JAVASCRIPT

HANDWRITTEN NOTES

Prepared by:  TOPPERWORLD LEARN & GROW

classmate
Date
Page

INDEX

JavaScript

JavaScript is used to create client-side dynamic pages. JavaScript is an object-based scripting language which is lightweight and cross-platform. JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript 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. 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

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.
4. JavaScript is an object-oriented programming language, wherever it is also use as prototypes rather than using classes for inheritance.
5. It is a lightweighted 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 HTMT 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'.

Application of JavaScript

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

1. Client - Side validation.
2. Dynamic drop-down menus.
3. Displaying date and time.
4. Displaying pop-up windows and dialog boxes.
5. 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.

Example

```
<script type = "text/javascript">
document.Write("JavaScript is a simple language");
</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. 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.

```
<script type = "text/javascript">
alert ("Hello Javapoint");
</script>
```

2. JavaScript Example : Code between the head tag
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 Javapoint");
}
</script>
</head>
<body>
<p> Welcome to JavaScript </p>
<form>
<input type = "button" value = "click" onclick="msg()"/>
</form>
</body>
</html>
```

3. 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.

At external JavaScript file must be saved by js extension. It is recommended to embed all JavaScript files into a single file. It increase the speed of the webpage.
Let's create an external JavaScript
message.js

```
function msg(){
alert("Hello Javapoint");
}
```

Let's include the JavaScript file into html page. It calls the 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

1. It helps in the resuability of code in more than one HTML file.
2. It allows easy code ra 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 parallely and separately, ie., 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

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 JS comments.

1. To make code easy to understand.
2. To avoid the unnecessary code.

Types of JavaScript comments

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.

```
<script>
//It is single line comment
document.Write("hello javascript");
</script>
```

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

```
/*your code here*/
```

JavaScript Variable

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

Rules for declaring Variable

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 Value 1.
3. JavaScript Variables are case sensitive.

JavaScript local variable

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

```
<script>
function abc(){}
var x = 10;// 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 know as global variable.

```
<script>
var data = 200; // global variable
function a(){
}
</script>
```

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 data type

```
var a = 40;//holding number
var b = "Rahul"; // holding string
```

JavaScript Primitive data types

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

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.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10 + 20 = 30
−	Subtraction	20 − 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	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 2 operands.

Operator	Description	Example
==	Is equal to	10==20 = false
===	Indentical	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.

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 0	(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
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 interator.

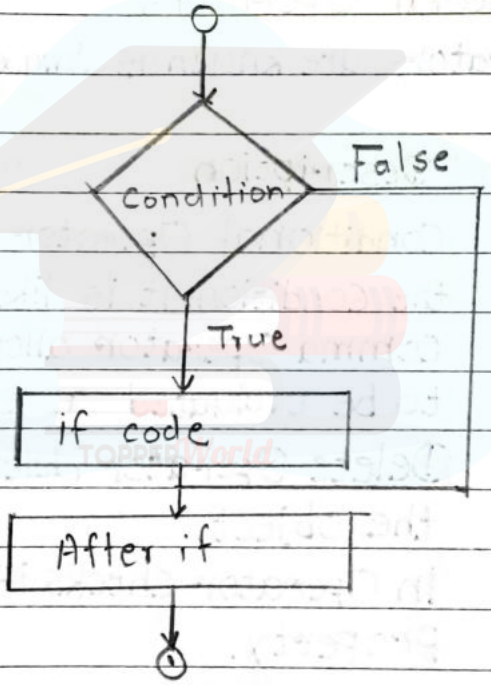
JavaScript If - else

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.

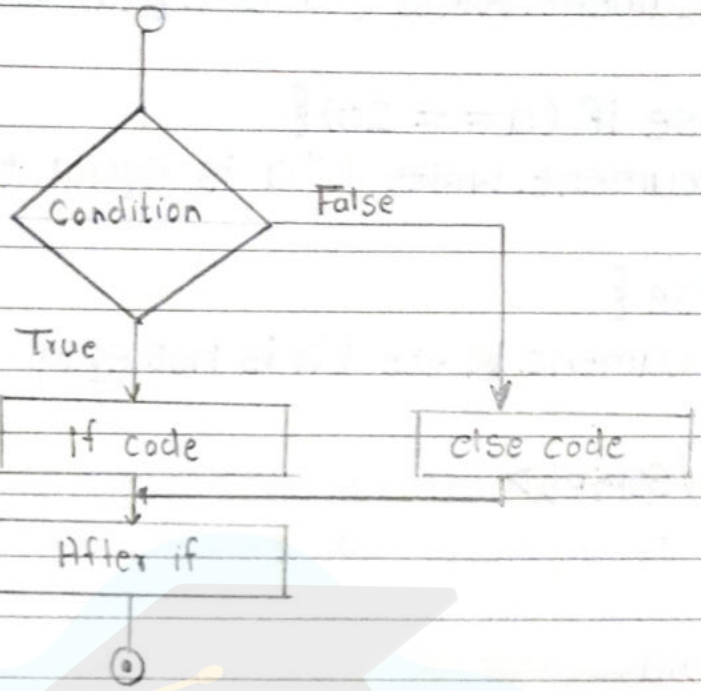


Example

```
<script>
var a = 20;
if (a>10){
document.write("value of a is grater than 10");
}
</script>
```

JavaScript If... else Statement

- If evaluates the content whether condition is true of false.



Example

```
<script>
Var a = 20;
if (a % 2 = 0) {
document.Write ("a is even number");
}
 else {
 document. Write ("a is odd number");
 }
 </script>
```

JavaScript If... else if statement

It evaluates the content only if expression is true.

Example

```
<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>
```

Javascript Switch

The Javascript switch statement is used to execute one code from multiple expressions.

Example

```
<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>
```

Output
B Grade

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) Javascript For loop

The JavaScript for loop iterates the elements for the fixed number of times. It should be used if number of interation is known.

Example

```
<script>
for(i=1;i<=5;i++)
{
document.Write(i+"<br />")
}
</script>
```

Output

```
1
2
3
4
5
```

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.

Example

```
<script>
Var i = 11;
While (i<=15)
{
document.Write(i+"<br/>");
i++;
}
</script>
```

Output

11
12
13
14
15

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.

Example

```
<script>
Var i = 21;
do {
document.Write(i+"<br/>");
j++;
} While (i<=25);
</script>
```

Output

21
22
23
24
25

JavaScript Functions

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

Advantage of JavaScript function

1. Code reusability
2. Less coding

JavaScript Function Example

```
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="Call function"/>
```

Output call function

JavaScript Function Arguments

We can call function by passing arguments.

Example

```
<script>
function getcube(number){
alert(number* number * number);
}
</script>
<form>
<input type="button"= "click" onclick="getcube(4)"/>
</form>
```

Output Click

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.

JavaScript Function Methods

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
Call()	It is used to call a function contains this value and an argument list.
toString()	It returns the result in a form of a string.

Example

```
<script>
Var add =new Function("num1","num2","returnnum1+num2");
document.Write In(add(2,5));
</script>
```

Output 7

JavaScript Objects

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) JavaScript Object by object literal

The syntax of creating object using object literal.

```
Object = {property 1; Value 1, property 2; value 2 .... propertyN: ValueN}
```

2) By creating instance of Object

The syntax of creating object directly.

```
Var objectname = new Object();
```

Here, new keyword is used to create object.

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.

Example

```
<script>
function emp (id,name, salary){
this. id = id;
this. name = name;
this. salary = salary;
}
e = new emp(103, "Vimal Jaiswal", 30000);
document. write (eid + " " + e.name + " " + e. salary);
</script>
```

Output 103 Vimal Jaiswal 30000

JavaScript Object Methods

1. Object.assign()
This method is used to copy enumerable and own properties from a source object to a target object.

2. Object.create()
This method is used to create a new object with specified prototype object and properties.

3. Object.defineProperty()
This method is used to describe some behavioral attributes of the property.

4. Object.defineProperties()
This method is used to create or configure multiple object properties.

5. Object.entries()
This method returns an array with arrays of the key, value pairs.

6. Object.freeze()
This method prevents existing properties from being removed.

7. Object.getOwnPropertyDescriptor()
This method returns a property descriptor for the specified property of the specified object.

8. Object.getOwnPropertyDescriptors()
This method returns all own property descriptors of a

given object.

9. Object.getOwnPropertyNames()
The method returns an array of all properties found.

10. Object.getOwnPropertySymbols()
This method returns an array of all own symbol key properties.

11. Object.getPrototypeOf()
This method returns the prototype of the specified object.

12. Object.is()
This method determines whether two values are the same value.

13. Object.isExtensible()
This method determines if an object is extensible.

14. Object.isFrozen()
This method determines if an object was frozen.

15. Object.isSealed()
This method determines if an object is sealed.

16. Object.keys()
This method returns an array of a given object's own property names.

17. Object.preventExtensions()
This method is used to prevent any extensions of an object.

18. Object.setPrototypeOf()
This method sets the prototype of a specified object to another object.

19. Object.seal()
This method prevents new properties from being added and marks all existing properties as non-configurable.

20. Object.values()
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) JavaScript array literal

```
Var arrayname = [value1, value2.....valueN];
```

2) JavaScript Array directly

```
Var arrayname = new Array();
```

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

3) JavaScript array constructor

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

Example

```
<script>
```

```
Var emp = new Array ("Jai", "Vijay", "Smith");
for (i=0; i< emp. length; i++) {
document. Write ( emp [i] + "<br>");
}
</ script >
```

Output
Jai
Vijay
Smith

JavaScript Array Methods

1. concat ()
It returns a new array object that contains two or more merged arrays.

2. copyWithin()
It copies the part of the given array with its own elements and returns the modified array.

3. entries()
It creates an iterator object and a loop that iterates over each key/value pair.

4. every()
It determines whether all the elements of an array are staisfying the provided function conditions.

5. flat()
It creates a new array carrying sub-array elements concatenated recursively till the specified depth.

6. flatMAP()
It maps all array elements via mapping function, than flattens the result into a new array.

7. fill()
It fills elements into an array With static values.

8. forEach()
It invokes the provided function once for each element of an array.

9. includes()
It checks whether the given array contains the specified element.

10. isArray()
It tests if the passed value ia an array.

11. join()
It joins the elements of an array as a string.

12. push()
It adds one or more elements to the end of an 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

The String literal is created using double quotes.

Var String name = "string value";

2) By string object

The syntax of creating string object using new keyword.

Var Stringname = new String ("string literal");

JavaScript String Methods

1. char At ()
   It provides the char value present at the specified index.

2. Char Code At()
   It provides the Unicode value of a character present at the specified index.

3. concat()
   It provides a combination of two or more strings.

4. index Of()
   It provides the position of a char value present in the given string.

5. last Index Of ()
   It provides the position of a char value present in the given string by searching a character from the last position.

6. Search ()
   It searches a specified regular expression in a given string and returns its position if a match occurs.

7. match()

It searches a specified regular expression in a given string and returns that regular expression if a match occurs.

8. replace()
It replaces a given string with the specified replacement.

9. substr()
It is used to fetch the part of the given string on the basis of the specified starting position and length.

10. substring()
It is used to fetch the part of the given string on the basis of the specified index.

11. toLowerCase()
It converts the given string into lowercase letter.

12. toLocaleLowerCase()
It converts the given string into lowercase letter on the basis of host?s current locale.

13. toUpperCase()
It converts the given string into uppercase letter.

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.

Constructor

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

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

JavaScript Date Methods

1. getDate()
   It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.

2. getDay()
   It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.

3. getFullYears()
   It returns the integer value that represents the year on the basis of local time.

4. getHours()
   It returns the integer value between 0 and 23 that represents the hours on the basis of local time.

5. getMilliseconds()
   It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.

6. getMinutes()
It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.

7. getMonth()
It returns the integer value between 0 and 11 that represents the month on the basis of local time.

8. getSeconds()
It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.

9. getUTCDate()
It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.

10. getUTCDay()
It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.

11. getUTCFullYears()
It returns the integer value that represents the year on the basis of universal time.

JavaScript Math Methods

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

1. abs()
   It returns the absolute value of the given number.

2. acos()
   It returns the arccosine of the given number in radians.

3. asin()
   It returns the arcsine of the given number in radians.

4. atan()
   It returns the arc-tangent of the given number in radians.

5. cbrt()
   It returns the cube root of the given number.

6. ceil()
   It returns a smallest integer value, greater than or equal to the given number.

7. cos()
   It returns the cosine of the given number.

8. cosh()
   It returns the hyperbolic cosine of the given number.

9. exp()
   It returns the exponential form of the given number.

10. floor()
    It returns largest integer value, lower than or equal to the given number.

11. hypot()
It returns square root of sum of the squares of given numbers.

12. log()
It returns natural logarithm of a number.

13. max()
It returns maximum value of the given numbers.

14. min()
It returns minimum value of the given numbers.

15. pow()
It returns value of base to the power of exponent.

16. random()
It returns random number between 0 (inclusive) and 1 (exclusive).

17. round()
It returns closest integer value of the given number.

18. sign()
It returns the sign of the given number.

19. sin()
It returns the sine of the given number.

20. sinh()
It returns the hyperbolic sine of the given number.

21. sqrt()
It returns the square root of the given number.

22. trunc()
It returns an integer part of the given number.

JavaScript Number Object

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

```
Varn = new Number(value);
```

JavaScript Number Constant

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

1. isFinite()
It determines whether the given value is a finite number.

2. isInteger()
It determines whether the given value is an integer.

3. parseFloat()
It converts the given string into a floating point number.

4. parseInt()
It converts the given string into an integer number.

5. toExponential()
It returns the string that represents exponential notation of the given number.

6. toFixed()
It returns the string that represents a number with exact digits after a decimal point.

7. toPrecision()
It returns the string that represents a number of specified precision.

8. toString()
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.

```
Boolean b = new Boolean(value);
```

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.

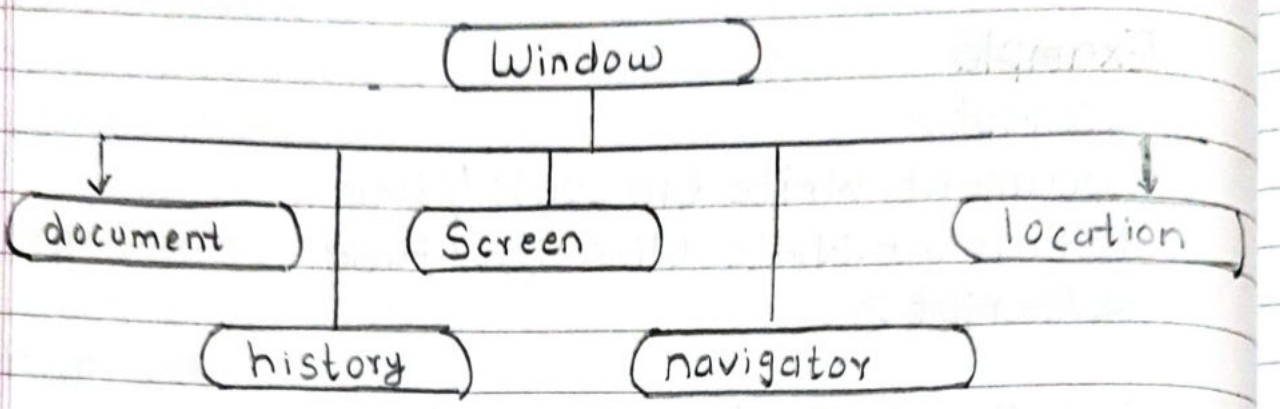
JavaScript Boolean Methods

Method	Description
to Source()	returns the source of Boolean object as a string.
to String()	Converts Boolean into string.
Value Of()	Converts other type into Boolean.

Browser Object Model

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

```
Window.alert ("hello javatpoint");
is same as:
alert ("hello javatpoint");
```



Window Object

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

Example of alert() in javascript

```
<script type = "text/javascript">
function msg(){
alert ("Hello Alert Box");
}
</script>
<input type = "button" value = "click" onclick = "msg()"/>
```

Output click

Example of confirm() in javascript

```
<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()"/>
```

Output delete record

Example of prompt() in javascript

```
<script type = "text/javascript ">
function msg(){
Var v = prompt ("Who are you?");
alert ("I am " + v);
}
</script>
< input type = "button "value = "click" onclick = " msg ()"/>
```

Output click

Example of open() in javascript

```
<script type = "text/javascript">
function msg(){
open ("http://www.javatpoint.com");
}
</script>
<input type = "button" value = " javat point" onclick = "msg()"/>
```

Output javat point

Example of setTimeout() in javascript

```
<script type = "text/javascript">
function msg(){
setTimeout(
function(){
alert("Welcome to Javatpoint after 2 seconds")
}, 2000);
}
</script>
<input type = "button" value = "click" onclick = "msg()"/>
```

Output click

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.

```
Window.history
```

Or,

```
history
```

Property of JavaScript Object

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

Methods of JavaScript history object

1. forward()
   loads the next page.

2. back()
loads the previous page.

3. go()
loads the given page number.

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.

Window.navigator
Or
navigator

Property of JavaScript navigator object

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 Netscape and Firefox 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

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

JavaScript Screen Object

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

```
window.screen
```

Or;

```
screen
```

Property of JavaScript Screen Object

NO.	Property	Description
1.	width	returns the width of the screen.
2.	height	returns the height of the screen.
3.	availWidth	returns the available height.
4.	availHeight	returns the availableheight.

5.	colorDepth	returns the color depth.
6.	pixelDepth	returns the pixel depth.

Example

```
<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.vav
document.Writeln("<br/>screen color Depth:"+screen color Depth);
document.Writeln("<br/>screen pixelDepth:"+screen pixelDepth;)
</script>
```

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.

Window.document
is same as
document

Methods of document object

We can access and change the contents of document by its methods.

Method	Description
• Write("String")	Writes the given string on the document.
• Writeln("string")	Writes the given string on the document with newline character at the end.
• getElementById()	returns the element having the given id value.
• getElements ByName()	returns all the elements having the given name value.
• getElements By Tag Name()	returns all the elements having the given tag name.
• getElements By Class Name()	returns all the elements having the given class name.

Accessing field value by document object

Example

```
<script type = "text/javascript ">
function printvalue(){
Varname = document.form1.name.value;
alert("Welcome;" +name);
}
</script>
<form name = "form1">
Enter Name; <input type = "text" name = "name"/>
<input type = "button "onclick= "printvalue()"value="printname"/>
</form>
```

Output

Enter Name: [ ] Print name

JavaScript - document.getElement By Id() 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.getElement By ID() method to get value of the input text. But we need to define id for the input field.

Example

```
<script type = "text/javascript">
function getcube(){
var number = document.getElement ByID("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>
```

Output :

Enter No: [ ]

Cube

Get Elements By Class Name ()

The getElements By Class Name() 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 getElement By Class Name() method or any particular element, it Will search the Whole document and Will

return only thouse elements which match the specified or given class name.

```
Varele = document.getElementsByClassName('name');
```

Example

```
<html>
<head><h5> DOM Methods </h5></head>
<body>
<div class = "class" >
This is a simple class implementation
</div>
<Script type = "text/javascript">
Varx = document.getElement ByClassName('class');
document. Write ("On Calling x, It will return an arrsy-
    like object: <br>"+x);
</script>
</body>
</html>
```

Output:

DOM Methods
This is a simple class implementation
On calling x. It will return an array-like object:
[Object HTML Collection]

JavaScript - document.getElementsByName() method

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

```
document.getElements By Name("name")
```

Example

```
<script type = "text/javascript">
function totalelements()
{
Var allgenders = document.getElementsByName ("gender");
alert ("Total Genders."+ allgenders. length);
}
</script>
<form>
make: <input type = "radio" name = "gender" value = "male" >
Female: <input type = "radio" name = "gender" value = "female">
<input type = "button" onclick "totalelements()" value = "Total Genders ">
</form>
```

Output

Male: 0 Female: 0 | Total Genders |

JavaScript - document.get Elements By Tag Name() method

The document.getElements By Tag Name () method returns all the element of specified tag name.

```
document.getElements By Tag Name ("name")
```

Example

```
<script type = "text/javascript">
function countpara(){
Var totalpara = document.getElementsByTagName("p");
alert("total ptags are:"+ totalpara.length);
}
</script>
<P> This is a paragraph</p>
<P> Here we are going to count total number of
  Paragraphs by getElements By Tag Name()method.</p>
<P> Let's see the simple example</p>
<button onclick = "countpara()">Count paragraph</button>
```

Output

This is a pragraph.
Here we are going to count total number of paragraphs by getElementsByTagName() method.
Let's see the simple example.

Count Paragraph

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

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

```
function Showcommentform(){
Var data = "Name; <input type = 'text' name = 'name'><br>Comment:
<br>textarea rows = '5' cols = '80'></textarea>
<br><input type = 'submit' Value = 'Post Comment'>";
documet.getElementbyId('mylocation'). innerHTML = data;
}
</Script>
<form name = "my Form">
<input type = "button" value: "comment" onclick = "Showcommentform()">
<divid = "mylocation"></div>
</form>
```

Output

comment

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.

Example

```
<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>
<body>
<form name = "myform" method = "post" action = "abc.jsp"
   onsubmit = "return validate form()">
Name: <input type = "text" name = "name" = "password"><br/>
Password: <input type = "password" name = "password"><br/>
 <input type = "submit" value = "register">
  </form>
```

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).

Example

```
<script>
function validate email()
```

```
{
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 email address \n atpostion:"+
   atposition+"\n dotposition."+ dotposition);
return false;
  }
}
</script>
<body>
<form name = "myform" method = "Post" action = "#"
      onsubmit = "return Validateemail();">
Email: <input type = "text" name = "email"><br/>
     <input type = "submit" value = "register">
      </form>
```

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 declaration
- 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.

Example

```
<script>
//Declaring class
 Class Employee
{
//Initializing an object
  constructor(id, name)
 {
 this.id = id;
thh1.name = name;
}
//Declaring method
  detail()
 {
document.Writein(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>
```

Output: 101 Martin Roy
102 Duke William

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.

example

```
<script>
Var emp = class {
 constructor(id, name){
 this.id = id;
 this.name = name;
 }
};
document.Writein(emp.name)
 </script>
```

Output:

emp

JavaScript Objects

A javascript object is an entity having state and behavior. JavaScript is an object-based language. Everything is an object in JavaScript.

Javascript is template based not class based. Here, we don't

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.

example

```
<script>
Var emp = class {
 constructor(id, name){
 this.id = id;
 this.name = name;
 }
};
document.Writein(emp.name)
 </script>
```

Output:

emp

JavaScript Objects

A javascript object is an entity having state and behavior 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
3. By using an object constructor

1) JavaScript Object by object literal

Object = {property1.Value1.property2.Value2....propertyN.valueN}

Example

```
<script>
emp = {id:102,name."Shyam Kumar", Salary:40000}
document.Write(emp.id+" " +emp.name+" "+emp.salary);
</script>
```

Output

102 Shyam Kumar 40000

2) By Creating instance of Object

```
Var objectname = new Object();
```

Example

```
<script>
var emp = new object();
 emp.id = 101;
```

```
emp.name = "Ravi Malik"
emp.salary = 50,000;
document.Write(emp.id + " " + emp.name + " " + emp.Salary);
</script>
```

Output

101 Ravi 50000

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 the keyword.

example

```
<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>
```

Output

103, Vimal Jaiswal 30000

JavaScript Object Methods

1. Object.assign()

This method is used to copy enumerable and own properties from a source object to a target object.

2. Object.create()
This method is used to create a new object with the specified prototype object and properties.

3. Object.defineProperty()
This method is used to describe some behavioral attributes of the property.

4. Object.defineProperties()
This method is used to create or configure multiple object properties.

5. Object.entries()
This method returns an array with arrays of the key.value pairs.

6. Object.freeze()
This method prevents exisiting properties from being removed.

7. Object.getOwnPropertyDescriptor()
This method returns a property descriptor for the specified property of the specified object.

8. Object.getOwnPropertyDescriptors()
This method returns all own property descriptors of a given objects.

9. Object.getOwnPropertyNames()
This method returns an array of all properties found.

10. Object.getOwnPropertySymbols()
This method returns an array of all own symbol key properties.

11. Object.getPrototypeOf()
This method returns the prototype of the specified object.

12. Object.is()
This method determines whether two values are the same value.

13. Object.isExtensible()
This method determines if an object is extensible.

14. Object.isFrozen()
This method determines if an object was frozen.

15. Object.isSealed()
This method determines if an object is sealed.

16. Object.keys()
This method returns an array of a given object's own property names.

17. Object.preventExtensions()
This method is used to prevent any extensions of an object.

18. Object.seal()
This method prevents new properties from being added and marks all existing properties as non-configurable.

19. Object.setPrototypeOf()
This methods sets the prototype of a specified object to another object.

20. Object.values()
This method returns an array of values.

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.

```
Class Name.prototype.methodName
```

Prototype chaining

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

Example

```
<script>
function Employee (firstName, lastName)
  {
this.firstName = firstName;
 this.lastName = lastName;
}
 Employee.prototype.fullName = function()
```

```
{
return this. first Name +" " + this last Name;
}
Var employee1 = new Employee ("Martin", "Roy")
Var employee2 = new Employee ("Duke", "William")
document. Writeln (employee1. fullName() + "<br>");
document. Writeln (employee2. full Name());
</script>
```

Output

Martin Roy
Duke William

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

```
<script>
Class Employee {
 Constructor() {
 this. id =101;
this. name = "Martin Roy";
 }
}
Var emp = new Employee();
document. Writeln (emp.id +" " +emp.name);
 </script>
```

Output

101 Martin Roy

Constructor Method Example : super KeyWord

```
<Script>
Class CompanyName
 {
 Constructor()
 {
this. company ="Javatpoint";
 }
 }
 Class Employee extends Company Name {
 Constructor (id, name) {
 Super();
 this.id = id;
 this. name = name;
 }
}
Var emp = new Employee (1, "John");
```

```
document.Writeln(empid+" "+emp.name+" "+emp.company);
</script>
```

Output

1 John Javatpoint

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 all 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.

Example

```
<script>
class Test
```

```
{
static display()
{
return "Static method is invoked"
}
}
document.writeln(Test display());
</script>
```

Output

Static method is invoked

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data with the functions acting on the data. It allows us to control the data and validate. 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 the 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.

Example

```
<script>
Class Student
{
Constructor()
{
Var name;
Var marks;
}
get name()
{
return this.name;
}
SetName(name)
{
this.name = name;
}
get Marks()
{
return this.marks
}
setMarks(Marks)
{
this.marks = marks
}
}
Var stud = new Student();
Stud.setName("John");
Stud.setMarks(80);
document.Writeln(stud.getName()+" "+Studgetmarks());
</script>
```

Output: John 80

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 parents 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

```
<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>
```

Output:
Current date: 31 - 8 - 2018

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.

Example

```
<Script>
 ClassA
  {
   display()
   {
   document.Writeln("A is invoked");
   }
  }
   Class B extends A
   {
   }
  VarB = newB()
   b.display();
   </script>
```

Output:
A is invoked

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 reduse the duplication of code.

Example

```
<script>
//Creating a constructor function
 function Vehicle()
 {
  this.vehicleName = "Vehicle Name";
  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>
```

Output:
Vehicle is: Honda

Java Script 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 users 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.

```
document.cookie = "name = value";
```

Example

```
<!DOC TYPE html>
<html>
<head>
</head>
<body>
<input type="button" value = "setCookie" onclick=
    " setCookie()">
<input type = "button" value = "get Cookie" 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>
```

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 upto the Specified time. Here, time is given in seconds.
Path	It expands the scope of the cookie to all Pages of a website.
domain	It is used to specify the domain for which the cookie is vaild.

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 rppresents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Example

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

```
<body>
<Input type = "button" value = "set Cookie" onclick =
      " set cookie()">
<input type = "button" value = "get cookie" Onclick=
      " get Cookie()">
 <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 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 then store in a cookie.
- For each name-value pair, use a separate cookie.

Examples to store Name-Value Pair in a cookie

Example

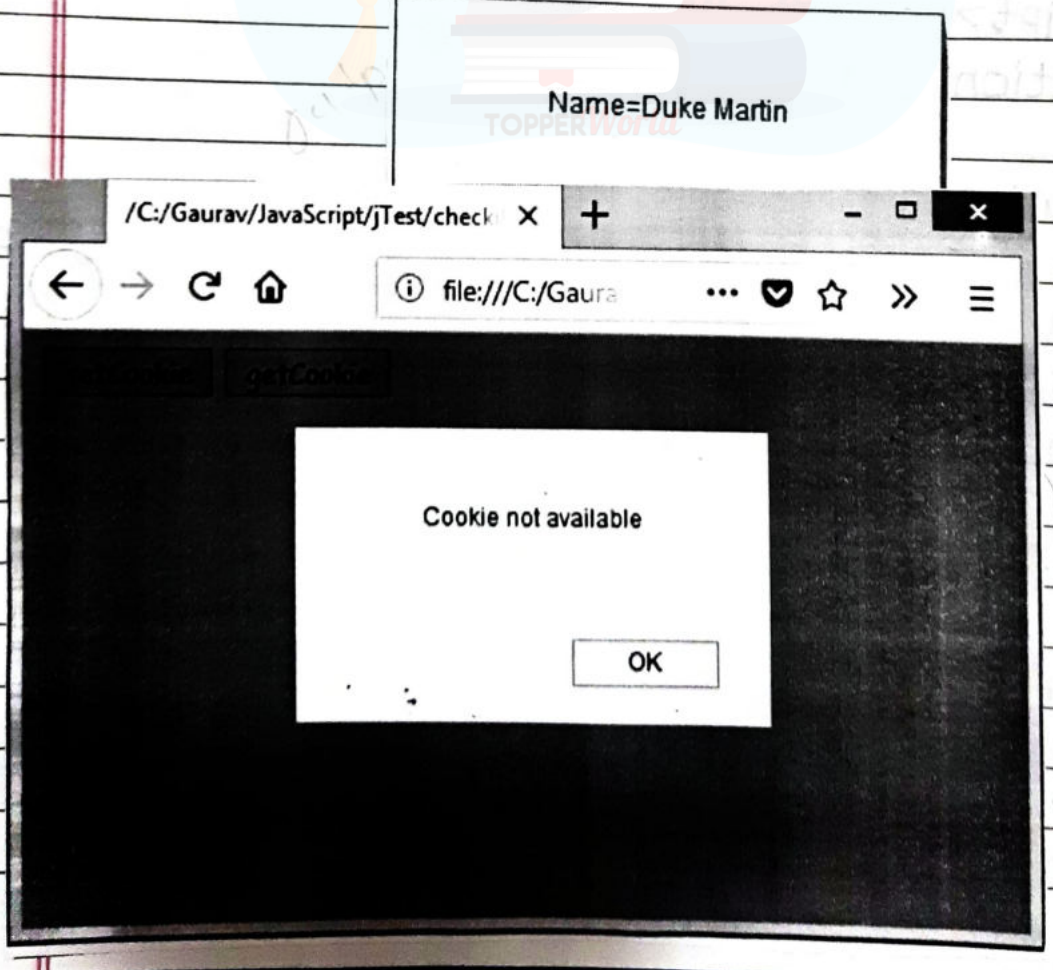
```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
Name : <input type = "text" id = "name"><br>
Email : <input type = "email" id = "email"><br>
Course : <input type = "text" id = "course"><br>
<input type = "button" value = "Set Cookie" onclick = "SetCookie()">
<input type = "button" value = "Get Cookie Onclick = "getCookie()">
<Script>
function setCookie()
{
//Declaring 3-key value pairs
var info = "Name = " + document.getElement By Id("name").
Value + "Email = " + document.get Element B
//Providing all 3 key-value pairs to a Single Cookie
document.cookie = info;
}
function get cookie()
{
if(document.cookie.length != 0)
{
//invoking key-value pair stored in a cookie
alert(document.cookie);
```

```
    }
    else
    {
    alert ("cookie not available")
    }
}
</script>
<body>
<html>
```

Output

Name: Duke Martin
Email: duke@gmail.com
Course: JavaScript
Set Cookie | Get Cookie

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



Deleting a Cookie in JavaScript

Different Ways to delete a Cookie

There 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.

Example

```
<!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>
```

JavaScript Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers. For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
Click	Onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element

mouseout	onmouseout	When the cursor of the mouse leaves an element.
mousedown	onmousedown	When the mouse button is pressed over the element.
mouseup	onmouseup	When the mouse button is released over the element.
mousemove	On mousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
key down & keyup	onkeydown & On keyup	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

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language = "Javascript" type = "text/Javascript">
<!--
   function clickevent()
   {
      document.Write("This is Java Tpoint");
   }
   //-->
</script>
<form>
<input type = "button" onclick = "clickevent()" value =
      "Who's this ?" />
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> JavaScript Events </h1>
</head>
<body>
<script language = "Javascript" type = "text/Javascript">
   <!--
```

```
    function mouseoverevent()
    {
      alert("This is JavaTpoint");
    }
    //-->
  </script>
  <p onmouseover="mouseoverevent()">Keep cursor over me</p>
  </body>
  </html>
```

Focus Event

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

Keydown Event

```
<html>
<head>Javascript Events</head>
<body>
<h2>Enter something here</h2>
```

```
<input type = "text" id = "input1" onkeydown =
    "keydownevent()" />
<Script >
<!--
  function keydownevent()
  {
    document.getElementById("input1");
     alert ("Pressed a key ");
   }
//-->
</script>
</body>
</html>
```

JavaScript addEventListener()

The addEventListener() method is used to attach an event handler to a particular element. It doesnot override the existing event handlers. Events are said to be an essential part of the JavaScript. A Webpage 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.

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

Parameter Values

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

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.

Example

```
<!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 javaTpoint.com";
}
</script>
</body>
</html>
```