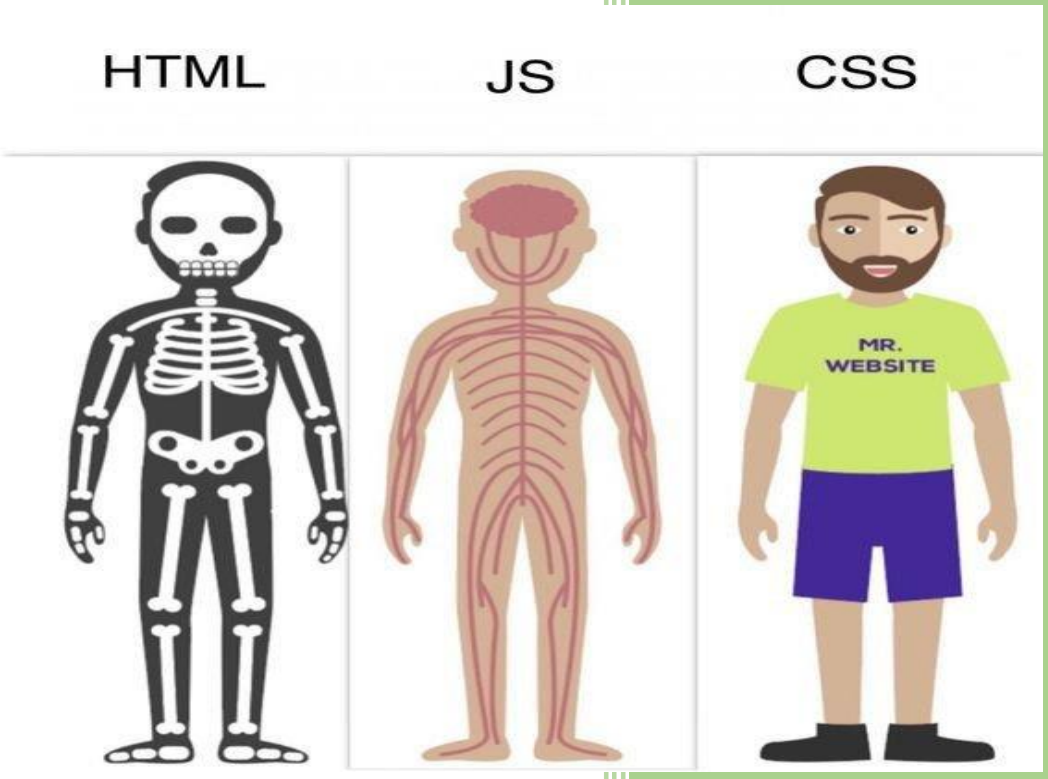
2022

JavaScript



Santosh Kumar

12/16/2022

What is JavaScript?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The advantages of using JavaScript are:

- Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces: You can use JavaScript to include such items as drag and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

JavaScript Syntax

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
    JavaScript code
</script>
```

The script tag takes two important attributes:

- **Language**: This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type**: This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So, your JavaScript syntax will look as follows.

```
<script language="javascript" type="text/javascript">
    JavaScript code
</script>
```

Your First JavaScript Program

```
<html>
        <head></head>
         <body>
             <script language="javascript" type="text/javascript">
                  document.write ("Hello World!")
             </script>
         </body>
</html>
```

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
     num1 = 15
     num2 = 62
</script>
```

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So, the identifiers Time and TIME will convey different meanings in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments. Thus:

- Any text begin with // is treated as a single line comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example

```
<script language="javascript" type="text/javascript">
    <!--
    // This is a comment. It is similar to comments in C++
    /*
       * This is a multiline comment in JavaScript
       * It is very similar to comments in C Programming
    */
    //-->
</script>
```

Ways to include JavaScript in an HTML Page

There is a flexibility given to include JavaScript code anywhere in an HTML document. However, the most preferred ways to include JavaScript in an HTML file are as follows:

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

JavaScript in <head>...</head> Section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows.

Example

```
<html>
    <head>
        <script type="text/javascript">
            function display() {
                alert("Hello, World")
            }
        </script>
    </head>
    <body>
        Click here for the result
        <input type="button" onclick="display()" value="Click Here" />
    </body>
</html>
```

JavaScript in <body>...</body> Section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript.

Example

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            document.write("Hello World")
        </script>
        <p>This is web page body </p>
    </body>
```

```
</html>
```

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows.

```
<html>
    <head>
        <script type="text/javascript">
            function show() {
                alert("Hello, World")
            }
        </script>
    </head>
    <body>
        <script type="text/javascript">
            document.write("Hello, World")
        </script>
        <input type="button" onclick="show()" value="Click Me" />
    </body>
</html>
```

JavaScript in External File

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

```
<html>
    <head>
        <script type="text/javascript" src="filename.js" ></script>
    </head>
    <body>
        .....
    </body>
```

```
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use show function in your HTML file after including the filename.js file.

```
function show() {
        alert("Hello, World")
}
```

JavaScript Basic

Keywords

JavaScript defines a list of reserved keywords that have specific uses. Therefore, you cannot use the reserved keywords as identifiers or property names by rules.

The following table shows the JavaScript reserved words defined

break	case	var	catch
continue	debugger	this	default
else	export	instanceof	extends
function	if	for	import
new	return	do	super
throw	try	const	null
void	while	yield	with
class	delete	typeof	finally
in	switch		

Identifier

An identifier is a name you choose for variables, parameters, functions, classes, etc.

Rules for defining identifier's:

1. An identifier name consist of letter (a-z, or A-Z), numbers (0-9), an underscore(_), or a dollar sign ($).
2. It cannot start with number.
3. It cannot be a keyword.

Variable

Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

JavaScript is a dynamically typed language. It means that a variable doesn't associate with a type. In other words, a variable can hold a value of different types.

Declare a variable

To declare a variable, you use the var keyword followed by the variable name as follows:

```
var message;
```

A variable name can be any valid identifier.

In ES6, you can use the let keyword to declare a variable like this:

```
let message;
```

It's a good practice to use the let keyword to declare a variable.

Initialize a variable

Once you have declared a variable, you can initialize it with a value.

To initialize a variable, you specify the variable name, followed by an equals sign (=) and a value.

For example:

```
let message;

message = "Hello";
```

Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables**: A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables**: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Constant

A constant holds a value that doesn't change.

To declare a constant, you use the const keyword. When defining a constant, you need to initialize it with a value.

For example:

```
const workday = 5;
```

Once defining a constant, you cannot change its value.

Difference between var, let and const

var	let	const
It is available right from the beginning when the JavaScript was introduced.	It is a new way to declare variables in JavaScript, starting from ES6.	const is used to store a value that will not be changed throughout the execution of the script. It is also introduced recently in ES6.
It has a global/function scope.	It has block scope.	It also has block scope.
Can be updated or re-declared in its scope.	We can't re-declare them.	const represents a constant value, so it can't be updated or re-declared.

Data Types

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript has the two types of data types:

1. Primitive data types:
    - null
    - undefined
    - boolean
    - number
    - string
    - bigint – available from ES2020
2. complex data type
    - object

undefined type

The undefined type is a primitive type that has only one value undefined. By default, when a variable is declared but not initialized, it is assigned the value of undefined.

null type

The null type is the second primitive data type that also has only one value null. For E

Example: let obj = null;

number type

JavaScript uses the number type to represent both integer and floating-point numbers.

boolean type

The boolean type has two literal values: true and false in lowercase.

string type

A string is a sequence of zero or more characters. A string literal begins and ends with either a single quote(') or a double quote (").

bigint type

The bigint type represents the whole numbers that are larger than 253 – 1. To form a bigint literal number, you append the letter n at the end of the number:

Example : let pageView = 9007199254740991n;

object type

In JavaScript, an object is a collection of properties, where each property is defined as a key-value pair.

Following example defines the person object with two properties: firstName and lastName.

```
let person = {
   firstName: 'John',
   lastName: 'Doe'
};
```

JavaScript Operator's

Operator

Operator is a symbol which is used to perform mathematical and logical operations.

JavaScript support the following types of operators:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment / Decrement Operators
6. Ternary Operator

Arithmetic Operator

JavaScript support the following arithmetic operator:

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1

Relational Operator

Relational operators are used to check the relationship among two operands and return true or false.

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Logical Operator

Logical operators are used when you want to perform a task on the basis of more then one condition. JavaScript support the following logical operator.

Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

Assignment Operator

An assignment operator (=) assigns a value to a variable. The syntax of the assignment operator is as follows:

```
Let val = 23;
```

The following table illustrates assignment operators that are shorthand for another operator and the assignment:

Operator	Example	Equivalent Expression
=	$m = 10$	$m = 10$
+=	$m += 10$	$m = m + 10$
−=	$m -= 10$	$m = m - 10$
*=	$m *= 10$	$m = m * 10$
/=	$m /=$	$m = m/10$
%=	$m \% = 10$	$m = m\%10$
&=	$a \&= b$	$a = a \& b$
^=	$a \hat{} = b$	$a = a \hat{} b$

String Concatenation Operator

When working with JavaScript strings sometimes you need to join two or more strings together into a single string. Joining multiple strings together is known as concatenation.

The concatenation operator (+) concatenates two or more string values together and return another string which is the union of the two operand strings.

Example:

```
Let s1 = “Java”;
Let s2 = “Script”;
document.write(s1+s2);
```

Increment / Decrement Operator

Increment and decrement operator are used to increase or decrease the value of variable by 1.

Operators	Meaning	Example	Result
+ +	Increment	A = 10; A+ +	11
− −	Decrement	A = 10; A− −	9

Bitwise Operator

The bitwise operators are used to perform bitwise operations on operands.

Operator	Description	Example
&	Boolean AND operation on each bit of its integer arguments	(10==20 & 20==33) = false
|	It performs a Boolean OR operation on each bit of its integer arguments	(10==20 | 20==33) = false
^	This operator performs Bitwise XOR operation	(10==20 ^ 20==33) = false
~	It is a unary operator and operates by reversing all the bits in the operand	(~10) = -10
<<	Moves all the bits in its first operand to the left by the number of places specified in the second operand.	(10<<2) = 40
>>	The left operand’s value is moved right by the number of bits specified by the right operand.	(10>>2) = 2
>>>	This operator is just like the >> operator, except that the bits shifted in on the left are always zero.	(10>>>2) = 2

Ternary Operator

A ternary operator evaluates a condition and executes a block of code based on the condition.

Its syntax is:

condition ? expression1 : expression2

- The ternary operator evaluates the test condition.
- If the condition is true, expression1 is executed.
- If the condition is false, expression2 is executed.

The ternary operator takes three operands, hence, the name ternary operator. It is also known as a conditional operator.

Example

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            var a = 10;
            var b = 20;
            document.write ("((a > b) ? 100 : 200) => ");
            result = (a > b) ? 100 : 200;
            document.write(result);
        </script>
    </body>
</html>
```

Miscellaneous Operators

- **typeof**
  JavaScript typeof operator return valid data type identifiers as a string of given expression. typeof operator return six possible values: "string", "number", "boolean", "object", "function", and "undefined".
  Syntax:

  typeof expression
  typeof(expression)

  Example:

  Let age = 23;

  Let name = “Santosh Kumar”

```
document.write(typeof(age));        // print number
document.write(typeof(name));       // print string
```

- delete
  JavaScript delete operator deletes object property or remove specific element in array. If delete is not allow (you can't delete if element not exist, array element undefined etc..) then return false otherwise return true.

- instanceof
  JavaScript instanceof indicate boolean result, return true, If object is an instance of specific class.
- new
  JavaScript new operator to create an instance of the object.
- this
  JavaScript this operator represents current object.
- in
  JavaScript in operator return boolean result if specified property exist in object.

Control Statement's

Control Statements

Statements that are used to control the flow of execution in a script are known as control statements. Control statements are used along with compound statements (a set of statements enclosed in braces).

Types of control statements

1. Conditional Statement
2. Iterative Statement (Loop)

Conditional Statements

Conditional statements are used to decide the flow of execution based on different conditions. If a condition is true, you can perform one action and if the condition is false, you can perform another action.

Different Types of Conditional Statements

There are mainly three types of conditional statements in JavaScript.

1. If statement
2. Switch statement

If statement

The if statement is a powerful decision-making statement. The if statement is used to control the flow of execution of statements.

Form of if statements:

1. **Simple If statement**
   It is used when you want to execute something based on some condition.
   Syntax:

```
if(text-expression)
{
        // code to be executed if condition is true
}
```

   If the if expression evaluates to true, the block following the if statement or statements are executed.

Example:

```
<script>
        var mySal = 1000;
        var yourSal = 500;
        if( mySal > yourSal)
        {
                alert("My Salary is greater than your salary");
        }

</script>
```

2. **If-else statement**

if-else is a two-way decision-making statement. The syntax of if……else statements is:

```
if (test-expression)
{
        //statement-if-block;
}
else
{
        statement-else-block;
 }
```

IF the if expression evaluates to true, the block following the if statement or statements are executed.

The else statement is optional. It is used only if a statement or a sequence of statements are to be executed in case the if expression evaluates to false.

Example:

```
<script>
        var mySal = 500;
        var yourSal = 1000;
        if( mySal > yourSal)
        {
                alert("My Salary is greater than your salary");
        }
        else
        {
                alert("My Salary is less than or equal to your salary");
        }
</script>
```

3. If…Else…if statement

If else if statements are used to execute when you want to perform the task on more the two conditions. The general syntax of if..else…if statement is:

```
if (test-expression 1)
{
        Statement(s) to be executed if test-expression 1 is true
}
else if (test-expression 2)
{
        Statement(s) to be executed if test-expression 2 is true
}
else if (test-expression 3)
{
        Statement(s) to be executed if test-expression 3 is true
}
…….
…….
…….
else
{
        Statement(s) to be executed if no test-expression is true
}
```

Example:

```
<script>
        var mySal = 500;
        var yourSal = 1000;
        if( mySal > yourSal)
        {
                alert("My Salary is greater than your salary");
        }
        else if(mySal < yourSal)
        {
                alert("My Salary is less than your salary");
        }
        else if(mySal == yourSal)
        {
                alert("My Salary is equal to your salary");
        }
</script>
```

Switch statement

The switch statement allows to make a decision from the number of choices.

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default case will be executed.

Syntax of switch statement:

```
switch (Expression) {
    case labe1:
        statements;
        break;
    case labe2:
        statements;
        break;
    case labe3:
        statements;
        break;
    …….
    …….
    default:
        Statements;
}
```

Note: Each case inside the switch is end with break keyword.

Example:

```
<script>
    var day = prompt("Enter Day Number : ");
    switch(day)
    {
        case 1:
            document.write("Monday");
            break;
```

```
                case 2:
                        document.write("Tuesday");
                        break;
                case 3:
                        document.write("Wednesday");
                        break;
                case 4:
                        document.write("Thursday");
                        break;
                case 5:
                        document.write("Friday");
                        break;
                case 6:
                        document.write("Saturday");
                        break;
                case 7:
                        document.write("Sunday");
                        break;
                default:
                        document.write("Invalid Day Number !!!!");
        }
</script>
```

Iterative Statement (Loop)

In looping, a sequence of statements is executed until some condition for termination of the loop are satisfied.

Types of loops

1. While loop
2. For loop
3. For-in loop
4. Do-while loop

For Loop

The for loop is the most compact form of looping. It includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Syntax

The syntax of for loop is JavaScript is as follows:

```
for (initialization; test condition; iteration statement) {
    Statement(s) to be executed if test condition is true
}
```

Example:

```
<html>
<body>
<script type="text/javascript">
    var count;
    document.write("Starting Loop" + "<br />");
    for(count = 0; count < 10; count++){
```

```
        document.write("Current Count : " + count );
        document.write("<br />");
       }
       document.write("Loop stopped!");
       </script>
   </body>
   </html
```

Output:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

While Loop

while loop is used to execute code repeatedly till it satisfies a specified condition. Unlike for loop, while loop only requires condition expression.

Syntax

The syntax of while loop is:

```
while(condition expression)
{
   /* code to be executed
   till the specified condition is true */
```

```
}
```

Example:

```
<html>
    <body>
    <script type="text/javascript">
        var count = 0;
        document.write("Starting Loop ");
        while (count < 10){
            document.write("Current Count : " + count + "<br />");
            count++;
        }
        document.write("Loop stopped!");
    </script>
    <p>Set the variable to different value and then try...</p>
    </body>
</html>
```

Do while loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

Syntax of do-while loop:

```
do{
    Statement(s) to be executed;
} while (expression);
```

Example:

```
<html>
<body>
    <script type="text/javascript">
        var count = 0;
        document.write("Starting Loop" + "<br />");
        do{
            document.write("Current Count : " + count + "<br />");
            count++;
        }while (count < 5);
        document.write ("Loop stopped!");
    </script>
</body>
</html>
```

Output:

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!

For-in loop

The for...in loop is used to loop through an object's properties

Syntax:

The syntax of 'for..in' loop is:

```
for (variablename in object)
{
 statement or block to execute
}
```

Example:

```
<html>
<body>
    <script type="text/javascript">
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
        for (aProperty in navigator)
        {
            document.write(aProperty);
            document.write("<br />");
        }
        document.write ("Exiting from the loop!");
    </script>
</body>
</html>
```

Break Statement

Break keyword has two uses :

1. In switch Statement
2. In Looping Control

When the keyword break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an if.

Example:

```
<html>
<body>
	<script type="text/javascript">
		var x = 1;
		document.write("Entering the loop<br /> ");
		while (x < 20)
		{
			if (x == 5){
			break; // breaks out of loop completely
			}
		x = x + 1;
		document.write( x + "<br />");
		}
		document.write("Exiting the loop!<br /> ");
	</script>
</body>
</html>
```

Continue Statement

- The **continue statement** is used inside loops.
- When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.

Example:

```
<html>
<body>
	<script type="text/javascript">
		var x = 1;
		document.write("Entering the loop<br /> ");
		while (x < 10)
		{
			x = x + 1;
			if (x == 5){
				continue; // skill rest of the loop body
```

```
                }
            document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");
        </script>
</body>
</html>
```

Function In JavaScript

Function

In JavaScript, a function allows you to define a block of code, give it a name and then execute it as many times as you want.

There are mainly two advantages of JavaScript functions.

1. Code reusability: We can call a function several times so it saves 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.

A JavaScript function can be defined using function keyword.

Syntax:

```
//defining a function
    function <function-name>()
    {
       // code to be executed
    };

//calling a function
    <function-name>();
```

Example:

```
    function ShowMessage() {
       alert("Hello World!");
    }

    ShowMessage();
```

Function Parameters

A function can have one or more parameters, which will be supplied by the calling code and can be used inside a function. JavaScript is a dynamic type scripting language, so a function parameter can have value of any data type.

Example: Function Parameters

```
    function ShowMessage(firstName, lastName)
    {
       alert("Hello " + firstName + " " + lastName);
    }

ShowMessage("Steve", "Jobs");
ShowMessage("Bill", "Gates");
ShowMessage(100, 200);
```

Return Value

A function can return zero or one value using return keyword.

Example: Return value From a Function

```
function Sum(val1, val2)
{
   return val1 + val2;
}

var result = Sum(10,20); // returns 30
```

Anonymous Function

Anonymous function is useful in passing callback function, creating closure or immediately invoked function expression.

JavaScript allows us to define a function without any name. This unnamed function is called anonymous function. Anonymous function must be assigned to a variable.

Example: Anonymous Function

```
var showMessage = function ()
{
   alert("Hello World!");
}

showMessage();

var sayHello = function (firstName)
{
   alert("Hello " + firstName);
};

sayHello("Bill");
```

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.

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 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

Document Object Model (DOM)

- DOM is short form of Document Object Model.
- It is created by the browser when a web page is rendered.
- DOM of a webpage can be represented graphically in the form of a tree.
- JavaScript uses DOM to perform multiple tasks such as creating and deleting elements on the web page.
- Using DOM, we can Add or remove HTML elements.
- We can add CSS styles to the elements in the web page.
- We can add user interaction by attaching event listeners, like click or submit, to the elements.

Properties and Methods of DOM

In DOM, all HTML elements are defined as objects. So it will have both property and methods. Some important methods and properties are :

Methods :

1. write(string)
2. writeln(string)
3. getElementById()
4. getElementsByName()
5. getElementsByTagName()
6. getElementsByClassName()

Properties :

1. innerHTML – The text value of a node (element)
2. parentNode – The parent node of a node
3. childNodes – The child nodes of node

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

```
<script type="text/javascript">
function printvalue()
{
        var name=document.getElementById(“name”).value;
        alert("Welcome: "+name);
}
</script>

<form name="form1">
        Enter Name:<input type="text" id="name"/>
        <input type="button" onclick="printvalue()" value="print name"/>
</form>
```

JS 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. **By array literal**

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

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

   Example:

   ```
   <script>
       var emp=["Sonoo","Vimal","Ratan"];
       for (i=0;i<emp.length;i++)
       {
           document.write(emp[i] + "<br/>");
       }
   </script>
   ```

2. **By Array directly (new keyword)**

   The syntax of creating array directly is given below:

   ```
   var arrayname=new Array();  arrayname[0]=12;
   ```

   Example:

   ```
   <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>
   ```

3. **By using 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.

var arrayname=new Array(value1, value2, value3, ……. valueN);

Example:

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

Array Properties

length	It returns the length of an array.

Array Methods

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.
fill()	It fills elements into an array with static values.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.
join()	It joins the elements of an array as a string.
lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
shift()	It removes and returns the first element of an array.
sort()	It returns the element of the given array in a sorted order.

JS String

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";
   ```
2. **By string object**
   The syntax of creating string object using new keyword is given below:
   ```
   var stringname=new String("string literal");
   ```

String Methods

Methods	Description
charAt()	It provides the char value present at the specified index.
charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
toLowerCase()	It converts the given string into lowercase letter.
toUpperCase()	It converts the given string into uppercase letter.
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.

JS Date

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.

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)

Methods	Description
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local 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()	It sets the hour value for the specified date on the basis of local time.
setMilliseconds()	It sets millisecond value for the specified date on the basis of local time.
setMinutes()	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()	It sets the second value for the specified date on the basis of local time.

toString()	It returns the date in the form of string.
toTimeString()	It returns the time portion of a Date object.

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

```
Current Date and Time: <span id="txt"></span>
<script>
    var today=new Date();
    document.getElementById('txt').innerHTML=today;
</script>
```

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

```
<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>
```

JS Math Object

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

JavaScript Math Functions

Methods	Description
abs()	It returns the absolute value of the given number.
ceil()	It returns a smallest integer value, greater than or equal to the given number.
cos()	It returns the cosine of the given number.
floor()	It returns largest integer value, lower than or equal to the given number.
max()	It returns maximum value of the given numbers.
min()	It returns minimum value of the given numbers.
pow()	It returns value of base to the power of exponent.
random()	It returns random number between 0 (inclusive) and 1 (exclusive).
round()	It returns closest integer value of the given number.
sin()	It returns the sine of the given number.
sqrt()	It returns the square root of the given number

Example of Random function:

```
Random Number is: <span id="p2"></span>
<script>
    document.getElementById('p2').innerHTML=Math.random();
</script>
```

JS Number Object

JavaScript number object enables you to represent a numeric value. It may be integer or floating-point.

By the help of Number() constructor, you can create number object in JavaScript.
For example: var n=new Number(value);

JavaScript Number Constants:

Constant	Description
MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.

Number Object Methods

Methods	Description
isInteger()	It determines whether the given value is an integer.
parseFloat()	It converts the given string into a floating point number.
parseInt()	It converts the given string into an integer number.
toString()	It returns the given number in the form of string.

JS Boolean Object

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);

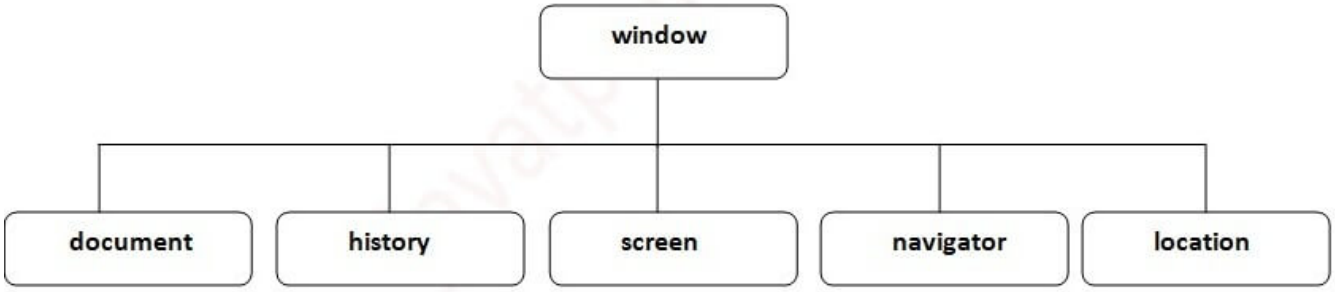
JavaScript Boolean Methods

Method	Description
toString()	converts Boolean into String.
valueOf()	converts other type into Boolean.

Browser Object Model

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.



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.

Methods of window object

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.
setInterval()	repeats a block of code at every given time interval.

clearInterval()	clears a timer set with the setInterval() method

JS History Object

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 object or directly.

Property of JavaScript history object

Property	Description
length	returns the length of the history URLs.

Methods of JavaScript history object

Method	Description
forward()	loads the next page.
back()	loads the previous page.
go()	loads the given page number.

JS Navigator Object

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

Property of JavaScript navigator object:

Property	Description
appName	returns the name
appVersion	returns the version
appCodeName	returns the code name
cookieEnabled	returns true if cookie is enabled otherwise false
userAgent	returns the user agent
platform	returns the platform e.g. Win32.
online	returns true if browser is online otherwise false.

JS Screen Object

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

Property of JavaScript Screen Object

Property	Description
width	returns the width of the screen
height	returns the height of the screen
availWidth	returns the available width
availHeight	returns the available height
colorDepth	returns the color depth
pixelDepth	returns the pixel depth.

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 syntax contains two components:

- Class declarations
- Class expressions

Class Declaration

A class keyword is used to declare a class with any particular name.

```
Ex:
class Student
{
        // Class Member
}
```

Class Declarations Example

```
<script>
        class Employee
        {
           constructor(id,name)
           {
                this.id=id;
                this.name=name;
           }
           detail()
           {
                document.writeln(this.id+" "+this.name+"<br>")
           }
        }

        var e1=new Employee(101,"Martin Roy");
        var e2=new Employee(102,"Duke William");
        e1.detail(); //calling method
        e2.detail();
</script>
```

Class Expression

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.

```
var stu = class {
        // class Member's
 };
```

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.

Example

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

JS Objects

A javaScript object is an entity having state and behavior (properties and method).

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. **By object literal**

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

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

Example

```
<script>
        emp={
                id:102,
                name:"Santosh Kumar",
                salary:90000}
        document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

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.

Example

```
<script>
        var emp=new Object();
        emp.id=101;
        emp.name="Ravi Malik";
        emp.salary=50000;
        document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

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(e.id+" "+e.name+" "+e.salary);
</script>
```

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.

```
<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>
```

JS 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
```

Example

```
<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>
```

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>
```

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.
- 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-1

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

Example-2

```
<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>
```

Encapsulation

In 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.

Example

```
<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>
```

Inheritence

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.
In 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 behaviour 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.

```
<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>
```

Example: Custom class

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

```
<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>
```

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>
        class A
          {
            display()
            {
              document.writeln("A is invoked<br>");
            }
          }
```

```
        class B extends A
          {
            display()
            {
              document.writeln("B is invoked");
            }
          }

        var ob=new B();
        ob.display();
</script>
```

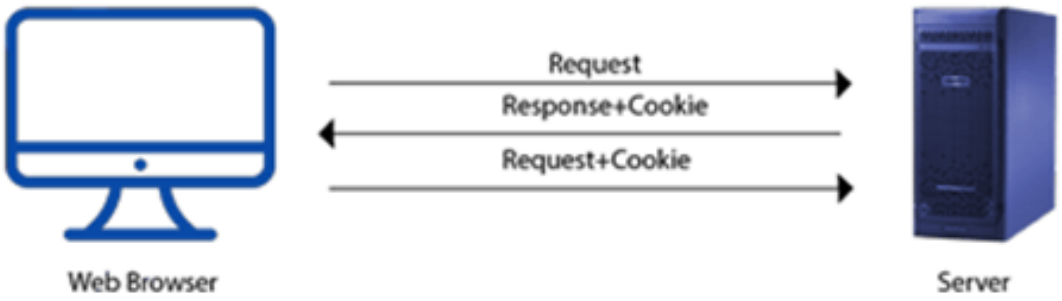
Cookies

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 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.
- 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, 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";
```

Cookie Attributes

JavaScript provides some optional attributes that enhance the functionality of cookies. Some attributes are:

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.

Example:

```
<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>
```

Deleting a Cookie

These are the following ways to delete a cookie:

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

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

```
<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=David Miller; expires=Mon, 16 Jan 2023 12:00:00 UTC";

}
function getCookie()
{
   if(document.cookie.length!=0)
   {
   alert(document.cookie);
   }
   else
   {
      alert("Cookie not avaliable");
   }
}
</script>
</body>
</html>
```

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

```
<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=Andy Smith;max-age=0";
        }
        function getCookie()
        {
           if(document.cookie.length!=0)
           {
           alert(document.cookie);
           }
           else
           {
              alert("Cookie not avaliable");
           }
        }

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

Exception Handling

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.

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:

- name: This is an object property that sets or returns an error name.
- message: This property returns an error message in the string form.

Standard built-in error types:

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.

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.

Syntax

```
try
{
        expression;  //code to be written.
}
catch(error)
{
        expression;  // code for handling the error.
}
```

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.
        }
```

Example

```
<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>
```