c++" onclick="cpp()">C++</a>

<a href="#c" onclick="c()">C</a>

</div>

</div>

<div class="dropdown">

<button onclick="databaseList()" class="dropbtn">Database</button>

<div id="myDropdown2" class="dropdown-content">

<a href="#mysql" onclick="mysql()">MySQL</a>

<a href="#mdb" onclick="mDB()">MongoDB</a>

<a href="#cass" onclick="cassandra()">Cassandra</a>

</div>

</div>

<div class="dropdown">

<button onclick="WebTechList()" class="dropbtn">Web Technology</button>

<div id="myDropdown3" class="dropdown-content">

<a href="#php" onclick="php()">PHP</a>

<a href="#css" onclick="css()">CSS</a>

<a href="#js" onclick="js()">JavaScript</a>

</div>

</div>

<script>

/\* methods to hide and show the dropdown content \*/

function programmingList() {

document.getElementById("myDropdown1").classList.toggle("show");

}

function databaseList() {

document.getElementById("myDropdown2").classList.toggle("show");

}

function WebTechList() {

document.getElementById("myDropdown3").classList.toggle("show");

}

/\* methods to redirect to tutorial page that user will select from dropdown list \*/

function java() {

window.location.replace("https://www.ihubtalent.com/java-tutorial");

}

function python() {

window.location.replace("https://www.ihubtalent.com/python-tutorial");

}

function cpp() {

window.location.replace("https://www.ihubtalent.com/cpp-tutorial");

}

function c() {

window.location.replace("https://www.ihubtalent.com/c-programming-language-tutorial");

}

function mysql() {

window.location.replace("https://www.ihubtalent.com/mysql-tutorial");

}

function mDB() {

window.location.replace("https://www.ihubtalent.com/mongodb-tutorial");

}

function cassandra() {

window.location.replace("https://www.ihubtalent.com/cassandra-tutorial");

}

function php() {

window.location.replace("https://www.ihubtalent.com/php-tutorial");

}

function css() {

window.location.replace("https://www.ihubtalent.com/css-tutorial");

}

function js() {

window.location.replace("https://www.ihubtalent.com/javascript-tutorial");

}

// Close the dropdown menu if the user clicks outside of it

window.onclick = function(event) {

if (!event.target.matches('.dropbtn')) {

var dropdowns = document.getElementsByClassName("dropdown-content");

var i;

for (i = 0; i<dropdowns.length; i++) {

var openDropdown = dropdowns[i];

if (openDropdown.classList.contains('show')) {

openDropdown.classList.remove('show');

}

}

}

}

</script>

</body>

</html>

# **How to disable radio button using JavaScript?**

Radio button is an input type that is used to get input from the user by selecting one value from multiple choices. You have seen the radio buttons to choose gender between male and female. We select only one entry, either male or female and leave the other entries are unselected.

There might be some cases when we need to disable the other entries based on some conditions. You can enable and disable the radio button by using the **disabled** property of HTML DOM. Set this property to true (**disable=true**) to disable the radio button in JavaScript.

### **Disable the radio button**

Sometimes, we need to disable the radio button for a specific condition. These are special conditions when we disable the radio button. When the radio buttons get disabled, their color changed to **grey**.

In the below examples, we will learn how to disable the radio button:

### **Disable radio button using dropdown**

Here, we will use a dropdown list having **Yes** and **No** as values to enable or disable the radio buttons. If you choose No, all the radio buttons will be disabled. On the other hand, all the radio buttons will be enabled if you select Yes.

#### **Note that you can use the checkbox as well instead of a dropdown list.**

<html>

<script>

function verifyAnswer() {

//get the selected value from the dropdown list

var mylist = document.getElementById("myAns");

var result = mylist.options[mylist.selectedIndex].text;

if (result == 'No') {

//disable all the radio button

document.getElementById("csharp").disabled = true;

document.getElementById("js").disabled = true;

document.getElementById("angular").disabled = true;

} else {

//enable all the radio button

document.getElementById("csharp").disabled = false;

document.getElementById("js").disabled = false;

document.getElementById("angular").disabled = false;

}

}

</script>

<body>

<h2> Disable radio Button using dropdown </h2>

<form>

<!-- create a dropdown list -->

<h3> Are you a developer? </h3>

<select id = "myAns" onchange = "verifyAnswer()" >

<option value="choose"> --choose -- </option>

<option value="yes"> Yes </option>

<option value="no"> No </option>

</select>

</form>

<p><b> If Yes, Choose language your preferred programming Language</b></p>

<!-- create a set of radio buttons -->

<label><input type="radio" name="programming" id="csharp" value= "csharp"> C# </label>

<label><input type="radio" name="programming" id="js" value= "js"> JavaScript </label>

<label><input type="radio" name="programming" id="angular" value= "angular"> Angular </label>

</body>

</html>

### **Disable radio button using checkbox**

<html>

<script>

function verifyAnswer() {

//disable all the radio button

document.getElementById("csharp").disabled = true;

document.getElementById("js").disabled = true;

document.getElementById("angular").disabled = true;

//get the value if checkbox is checked

var dev = document.getElementById("myCheck").checked;

if (dev == true) {

//enable all the radio button

document.getElementById("csharp").disabled = false;

document.getElementById("js").disabled = false;

document.getElementById("angular").disabled = false;

}

}

</script>

<body>

<h2> Disable radio Button using checkbox </h2>

<form>

<!-- create a dropdown list -->

<h3> Are you a developer? </h3>

Yes:

<input type="checkbox" id="myCheck" onchange="verifyAnswer()" checked>

</form>

<p><b> If Yes, Choose language your preferred programming Language</b></p>

<!-- create a set of radio buttons -->

<label><input type="radio" name="programming" id="csharp" value= "csharp"> C# </label>

<label><input type="radio" name="programming" id="js" value= "js"> JavaScript </label>

<label><input type="radio" name="programming" id="angular" value= "angular"> Angular </label>

</body>

</html>

### **A simple radio button example**

It is a simple example of radio button created. In this example, we will create set of radio button for gender and language selection. The input will be taken using the HTML form and calculated by JavaScript. See the code below:

<html>

<script>

function calValue() {

//fetch all gender radio button data

var male = document.getElementById('g1');

var female = document.getElementById('g2');

var otherg = document.getElementById('g3');

//fetch all language radio button data

var hindi = document.getElementById('l1');

var english = document.getElementById('l2');

var otherl = document.getElementById('l3');

var gender;

var language;

//check which gender is selected using radio button

if(male.checked == true) {

gender = male;

} else if(female.checked == true) {

gender = female;

} else if(otherg.checked == true) {

gender = otherg

}

//check which language is selected using radio button

if(hindi.checked == true) {

language = hindi;

} else if(english.checked == true) {

language = english;

} else if(otherl.checked == true) {

language = otherl

}

//return data to HTML form

return document.getElementById("result").innerHTML = "Your selected gender is: " + gender.value + "</br> and </br> Selected language is: " + language.value;

}

</script>

<body>

<h2> Simple radio Buttons Example </h2>

<!-- create radio button for gender selection -->

<p><b> Select your gender: </b></p>

<input type="radio" id="g1" name="gender" value="male">

<label for="male"> Male </label><br>

<input type="radio" id="g2" name="gender" value="female">

<label for="female"> Female </label><br>

<input type="radio" id="g3" name="gender" value="other">

<label for="other"> Other </label>

<br>

<!-- create radio button for language selection -->

<p><b> Select your language: </b></p>

<input type ="radio" id="l1" name="language" value="hindi">

<label for ="male"> Hindi </label><br>

<input type="radio" id="l2" name="language" value="english">

<label for="female"> English </label><br>

<input type="radio" id="l3" name="language" value="other">

<label for="other"> Other </label><br><br>

<input type="submit" value="Submit" onclick="calValue()">

<h3 id="result" style="color:blue"></h3>

</body>

</html>

# **Check if the value exists in Array in Javascript**

In a programming language like Javascript, to check if the value exists in an array, there are certain methods. To be precise, there are plenty of ways to check if the value we are looking for resides amongst the elements in an array given by the user or is predefined. Let's discuss these methods one by one using various examples.

## indexof() method

The indexof() method in Javascript is one of the most convenient ways to find out whether a value exists in an array or not. The indexof() method works on the phenomenon of index numbers. This method returns the index of the array if found and returns -1 otherwise. Let's consider the below code:

**<script>**

     var army=["Marcos", "DeltaForce", "Seals", "SWAT", "HeadHunters"];

if(army.indexOf("Marcos") !== -1)  {

        alert("Yes, the value exists!")

}

else  {

        alert("No, the value is absent.")

}

**</script>**

The above code prints the given out because the value is already present in the array. It is quite easy to understand that the expected value is present at position 0. Thus, the indexof() method tells you that the value expected is present in the given array.

includes() method

The [includes() method](https://www.javatpoint.com/javascript-array-includes-method) is one such method using which we can easily find out whether the expected value exists in the given array. There are various ways to use include() method. This method returns a Boolean value i.e. **true** if the value exists and **false** if it incorrect. The includes() method can be used in various ways to find out if the value exists. To name a few, take a look at the below examples to understand.

varspecialForces=["BlackCats","Marcos", "Demolishers","HeadHunters"];

r\_name = specialForces.includes("HeadHunters");

In the above method, we have defined two variables as shown. The includes() methods return **true** because the value which we are looking for is already present in the given array. If the value was not present in the array, the includes() methods might have returned false.

Another way of using the includes() method is by assigning the index value through which the element we are looking for is generated as output. See the below code for reference.

var actors = ["Hrithik", "SRK", "Salman", "Vidyut"];

var names = actors.includes("Vidyut", 3);

In the above code snippet, we have defined the variable "**actors**" which the value. We have also defined a variable "names" which would return true or false, if the includes() method returns the shown result. The code above will return true since the value and the index number have been correctly assigned and would return the output.

The above examples are some of the predefined methods that we have used to check whether an element exists in the array or not. We have another approach to find out an array element using loops. Let's discuss how can we check if the element exists in an array using loops as shown in the below code snippet.

Using loops

var example\_array = ['Rahul','Rajesh','Sonu','Siddhi','Mark','George'];

function checkArray(value,array{

   var status = 'Absent';

   for(var i=0; i**<array.length**; i++)  {

     var name = array[i];

     if(name == value){

       status = 'Present';

       break;

     }

   }

  return status;

}

# **JavaScript Debouncing**

In this article, we will discuss the **JavaScript debouncing ()** method and its implementation.

## What is Debouncing?

Debouncing is a method used in JavaScript to increase browser performance. There may be some features on a web page that needs time-consuming computations. If such type of method is applied frequently, it may greatly affect the browser's performance because Javascript is a **single-threaded language**. Debouncing is a programming technique that assures that time-consuming activities do not trigger the web page's performance decreases. In other words, the Debounce methods do not run when invoked. Instead, they wait a predetermined period of time until executing. When we call the same process again, the previous process is canceled, and the timer is reset.

A debounce is a throttle cousin, and they both help improve the web application's performance. Although, they are seen in different situations. When we just think about the final state, a debounce is used. **For example**, they are waiting until a user has finished typing to retrieve typeahead search results. If we want to manage all intermediate states at a regulated pace, a throttle is the best tool to use.

## Implementation of debouncing

Let's take an example to see the implementation of debounce method in the program.

const debounce = (func, wait) =**>** {

  let timeout;

  return function mainFunction(...args) {

     const later = () =**>** {

       clearTimeout(timeout);

       func(...args);

     };

     clearTimeout(timeout);

     timeout = setTimeout(delay, wait);

  };

};

A higher-order function that returns another function is known as a debounce function. It is used to create a closure around the **func** and **wait** function parameters and the **timeout variable** to hold their values. The definitions of the following variables are:

1. **Func:** It is the **func** function that we want to execute after the debounce time.
2. **Wait:** The time after the last received action that the debounce function can wait until executing func.
3. **Timeout:** The timeout function is the value that is used to indicate a running debounce.

### **Example:**

Let's take an example to understand the **debouncing()** method in JavaScript. A button is connected to an event listener, which calls a debounce function in the below example. The Debounce function has **two parameters**: a **function** and the other is a **number (time)**. A **Timer** declares, which as the name implies, and calls the debounce function after a certain amount of time.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<h1>**JavaScript Debounce**</h1>**

**<input** type = "button" id="debounce" value = "Click Here"**>**

**<script>**

var button = document.getElementById("debounce");

const debounce = (func, wait) =**>** {

    let debounceTimer

    return function() {

        const context = this

        const args = arguments

            clearTimeout(debounceTimer)

                debounceTimer

            = setTimeout(() =**>** func.apply(context, args), wait)

    }

}

button.addEventListener('click', debounce(function() {

        alert("Hello\n This message will be displayed after 3 seconds, and no matters how many times we click the button.")

}, 4000));

**</script>**

**</body>**

**</html>**

As we have seen in the above screenshot, there is a **"Click here"** button. When the **Click Here** button is pressed, a warning box appears and displays an alert message. The feature updates every time, which means that if the button is pressed before the delay time (4 seconds), the initial timer is cleared, and a new timer is started. The **clearTimeOut()** function is used to complete this mission.

## Implementation of debounce function with immediate function

The following debounce implementation returns a function that will not be called as long as it is invoked. After **N milliseconds** of inactivity, the function will be called again. When the function is called with the initial function as an argument, it called the function immediately and waits for the interval before calling it again.

function debounce(func, wait, immediate) {

  var timeout;

  return function mainFunction() {

    var cont = this;

    var args = arguments;

    var later = function() {

      timeout = null;

      if (!immediate) func.apply(cont, args);

    };

    var callNow = immediate && !timeout;

    clearTimeout(timeout);

    timeout = setTimeout(delay, wait);

    if (callNow) func.apply(cont, args);

  };

};

Debounce returns a function that may be passed on to the function's event listeners when a function and a time interval are passed.

var returnedFunction = debounce(function() {  }, 3000);

window.addEventListener('resize', returnedFunction);

### **Example:**

Let's take an example to understand the use of debounce function with the immediate function.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<h1>**JavaScript Debounce**</h1>**

**<button** id="debounce"**>**Click here **</button>**

**<script>**

    var button = document.getElementById("debounce");

    const debounce = (func, wait, immediate)=**>** {

     var timeout;

    return function executedFunction() {

        var cont = this;

        var args = arguments;

        var later = function() {

        timeout = null;

        if (!immediate) func.apply(cont, args);

        };

        var callNow = immediate && !timeout;

        clearTimeout(timeout);

        timeout = setTimeout(later, wait);

        if (callNow) func.apply(cont, args);

        };

    };

    button.addEventListener('click', debounce(function() {

            alert("This message will be displayed after 3 seconds, and no matters how many times we click the button.")

                            }, 3000));

**</script>**

**</body>**

**</html>**

# **JavaScript print() method**

In this section, we will discuss the print() method in the JavaScript language. 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.

### **Syntax**

The following syntax is used to print the current content of the window:

window.print()

In the above syntax, we use the window.print() method that prints the currently visible content of the window screen.

**Parameters**: It does not contain any parameters.

**Returns**: The window.print() method does not return anything.

### **Supported Browser of the print() method**

Following are the browsers that support the window.print() method.

1. Google Chrome
2. Internet Explorer
3. Firebox
4. Opera
5. Safari

### **Example 1: Program to print a web page using print() method**

In this program, we are using a window.print() method that prints the current visible content in the window screen.

**<html>**

**<head>**

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

**</head>**

**<body>**

**<h2>** To print the Current Content of the window using print() method **</h2>**

**<p>**

As the name suggests, the print () method is used to print the contents of the current window. When we use a print () method, it opens the print dialog box, which allows the user or programmer to select an appropriate option for printing the current content of the window.

**</p>**

**<form>**

<!-- When a user click on the print button, the onclick function calls the window.print() method to print the currently visible content in the window screen. -->

**<input** type = "button" value = "Print" onclick = "window.print()" **/>**

**</form>**

**</body>**

**</html>**

### **Example 2: Program to print the Student Registration Form using the print() method**

In this program, we create a student registration form and then print it using window,print() method.

<!-- Create a web page to print the Student registration Form in JavaScript using print() method. -->

**<html>**

**<head>**

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

function printFun{

window.print();

}

**</script>**

<!-- Start the coding for CSS -->

**<style>**

/\* Create the Outer layout of the Calculator. \*/

.formstyle  {

width: 400px;

height: 400px;

margin: 20px auto;

border: 3px solid skyblue;

border-radius: 5px;

padding: 20px;

text-align: center;

background-color: lightgreen;  }

/\* Display top horizontal bar that contain some information. \*/

h1 {

    text-align: center;

    padding: 23px;

    background-color: skyblue;

    color: white;

    }

\*{

margin: 0;

padding: 0;

}

**</style>**

**</head>**

**<body** bgcolor = "lightgrey"**>**

**<h1>** Program to print the Student Registration Form using JavaScript print() method **</h1>**

**<div** class = "formstyle">

**<form** name = "form1"**>**

**<fieldset>**

**<br>**

**<legend>** Student Registration Form: **</legend>**

**<label>** First name **</label>**

**<input** type = "text" name = "fname" size = "30" **/>** **<br>**

**<br>**

**<label>** Last name **</label>**

**<input** type = "text" name = "lname" size = "30" **/>** **<br>**

**<br>**

**<label>** Father name **</label>**

**<input** type = "text" name = "f\_name" size = "30" **/>** **<br>**

**<br>**

**<label>** Mother name **</label>**

**<input** type = "text" name = "m\_name" size = "30" **/>** **<br>**

**<label>** Gender: **</label>**

**<input** type = "radio" name = "gender" **/>** Male

**<input** type = "radio" name = "gender" **/>** Female **<br>**

**<br>**

**<label>**Address**</label>**

**<textarea** cols = "30" rows = "3" value = "address"**>**

**</textarea>**

**<label>**  Email  **</label>**

**<input** type = "email" id = "email" name = "email" size ="30" **/>** **</br>**

**<br>**

**<label>** Password:  **</label>**

**<input** type = "password" id = "pass" name = "pwd" size = "30"**>** **<br>**

**<br>**

**<input** type = "reset"  value = "Reset"**/>**

**<input** style = "background-color:skyblue;" width = 30px height = 20px type = "button" value = "Print" onclick = "printFun()"**/>**

**</fieldset>**

**</form>**

**</div>**

**</body>**

**</html>**

Now we fill in all the details of the student registration form and click on the Print button, which shows the below image.

After click on the Print button, it shows the different operations of the **window.print** method. Once we are done with all settings in the Print dialog box, click on the Save button to save the file or page in your system.

**Explanation of the code:** In the above program, we have created a student registration form, and this form has two buttons Reset and Print button. The Reset button is used to reset the content, and the Print button is used to print the displayed content. When we click on the Print button, an onclick function will activate and call the PrintFun() function. Now PrintFun() function is executed that contains window.print() method to print the displayed content on the window screen.

# **JavaScript editable table**

In this article, we will understand how to create an editable table with the help of JavaScript. In the beginning, we will understand a basic overview of JavaScript programming. After that, we will understand this concept with the help of some examples.

### **What is JavaScript?**

JavaScript abbreviated as JS. It is a dynamic programming language used for web development, web applications, game development, and many more. It allows you to implement dynamic features on web pages that cannot be done only with HTML and CSS.

It is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax {}, dynamic typing, prototype-based object orientation, and first-class functions.

### **What do you mean by JavaScript editable table?**

After creating a table in JavaScript, you can insert and combine rows and columns or format the table by adjusting table cell widths, colors, and alignment. You can use the **contenteditable** attribute on the cells, rows, or table to edit a table.

**Let's take various examples of JavaScript editable tables.**

### **Example 1:**

<! DOCTYPE html**>**

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width, initial-scale=1"**>**

**<meta** charset="UTF-8"**>**

**<title>** JavaScript editable table **</title>**

**<link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.css"**>**

**<link** href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"**>**

**<style>**

@import url('https://fonts.googleapis.com/css?family=Montserrat:400,500');

body{

  font-family: 'Montserrat', sans-serif;

  text-align: center;

  background-color: rgb(63,72,83);

  font-family: sans-serif;

  color: rgb(220,220,220);

  overflow-x: hidden;

}

tr:first-child { color: #FB667A; }

td:hover {

  color: white;

  font-weight: bold;

  transition-delay: 0s;

  transition-duration: 0.4s;

  transition-property: all;

  transition-timing-function: line;

}

h1 {

  position: relative;

  padding: 0;

  margin: 10;

  font-family: "Raleway", sans-serif;

 font-weight: 400;

  font-size: 40px;

  color: white;

  -webkit-transition: all 0.4s ease 0s;

  -o-transition: all 0.4s ease 0s;

  transition: all 0.4s ease 0s;

}

.table {

   width: 100%;

  thead {

    th {

      padding: 10px 10px;

      background: #00adee;

      font-size: 25px;

      text-transform: uppercase;

      vertical-align: top;

      color: #1D4A5A;

      font-weight: normal;

      text-align: left;

    }

  }

  tbody {

    tr {

      td {

        padding: 10px;

        background: #f2f2f2;

        font-size: 14px;

      }

    }

  }

}

.add {

  outline: none;

  background: none;

  border: none;

}

.edit {

  outline: none;

  background: none;

  border: none;

}

.save {

  outline: none;

  background: none;

  border: none;

}

.delete {

  outline: none;

  background: none;

  border: none;

}

.edit {

  padding: 5px 10px;

  cursor: pointer;

}

.save {

  padding: 5px 10px;

  cursor: pointer;

}

.delete {

  padding: 5px 10px;

  cursor: pointer;

}

.add {

  float: right;

  background: transparent;

  border: 1px solid  black;

  color: black;

  font-size: 13px;

  padding: 0;

  padding: 3px 5px;

  cursor: pointer;

  &:hover {

    background: #ffffff;

    color: #00adee;

  }

}

.save {

  display: none;

  background: #32AD60;

  color: #ffffff;

  &:hover {

    background: darken(#32AD60, 10%);

  }

}

.edit {

  background: #2199e8;

  color: #ffffff;

  &:hover {

    background: darken(#2199e8, 10%);

  }

}

.delete {

  background: #EC5840;

  color: #ffffff;

   &:hover {

    background: darken(#EC5840, 10%);

  }

}

**</style>**

**<body>**

**<table** class="table data"**>**

**<h1>** Example **</h1>**

**<thead>**

**<tr>**

**<th>** Name **</th>**

**<th>** Email **</th>**

**<th>** Number **</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr>**

**<td** class="data"**>** John Doe **</td>**

**<td** class="data"**>** johndoe@john.com **</td>**

**<td** class="data"**>** 666-666-666 **</td>**

**<td>**

**<button** class="save"**>** Save **</button>**

**<button** class="edit"**>** Edit **</button>**

**<button** class="delete"**>** Delete **</button>**

**</td>**

**</tr>**

**<tr>**

**<td** class="data"**>** John Doe **</td>**

**<td** class="data"**>** johndoe@john.com **</td>**

**<td** class="data"**>** 666-666-666 **</td>**

**<td>**

**<button** class="save"**>** Save **</button>**

**<button** class="edit"**>** Edit **</button>**

**<button** class="delete"**>** Delete **</button>**

**</td>**

**</tr>**

**</tbody>**

**<tbody>**

**<tr>**

**<td** class="data"**>** John Doe **</td>**

**<td** class="data"**>** johndoe@john.com **</td>**

**<td** class="data"**>** 666-666-666 **</td>**

**<td>**

**<button** class="save"**>** Save **</button>**

**<button** class="edit"**>** Edit **</button>**

**<button** class="delete"**>** Delete **</button>**

**</td>**

**</tr>**

**<tr>**

**<td** class="data"**>** John Doe **</td>**

**<td** class="data"**>** johndoe@john.com **</td>**

**<td** class="data"**>** 666-666-666 **</td>**

**<td>**

**<button** class="save"**>** Save **</button>**

**<button** class="edit"**>** Edit **</button>**

**<button** class="delete"**>** Delete **</button>**

**</td>**

**</tr>**

**</tbody>**

**</table>**

**<script** src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"**>** **</script>**

**<script>**

$(document).on('click', '.edit', function() {

  $(this).parent().siblings('td.data').each(function() {

     var content = $(this).html();

     $(this).html('**<input** value="' + content + '" **/>**');

  });

  $(this).siblings('.save').show();

  $(this).siblings('.delete').hide();

  $(this).hide();

});

$(document).on('click', '.save', function() {

  $('input').each(function() {

    var content = $(this).val();

    $(this).html(content);

    $(this).contents().unwrap();

  });

  $(this).siblings('.edit').show();

  $(this).siblings('.delete').show();

  $(this).hide();

});

$(document).on('click', '.delete', function() {

   $(this).parents('tr').remove();

});

$('.add').click(function() {

  $(this).parents('table').append('**<tr><td** class="data"**></td><td** class="data"**></td><td** class="data"**></td><td><button** class="save"**>**Save**</button><button** class="edit"**>**Edit**</button>** **<button** class="delete"**>**Delete**</button></td></tr>**');  });

**</script>**

**</body>**

**</html>**

**Example 2:**

<! DOCTYPE html**>**

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width, initial-scale=1"**>**

**<meta** charset="UTF-8"**>**

**<title>** JavaScript editable table **</title>**

**<link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.css"**>**

**<link** href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"**>**

**</head>**

**<style>**

 @import url('https://fonts.googleapis.com/css?family=Montserrat:400,500');

body {

  font-family: 'Montserrat', sans-serif;

  padding: 0;

  margin: 0;

  text-align: center;

}

h1 {

  position: relative;

  padding: 0;

  margin: 10;

  font-family: "Raleway", sans-serif;

  font-weight: 300;

  font-size: 40px;

  color: #080808;

  -webkit-transition: all 0.4s ease 0s;

  -o-transition: all 0.4s ease 0s;

  transition: all 0.4s ease 0s;

}

body {

  font-size: 0.75em;

}

table {

  border-spacing: 10px;

}

tr **>** th {

  text-align: right;

}

.table-fill {

  background: white;

  border-radius:3px;

  border-collapse: collapse;

  height: 320px;

  margin: auto;

  max-width: 600px;

  padding:5px;

  width: 100%;

  box-shadow: 0 5px 10px rgba(0, 0, 0, 0.1);

  animation: float 5s infinite;

}

th {

  color: #D5DDE5;;

  background: #1b1e24;

  border-bottom: 4px solid #9ea7af;

  border-right: 1px solid #343a45;

  font-size: 23px;

  font-weight: 100;

  padding: 24px;

  text-align: left;

  text-shadow: 0 1px 1px rgba(0, 0, 0, 0.1);

  vertical-align: middle;

}

th:first-child {

  border-top-left-radius: 3px;

}

th:last-child {

  border-top-right-radius: 3px;

  border-right: none;

}

  tr {

  border-top: 1px solid #C1C3D1;

  border-bottom-: 1px solid #C1C3D1;

  color: #666B85;

  font-size: 16px;

  font-weight: normal;

  text-shadow: 0 1px 1px rgba(256, 256, 256, 0.1);

}

 tr:hover td {

  background: #4E5066;

  color: #FFFFFF;

  border-top: 1px solid #22262e;

}

 tr:first-child {

  border-top: none;

}

tr:last-child {

  border-bottom: none;

}

tr:nth-child(odd) td {

  background: #EBEBEB;

}

tr:nth-child(odd):hover td {

  background: #4E5066;

}

tr:last-child td:first-child {

  border-bottom-left-radius: 3px;

}

tr:last-child td:last-child {

  border-bottom-right-radius: 3px;

}

 td:last-child {

  border-right: 0px;

}

td {

  background: #FFFFFF;

  padding: 20px;

  text-align: left;

  vertical-align: middle;

  font-weight: 300;

  font-size: 18px;

  text-shadow: -1px -1px 1px rgba(0, 0, 0, 0.1);

  border-right: 1px solid #C1C3D1;

}

**</style>**

**<body>**

**<h1>** Example **</h1>**

**<table** summary="Editable table with datasets ordered in columns" class="table-fill"**>**

**<tbody>**

**<tr>**

**<th** scope="col"**>** Month **</th>**

**<th** scope="col"**>** Sales **</th>**

**</tr>**

**<tr>**

**<th** scope="row"**>** January **</th>**

**<td>** 16000 **</td>**

**</tr>**

**<tr>**

**<th** scope="row"**>** February **</th>**

**<td>** 10000**</td>**

**</tr>**

**<tr>**

**<th** scope="row"**>** March **</th>**

**<td>** 20000 **</td>**

**</tr>**

**<tr>**

**<th** scope="row"**>** April **</th>**

**<td>** 7300 **</td>**

**</tr>**

**<tr>**

**<th** scope="row"**>** May **</th>**

**<td>** 12000 **</td>**

**</tr>**

**<tr>**

**<th** scope="row"**>** June **</th>**

**<td>** 1099 **</td>**

**</tr>**

**</tbody>**

**</table>**

**<br** **/>**

**<script** src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"**>** **</script>**

**<script>**

var table = document.querySelector('table');

var dataCells = table.querySelectorAll('tr **>** td');

var rows = table.querySelectorAll('tr');

var code = document.querySelector('code');

var resetButton = document.querySelector('.reset');

rows = Array.prototype.slice.call(rows, 1);

var ncols = rows[0].children.length - 1;

var initialData = {0:[1,2,3,4,5],1:[6,7,8,9,10],2:[11,12,13,14,15]};

function parseTable () {

  var d = {};

  Array.prototype.forEach.call(rows, function (row, i) {

    var rowrowCells = row.querySelectorAll('td');

    return Array.prototype.map.call(rowCells, function (cell, j) {

      if (!d[j]) d[j] = [];

      d[j].push(parseInt(cell.textContent, 10));

    });

  });

  return d;

}

function setTableData (data) {

  Array.prototype.forEach.call(rows, function (row, i) {

    var rowrowCells = row.querySelectorAll('td');

    return Array.prototype.map.call(rowCells, function (cell, j) {

      cell.textContent = initialData[j][i];

    });

  });

}

Array.prototype.forEach.call(dataCells, function (cell) {

  cell.contentEditable = true;

});

table.addEventListener('keyup', function (e) {

  if (e.target.tagName === 'TD') {

    setTimeout(function () {

      code.innerText = JSON.stringify( parseTable() );

    }, 0);

  }

});

**</script>**

**</body>**

**</html>**

### **Example 3:**

<! DOCTYPE html**>**

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width, initial-scale=1"**>**

**<meta** charset="UTF-8"**>**

**<title>** JavaScript editable table **</title>**

**<link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.css"**>**

**<link** href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"**>**

**</head>**

**<style>**

 @import url('https://fonts.googleapis.com/css?family=Montserrat:400,500');

body {

  font-family: 'Montserrat', sans-serif;

  padding: 0;

  margin: 0;

  text-align: center;

}

h1 {

  position: relative;

  padding: 0;

  margin: 10;

  font-family: "Raleway", sans-serif;

  font-weight: 300;

  font-size: 40px;

  color: #080808;

  -webkit-transition: all 0.4s ease 0s;

  -o-transition: all 0.4s ease 0s;

  transition: all 0.4s ease 0s;

}

tr:nth-of-type(odd) {

    background: #eee;

    }

    th {

    background: #3498db;

    color: white;

    font-weight: bold;

    }

@import "compass/css3";

.table-editable {

  position: relative;

  .glyphicon {

    font-size: 20px;

  }

}

table {

    width: 750px;

    border-collapse: collapse;

    margin:50px auto;

    }

    td, th {

    padding: 10px;

    border: 1px solid #ccc;

    text-align: left;

    font-size: 18px;

    }

.table-remove {

  color: #700;

  cursor: pointer;

  &:hover {

    color: #f00;

  }

}

.table-up {

  color: #007;

  cursor: pointer;

  &:hover {

    color: #00f;

  }

}

.table-down {

  color: #007;

  cursor: pointer;

  &:hover {

    color: #00f;

  }

}

.table-add {

  color: #070;

  cursor: pointer;

  position: absolute;

  top: 8px;

  right: 0;

  &:hover {

    color: #0b0;

  }

}

**</style>**

**<body>**

**<div** class="container"**>**

**<h1>** JavaScript Editable Table **</h1>**

**<div** id="table" class="table-editable"**>**

**<table** class="table"**>**

**<tr>**

**<th>** Name**</th>**

**<th>** Roll No **</th>**

**<th>** Class **</th>**

**<th>** Marks **</th>**

**</tr>**

**<tr>**

**<td** contenteditable="true"**>** Ram **</td>**

**<td** contenteditable="true"**>** 1 **</td>**

**<td** contenteditable="true"**>** BA **</td>**

**<td** contenteditable="true"**>** 48 **</td>**

**</tr>**

**<tr>**

**<td** contenteditable="true"**>** Rama **</td>**

**<td** contenteditable="true"**>** 10 **</td>**

**<td** contenteditable="true"**>** BSC **</td>**

**<td** contenteditable="true"**>** 40 **</td>**

**</tr>**

**<tr>**

**<td** contenteditable="true"**>** sham **</td>**

**<td** contenteditable="true"**>** 8 **</td>**

**<td** contenteditable="true"**>** BCA **</td>**

**<td** contenteditable="true"**>** 34 **</td>**

**</tr>**

**<tr>**

**<td** contenteditable="true"**>** shama **</td>**

**<td** contenteditable="true"**>** 3 **</td>**

**<td** contenteditable="true"**>** BCA **</td>**

**<td** contenteditable="true"**>** 30 **</td>**

**</tr>**

**</table>**

**</div>**

**</div>**

**<script** src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"**>** **</script>**

**<script>**

var $TABLE = $('#table');

var $BTN = $('#export-btn');

var $EXPORT = $('#export');

$('.table-add').click(function () {

  var $clone = $TABLE.find('tr.hide').clone(true).removeClass('hide table-line');

  $TABLE.find('table').append($clone);

});

$('.table-remove').click(function () {

  $(this).parents('tr').detach();

});

$('.table-up').click(function () {

  var $row = $(this).parents('tr');

  if ($row.index() === 1) return;

  $row.prev().before($row.get(0));

});

$('.table-down').click(function () {

  var $row = $(this).parents('tr');

  $row.next().after($row.get(0));

});

jQuery.fn.pop = [].pop;

jQuery.fn.shift = [].shift;

$BTN.click(function () {

  var $rows = $TABLE.find('tr:not(:hidden)');

  var headers = [];

  var data = [];

    $($rows.shift()).find('th:not(:empty)').each(function () {

    headers.push($(this).text().toLowerCase());

  });

  $rows.each(function () {

    var $td = $(this).find('td');

    var h = {};

        headers.forEach(function (header, i) {

      h[header] = $td.eq(i).text();

    });

        data.push(h);

  });

    });

**</script>**

**</body>**

**</html>**

## JavaScript Advance

# **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, listed below.

## 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.

### **Unit8ClampedArray**

* **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.

### **Uint32Array**

* **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:** 1.2x10-38 to 3.4x1038

### **Float64Array**

* **Size in bytes:** 8
* **Description:** 64-bit IEEE floating point number unrestricted double.
* **Type:** unrestricted double.
* **Value Range:** 5.0x10-324 to 1.8x10308

## JavaScript TypedArray Methods

Let's see the list of JavaScript TypedArray methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| copyWithin() | The copyWithin () method copies a portion of an array to another location in the same array and returns the size without modification. |
| entries() | The JavaScript entries() method returns a new Array Iterator object that contains key/value pairs for each index in the array. |
| every() | javaScriptArray.every() method test whether all the elements of the array satisfy the given condition or not. |
| fill() | The javaScriptfill() method is used to fill all the elements of array from a start index to an end index with a static value. |
| Filter() | The JavaScript Array filter() method form a new array that fall under a given criteria from an existing array. |
| find() | The JavaScript find() Method is used to get the value of the first element in the array that satisfies the provided condition. |
| findIndex() | The JavaScript findIndex() method is provide the index of the element that complete the given test in the array. |
| forEach() | The JavaScript forEach() method calls the provided function once for each element of the array. |
| includes() | The JavaScript Array includes() method is inbuilt function in JavaScript which is used to determines whether a particular element is present in the array or not. |
| indexof() | The JavaScript indexof() Method is used to find the index of the element provided as the argument to the function. |
| join() | The JavaScript join() method is used to join all elements of an Array into a string. |
| Keys() | The JavaScript Keys() method is an inbuilt function in JavaScript. This method returns an Array Iterator object with the keys of an array. |
| lastIndexof() | The javaScriptlastIndex()of method returns the last position of a value, or it return -1 if the value is not found. |
| map() | The JavaScript map() method form a new array with the result of calling a function for every element. |
| reduce() | The JavaScript reduce() method reduce the elements of an array into a single value. |
| reduceRight() | The JavaScript reduceRight() method reduce the elements of an array into a single value. |
| reverse() | The JavaScript reverse() method is used to reverse the array. |
| set() | The JavaScript set() method is used to store values into the given array. |
| Slice() | The JavaScript slice() method gives the selected elements of the array on which is implemented. |
| some() | JavaScript some() method examine the elements of the array that satisfies the given condition or not. |
| sort() | The JavaScript sort() method is used to sort the array and returns the updated array. |
| subarray() | The JavaScript subarray() method returns a new array and it does not change the original array. |
| values() | The JavaScript values() method is used to define the value of the contents in the array |
| toLocaleString() | The JavaScript toLocaleString() method is used to convert the element of the given array into a string. |
| toString() | The JavaScript toString() method is used to convert the element of the given array into a string. And these Strings are separated by such as a comma ",". |

# **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.

## Syntax

**new** Set([iterable])

## Parameter

**iterable** - 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

Let's see the list of JavaScript set methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| add() | It adds the specified values to the Set object. |
| clear() | It removes all the elements from the Set object. |
| delete() | It deletes the specified element from the Set object. |
| entries() | It returns an object of Set iterator that contains an array of [value, value] for each element. |
| forEach() | It executes the specified function once for each value. |
| has() | It indicates whether the Set object contains the specified value element. |
| values() | It returns an object of Set iterator that contains the values for each element. |

**Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Sets</h2>

<p>Create a Set from an Array:</p>

<p id="demo"></p>

<script>

// Create a Set

const letters = new Set(["a","b","c"]);

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

</body>

</html>

**Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Sets</h2>

<p>Add values to a Set:</p>

<p id="demo"></p>

<script>

// Create a Set

const letters = new Set();

// Add Values to the Set

letters.add("a");

letters.add("b");

letters.add("c");

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

</body>

</html>

**Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Sets</h2>

<p>forEach() calls a function for each element:</p>

<p id="demo"></p>

<script>

// Create a Set

const letters = new Set(["a","b","c"]);

// List all Elements

let text = "";

letters.forEach (function(value) {

text += value + "<br>";

})

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

</script>

</body>

</html>

**Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Sets</h2>

<p>entries() Returns an Iterator with [value,value] pairs from a Set:</p>

<p id="demo"></p>

<script>

// Create a Set

const letters = new Set(["a","b","c"]);

// List all entries

const iterator = letters.entries();

let text = "";

for (const entry of iterator) {

text += entry + "<br>";

}

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

document.getElementById("demo").innerHTML = typeof letters;

document.getElementById("demo").innerHTML = letters instanceof Set;

</script>

</body>

</html>

# **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.

## Syntax

**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

Let's see the list of JavaScript map methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
|  |  |
| clear() | It removes all the elements from a Map object. |
| delete() | It deletes the specified element from a Map object. |
| entries() | It returns an object of Map iterator that contains the key-value pair for each element. |
| forEach() | It executes the specified function once for each key/value pair. |
| get() | It returns the value of specified key. |
| has() | It indicates whether the map object contains the specified key element. |
| keys() | It returns an object of Map iterator that contains the keys for each element. |
| set() | It adds or updates the key-value pairs to Map object. |
| values() | It returns an object of Map iterator that contains the values for each element. |

## Example:

# !DOCTYPE html>

# <html>

# <body>

# <h2>JavaScript Map Objects</h2>

# <p>Creating a Map from an Array:</p>

# <p id="demo"></p>

# <script>

# // Create a Map

# const fruits = new Map([

# ["apples", 500],

# ["bananas", 300],

# ["oranges", 200]

# ]);

fruits.set("apples", 200);

# document.getElementById("demo").innerHTML = fruits.get("apples");

document.getElementById("demo").innerHTML = fruits.size;

fruits.delete("apples");

document.getElementById("demo").innerHTML = fruits.size;

fruits.clear();

document.getElementById("demo").innerHTML = fruits.size;

document.getElementById("demo").innerHTML = fruits.has("apples");

document.getElementById("demo").innerHTML = typeof fruits;

document.getElementById("demo").innerHTML = fruits instanceof Map;

let text = "";

fruits.forEach (function(value, key) {

text += key + ' = ' + value + "<br>"

})

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

let veggies = "";

for (const x of fruits.keys()) {

veggies += x + "<br>";

}

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

let total = 0;

for (const x of fruits.values()) {

total += x;

}

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

let text = "";

for (const x of fruits.entries()) {

text += x + "<br>";

}

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

# </script>

# </body>

# </html>

# **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.

## Syntax

**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.

## JavaScript WeakSet Methods

Let's see the list of JavaScript WeakSet methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| add() | It adds a new object to the end of WeakSet object. |
| delete() | It removes the specified object from the WeakSet object. |
| has() | It indicates whether the WeakSet object contains the specified object element. |

# **JavaScript WeakMap Object**

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values.

## Syntax

**new** WeakMap([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 WeakMap object allows the keys of object type only.
* In WeakMap, if there is no reference to a key object, they are targeted to garbage collection.
* In WeakMap, the keys are not enumerable. So, it doesn't provide any method to get the list of keys.
* A WeakMap object iterates its elements in insertion order.

## JavaScript WeakMap Methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| delete() | It deletes the specified element from the WeakMap object. |
| get() | It returns the value of specified key. |
| has() | It indicates whether the WeakMap object contains the specified value element. |
| set() | It adds or updates the key-value pairs to WeakMap object. |

# **JavaScript callback**

A callback function can be defined as a function passed into another function as a parameter. Don't relate the callback with the keyword, as the callback is just a name of an argument that is passed to a function.

In other words, we can say that a function passed to another function as an argument is referred to as a callback function. The callback function runs after the completion of the outer function. It is useful to develop an asynchronous JavaScript code.

In JavaScript, a callback is easier to create. That is, we simply have to pass the callback function as a parameter to another function and call it right after the completion of the task. Callbacks are mainly used to handle the asynchronous operations such as the registering event listeners, fetching or inserting some data into/from the file, and many more.

Now, let's see how to create a callback by using some illustrations.

It is an example of an asynchronous callback. Asynchronicity can be defined as if JavaScript has to wait to complete the operation and execute the rest of the program during waiting.

### **Example1**

In this example, there are two functions **getData( x, y, callback)** and **showData()**. Here, we are calling the **getData()** with the **showData();** that is, we are passing it as the third argument of the **getData()** function along with two parameters. As a result, the **getData()** is invoked with the specified parameters, including the callback.

The **getData()** function display the multiplication of two numbers and once it gets completed the callback function will get executed. In the output, we can see the data of the **showData()** function gets printed after the output of **getData()** function.

<html>

<body>

<h1> Hello World :) :) </h1>

<h3> The getData() function is called its arguments and the callback is executed after the completion of getData() function. </h3>

<script>

function getData(x, y, callback){

document.write(" The multiplication of the numbers " + x + " and " + y + " is: " + (x\*y) + "<br><br>" );

callback();}

function showData(){

document.write(' This is the showData() method execute after the completion of getData() method.');

}

getData(20, 30, showData);

</script>

</body>

</html>

Callbacks are generally used to continue the execution after completing an asynchronous operation - such are referred to as the **asynchronous callbacks**.

### **Example2**

It is another example of using callbacks. It is an example of synchronous callback that gets immediately executed.

Here, there are two functions **getData(callback),** which takes the input from the user using the prompt box, and the function **showData(name, amt),** which displays the data entered by the user using the alert dialog box.

<html>

<body>

<h1> Hello World :) :) </h1>

<h2> This is the ihubtalent.com </h2>

<script>

function showData(name, amt) {

alert(' Hello ' + name + '\n Your entered amount is ' + amt);

}

function getData(callback) {

var name = prompt(" Welcome to the ihubtalent.com \n What is your name?");

var amt = prompt(" Enter some amount...");

callback(name, amt);

}

getData(showData);

</script>

</body>

</html>

# **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 is 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.

Let's understand the closure by using an example.

### **Example1**

<!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>**

**</html>**

In the above program we have two functions: **fun()** and **innerfun()**. The function **fun()** creates the local variable **a** and the function **innerfun()**. The inner function **innerfun()** is only present in the body of **fun()**. The inner function can access the outer function's variable, so the function **innerfun()** can access the variable **'a'**, which is declared and defined in **fun()**.

This is the closure in action in which the inner function can have access to the global variables and outer function variables.

The entire body of function **innerfun()** is returned and stored in the variable **output**, due to the statement **return innerfun**. The inner function is not executed only by using the **return** statement; it is executed only when followed by the braces **()**.

In the output, the code will display the value of the variable **'a'**, defined in the parent function.

Now, there is another example in which we will use the parameterized function

### **Example2**

<!DOCTYPE html**>**

**<html>**

**<head>**

**<script>**

function fun(a)  {

function innerfun(b){

return a\*b;

 }

return innerfun;

}

var output = fun(4);

document.write(output(4));

document.write(" ");

document.write(output(5));

**</script>**

**</head>**

**</html>**

In the above program there are two parameterized functions: **fun()** and **innerfun()**. The function **fun()** has a parameter **a**, and the function **innerfun()** has the parameter **b**. The function **fun()** returns a function **innerfun()** which takes an argument and returns the multiplication of **a** and **b**. In the program, the **output** is the closure.

### **Example3**

<!DOCTYPE html**>**

**<html>**

**<head>**

**<script>**

function fun{

function closure(val) {

return function() {

return val;

}

}

var a = [];

var i;

for (i = 0; i **<** **5**; i++)  {

a[i] = closure(i);

}

return a;

}

var output = fun();

document.write(output[0]());

document.write(" ");

document.write(output[1]());

document.write(" ");

document.write(output[2]());

document.write(" ");

document.write(output[3]());

document.write(" ");

document.write(output[4]());

**</script>**

**</head>**

**</html>**

Closure points the variable and stores the reference of a variable. They don't remember the variable's value. In the above code, we are updating the function closure() argument with every call. So, we will get the different values of the variable i, at different index.

Closures are one of the slightly difficult to understand concept of JavaScript, but try to practice the closure in different scenarios like to create callbacks, getters/setter.

# **JavaScript date format**

The JavaScript date object can be used to get a year, month and day. We can display a timer on the webpage 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 formats 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"

The ISO date format follows a strict JavaScript's standard, while the other formats (Short date and Long date) are browser dependent and not so well defined.

Now, let's understand these date formats individually.

## 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.

The example of using the ISO date is given below.

### **Example**

The output of the below code will display the complete date, which is relative to the current time zone.

<!DOCTYPE html>

<html>

<body>

<div>

<h1> Welcome to the IHubTalent.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>

Now, we are discussing some other formats of the ISO date. Here, we are writing the input date and displaying the result that occurred when we use the corresponding syntax.

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");**

1. 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");**

1. 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");**

1. Wed Jan 01 2020 05:30:00 GMT+0530 (India Standard Time)

**4.** Now, in the fourth 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.

JavaScript uses the browser's time zone if we set the date without specifying the time zone.

**let val = new Date("2020-08-01T07:05:00Z");**

1. Sat Aug 01 2020 12:35:00 GMT+0530 (India Standard Time)

Now, we discuss the short date format along with an example.

## 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**

Here, we are specifying the short date format, i.e., "MM/DD/YYYY".

<!DOCTYPE html>

<html>

<body>

<div>

<h1> Welcome to the IHubTalent.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 in any order, and it is allowed to write a month in abbreviated (Aug) form or in full (August).

Now, we understand the Long date by using an example.

### **Example**

Here, we are using the Long date format, i.e., "MMM DD YYYY", and specifying the month in abbreviated form.

<!DOCTYPE html>

<html>

<body>

<div>

<h1> Welcome to the IHubTalent.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 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.

### **Syntax**

date.parse(datestring);

It contains a single parameter string that represents a date. This method returns a number that represents the number of milliseconds.

Let's see some illustrations of using the **parse()** method. In the first example, we are passing the valid date value, and in the second example, we are passing the invalid date value to see the result.

### **Example1**

In this example, we are passing a valid date to get the number of milliseconds between the specified date and midnight of **January 1, 1970**.

Here, the specified date is **"June 19, 2020"**.

<html>

<body>

<h1> Hello World :) :) </h1>

<p> Here, we are finding the number of milliseconds between the given date and midnight of January 1, 1970. </p>

<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>

### **Example2**

In this example, we are passing an invalid date to see what happens when we provide invalid input.

<html>

<body>

<h1> Hello World :) :) </h1>

<p> Here, we are finding the number of milliseconds between the given date and midnight of January 1, 1970. </p>

<script>

var d1 = "June 39, 2020"; //an invalid date

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>

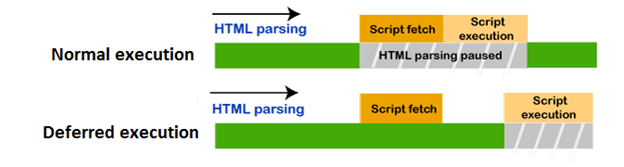
</body>

</html>

# **JavaScript defer**

The **defer** is a Boolean value, used to indicate that script is executed after the document has been parsed. It works only with external 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 the **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.

Sometimes the script takes more than expected loading time and displays the blank page instead of content. In mobile devices, it will be a worse situation because of the low memory of small devices. So, by using the **defer** attribute, we can increase the loading performance.

The **defer** attribute is not allowed in older browsers, so for older browsers we have to use the alternative of the **defer** attribute. The alternative solution is that we must have to specify the **<script>** section just before the </body> tag of HTML file. It can be done as follows:

### **Syntax**

**<script** defer**>**

### **Example**

Here, we are using an external javascript file, i.e., **myscript.js**.

<!DOCTYPE html**>**

**<html>**

**<head>**

**<script** src = "myscript.js" defer**>**  **</script>**

**</head>**

**<body>**

**<div>**

**<h1>** iHubTalent.com **</h1>**

**<h3>** This is an example of defer attribute.  **</h3>**

**</div>**

**</body>**

**</html>**

**myscript.js**

alert(" Hello World. \n Welcome to the iHubTalent.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 happens because of the page redirection. It is different from refreshing a page.

Generally, search engines do not analyze the JavaScript to check the redirection. So, if it is required to notify the search engines (SEO) about the URL forwarding, we need to add the **rel = "canonical"** tag within the head section of the web page.

**<link** rel = "canonical" href = "https://www.iHubTalent.com/"  **/>**

There are several methods used for performing page redirection, but **location.href** and **location.replace()** are widely used. The page redirection is easy in JavaScript.

## window.location and window.location.href

**window.location** object is a property of the window object. There are several methods to redirect a web page. Almost all methods are related to the window.location object.

It can be used for getting the address of the current URL or the web address. The window.location object can be written without adding the window prefix.

### **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.

### **Syntax**

window.location.replace("new URL");

Now, let's understand the page redirection by using some examples.

### **Example1**

It is a simple example of page redirection on the client-side. To redirect a page, we simply have to write a statement in the script section.

In this example, there is a button which redirect's the visitor to **'iHubTalent.com'**. We have to click the button to naviagate on the appropriate link.

<html>

<head>

<script type = "text/javascript">

function page\_redirect(){

window.location = "https://www.iHubTalent.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>

### **Example2**

In this example, we are using the **setTimeout()** method which automatically redirect the user to the appropriate link. It would take some time to load a new page. The **setTimeout()** method executes another function after a given time interval.

<html>

<head>

<script type = "text/javascript">

function page\_redirect() {

window.location = "https://www.iHubTalent.com/";

}

setTimeout('page\_redirect()', 5000);

</script>

</head>

<body>

<h2> After 5 seconds, you will redirected to another page. </h2>

<p> Wait for 5 seconds to see the effect. </p>

</body>

</html>

### **Example3**

In this example, we are using the **replace()** method for page redirection. The **replace()** method will replace the current document with the new document.

Here, there is an HTML button that is to be clicked to replace the document with the new one.

<!DOCTYPE html>

<html>

<head>

<script>

function page\_redirect() {

location.replace("https://www.iHubTalent.com")

}

</script>

</head>

<body>

<h2> Example of redirecting a page using replace() </h2>

<p> Using the replace() the currct document will replace with new one. </p>

<p> Click the following button to see the effect. </p>

<button onclick = "page\_redirect()"> Replace </button>

</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 global 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.

In the function's body, the precedence of the local variable is more than the global variable with the same name. If the name of the function's local variable is same as the name of the global variable, then the local variable hides the global variable.

### **Example1**

In this example, we are declaring two variables one variable has a global scope, and the second variable has local scope. Both of the variables are declared with the same name.

In the output, we can see that the variable with locally scoped overrides the value of the global variable.

<!DOCTYPE html>

<html>

<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>

When we declare a variable inside a function without using the **var** keyword, it acts as a global variable. Let's see an illustration of the same.

### **Example2**

In this example, we are declaring a variable inside the function without using any variable declaration keyword. Then we are accessing the corresponding variable outside the function.

In the output, we can see that there is no error is generating related to the **undefined variable.** The code is successfully executed without generating any error.

<!DOCTYPE html>

<html>

<body>

<script>

function example() {

$var12 = 300;

document.write("Inside example() function = " + $var12);

}

example();

document.write("<br>");

document.write("Outside example() function = " + $var12);

</script>

</body>

</html>

In the above code, if we use the variable without calling the function, then the variable is undefined, and there is no output will be generated. In this case, the error will generate related to the undefined variable.

# **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 JavaScript's **addEventListener()** method and pass a **scroll** event to it for greater flexibility.

### **Syntax**

Now, we see the syntax of using the **onscroll** event in HTML and in JavaScript (without **addEventListener()** method or by using the **addEventListener()** method).

**In HTML**

**<element** onscroll = "fun()"**>**

**In JavaScript**

object.onscroll = function() { myScript };

**In JavaScript by using the addEventListener() method**

object.addEventListener("scroll", myScript);

Let's see some of the illustrations to understand the scroll event.

### **Example - Using onscroll attribute in HTML**

In this example, we are using the HTML **onscroll** attribute. There is a paragraph element with **id = "para"** on which we are applying the **onscroll** attribute. When the user scrolls the paragraph, the color and the background color of the paragraph will change.

<!DOCTYPE html>

<html>

<head>

<style>

#para{

width: 250px;

height: 150px;

overflow: scroll;

border: 1px solid red;

font-size: 25px;

}

</style>

</head>

<body>

<h1> Hello world :):) </h1>

<h2> Scroll the bordered text to see the effect. </h2>

<p> This is an example of using the <b>onscroll</b> attribute. </p>

<p id = "para" onscroll = "fun()"> Hi, Welcome to the iHubTalent.com. This site is developed so that students may learn computer science related technologies easily. The iHubTalent.com is always providing an easy and in-depth tutorial on various technologies. No one is perfect in this world, and nothing is eternally best. But we can try to be better. </p>

<script>

function fun() {

document.getElementById("para").style.color = "red";

document.getElementById("para").style.background = "lightgreen";

}

</script>

</body>

</html>

Now, we will see how to use **onscroll** event using JavaScript.

### **Example - Using JavaScript**

<!DOCTYPE html>

<html>

<head>

<style>

#para{

width: 250px;

height: 150px;

overflow: scroll;

border: 1px solid red;

font-size: 25px;

}

</style>

</head>

<body>

<h1> Hello world :):) </h1>

<h2> Scroll the bordered text to see the effect. </h2>

<p> This is an example of using JavaScript's <b>onscroll</b> event. </p>

<p id = "para"> Hi, Welcome to the iHubTalent.com. This site is developed so that students may learn computer science related technologies easily. The iHubTalent.com is always providing an easy and in-depth tutorial on various technologies. No one is perfect in this world, and nothing is eternally best. But we can try to be better. </p>

<p id = "para1"></p>

<script>

document.getElementById("para").onscroll = function() {fun()};

function fun() {

document.getElementById("para").style.color = "red";

document.getElementById("para").style.background = "lightgreen";

document.getElementById("para1").innerHTML = "You are scrolling the content";

}

</script>

</body>

</html>

### **Example - Using addEventListener()**

<!DOCTYPE html>

<html>

<head>

<style>

#para{

width: 250px;

height: 150px;

overflow: scroll;

border: 1px solid red;

font-size: 25px;

}

</style>

</head>

<body>

<h1> Hello world :):) </h1>

<h2> Scroll the bordered text to see the effect. </h2>

<p id = "para"> Hi, Welcome to the iHubTalent.com. This site is developed so that students may learn computer science related technologies easily. The iHubTalent.com is always providing an easy and in-depth tutorial on various technologies. No one is perfect in this world, and nothing is eternally best. But we can try to be better. </p>

<p id = "para1"></p>

<script>

document.getElementById("para").addEventListener("scroll", fun);

function fun() {

document.getElementById("para").style.color = "red";

document.getElementById("para").style.background = "lightgreen";

document.getElementById("para1").innerHTML = "You are scrolling the content";

}

</script>

</body>

</html>

# **JavaScript blur**

<!DOCTYPE html>

<html>

<body>

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>

</body>

</html>

## Definition and Usage

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](https://www.w3schools.com/jsref/event_onfocus.asp) event.

**Tip:**The onblur event is similar to the [onfocusout](https://www.w3schools.com/jsref/event_onfocusout.asp) event. The main difference is that the onblur event does not bubble. Therefore, if you want to find out whether an element or its child loses focus, you could use the onfocusout event. However, you can achieve this by using the optional *useCapture* parameter of the [addEventListener()](https://www.w3schools.com/jsref/met_element_addeventlistener.asp) method for the onblur event.

<!DOCTYPE html>

<html>

<body>

<p>This example demonstrates how to assign an "onblur" event to an input element.</p>

<p>Write something in the input field, and then click outside the field to lose focus (blur).</p>

<input type="text" onblur="myFunction()">

<script>

function myFunction() {

alert("Input field lost focus.");

}

</script>

</body>

</html>

**By using addEventListenet():**

<!DOCTYPE html>

<html>

<body>

<p>This example uses the addEventListener() method to attach a "blur" event to an input element.</p>

<p>Write something in the input field, and then click outside the field to lose focus (blur).</p>

<input type="text" id="fname">

<script>

document.getElementById("fname").addEventListener("blur", myFunction);

function myFunction() {

alert("Input field lost focus.");

}

</script>

</body>

</html>

**By using onfocus() and onblur() together:**

<!DOCTYPE html>

<html>

<body>

<p>When you enter the input field, a function is triggered which sets the background color to yellow. When you leave the input field, a function is triggered which sets the background color to red.</p>

Enter your name: <input type="text" id="myInput" onfocus="focusFunction()" onblur="blurFunction()">

<script>

function focusFunction() {

// Focus = Changes the background color of input to yellow

document.getElementById("myInput").style.background = "yellow";

}

function blurFunction() {

// No focus = Changes the background color of input to red

document.getElementById("myInput").style.background = "red";

}

</script>

</body>

</html>

# **JavaScript** preventDefault()**Event Method**

<!DOCTYPE html>

<html>

<body>

<a id="myAnchor" href="https://w3schools.com/">Go to W3Schools.com</a>

<p>The preventDefault() method will prevent the link above from following the URL.</p>

<script>

document.getElementById("myAnchor").addEventListener("click", function(event){

event.preventDefault()

});

</script>

</body>

</html>

## Definition and Usage

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

* Clicking on a "Submit" button, prevent it from submitting a form
* Clicking on a link, prevent the link from following the URL

**Note:** Not all events are cancelable. Use the [cancelable](https://www.w3schools.com/jsref/event_cancelable.asp) property to find out if an event is cancelable.

**Note:** The preventDefault() method does not prevent further propagation of an event through the DOM. Use the stopPropagation() method to handle this.

## Syntax

event.preventDefault()

<!DOCTYPE html>

<html>

<body>

Try to check this box: <input type="checkbox" id="myCheckbox">

<p>Toggling a checkbox is the default action of clicking on a checkbox. The preventDefault() method prevents this from happening.</p>

<script>

document.getElementById("myCheckbox").addEventListener("click", function(event){

event.preventDefault()

});

</script>

</body>

</html>

# **JavaScript sleep/wait**

The programming languages such as PHP and C has a **sleep(sec)** function to pause the execution for a fixed amount of time. Java has a **thread.sleep()**, python 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. Because of this asynchronous behavior, it continues its work and does not wait for anything during execution. **Async/await** functions help us to write the code in a synchronous manner.

## How to use sleep function in JavaScript?

Before implementing the sleep function in JavaScript, it is important to understand the execution of the JavaScript code.

### **Syntax of sleep() in JavaScript**

sleep(delayTime in milliseconds).then(() =**>** {

// code to be executed

})

The sleep() function can be used along with the async/await to get the pause between the execution. The syntax for the same is given as follows:

### **Syntax**

const func = async () =**>** { await sleep(delayTime in milliseconds)

//code to be executed

}

fun()

The above syntaxes are the way to implement sleep functionality in JavaScript. Now, we will see the examples of using the sleep() function in JavaScript.

### **Example1**

In this example, we are using the **sleep()** function with the **async/await** functionalites. There is a function **fun()** is defined with some statements. Initially, the text **"Hello World"** is displayed on the screen once the function is started. Then, because of the sleep function the **fun()** is paused for 2 seconds. After the completion of the given time period, the text **"Welcome to the iHubTalent.com"** will be displyed on the screen and repeated until the termination of the loop. The text is going to be repeated 10 times on the screen with a pause of two seconds on every iteration of the loop.

<html>

<body>

<h1> Example of using sleep() in JavaScript </h1>

<script>

function sleep(milliseconds) {

return new Promise(resolve =>setTimeout(resolve, milliseconds));

}

async function fun() {

document.write('Hello World');

for (let i = 1; i<=10 ;i++) {

await sleep(2000);

document.write( i + " " + "Welcome to the iHubTalent.com" + " " + "</br>");

}

}

fun();

</script>

</body>

</html>

### **Example2**

Here, we are creating a promise with the **setTimeout()** function. The **setTimeout()** function executes code after the specified amount of time. We are also using the **then()** method, which executes the required function after the completion of the promise.

Initially, some of the statements are displayed on the screen. Then, after the delay of 2 seconds, the text **"End"** will be displayed on the screen.

This approach is preferred to delay a function. Because of using promises, it is supported in ES6.

<!DOCTYPE html>

<html>

<body>

<h1> Example of using the sleep() in JavaScript</h1>

<p>There is a sleep of 2000 milliseconds</p>

<script>

let sleep = ms => {

return new Promise(resolve =>setTimeout(resolve, ms));

};

document.write("Begin" + "<br>");

document.write("Welcome to the iHubTalent.com" + "<br>");

sleep(2000).then(() => {

document.write("End");

});

</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 the page.

### **Syntax**

void expression;

Let's understand the use of **javascript:void(0);** using some examples.

### **Example1**

In the following example, we are defining two links. In the first link, we are using the void keyword. When the corresponding link gets clicked, it will do nothing because of the **void(0)**.

When we click on the second link, it will display an alert dialog box.

<html>

<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 iHubTalent');"> Click here for an alert </a>

</center>

</body>

</html>

### **Example2**

In this example, we are passing the **javascript:void(0);** and link of the website to the **href** attribute. Here, we are also using the **ondblclick** attribute, which shows an alert box on double-clicking the hyperlink. On closing the alert dialog box, the page will not redirect to the specified link of the website.

<html>

<head>

<style>

a{

font-size: 22px;

}

</style>

</head>

<body>

<center>

<h1> Hello World :) :) </h1>

<h2> Click the following link to see the changes </h2>

<h4> You can see on closing the alert dialog box, the page will not redirect to https://www.iHubTalent.com/ </h4>

<a href = "javascript:void(0);https://www.iHubTalent.com/" ondblclick = "alert('Welcome to the iHubTalent.com')">

Double click the link

</a>

</center>

</body>

</html>

# **JavaScript Form**

In this tutorial, we will learn, discuss, and understand the JavaScript form. We will also see the implementation of the JavaScript form for different purposes.

Here, we will learn the method to access the form, getting elements as the JavaScript form's value, and submitting the form.

## Introduction to Forms

Forms are the basics of HTML. We use HTML form element in order to create the JavaScript form. For creating a form, we can use the following sample code:

**<html>**

**<head>**

**<title>** Login Form**</title>**

**</head>**

**<body>**

**<h3>** LOGIN **</h3>**

**<form** name="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<input** type="submit" value="Login"**/>**

**<input** type="button" value="SignUp" onClick="create()"**/>**

**</form>**

**</body>**

**</html>**

In the code:

* 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 want 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 value will be password.
* 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 useful 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.

The syntax of using the **getElementById()** method is as follows:

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

Using the Id, we can make the reference.

### **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 listener 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>**

**<form** name="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="Set a password" id="p1"**></td></tr>**

**<tr><td>**Confirm 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.getElementById("n1").value;

var e=document.getElementById("e1").value;

var p=document.getElementById("p1").value;

var cp=document.getElementById("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.test(n)) {

             alert('Name is incorrect must contain alphabets only');

         }

else if (!email\_val.test(e))  {

             alert('Invalid email format please enter valid email id');

         }

else if(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 successfully... Redirecting to IHubTalent.com");

window.location="https://www.iHubTalent.com/";

}

}

**</script>**

**</body>**

**</html>**