JavaScript

- JavaScript is the world most popular lightweight, interpreted compiled programming language.
- It is also known as scripting language for web pages. It is well-known for the development of web pages, many non-browser environments also use it.
- JavaScript can be used for Client-side developments as well as Server-side developments.
- It is designed for creating network-centric applications. It is complimentary to and integrated with Java.
- JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Why is it called JavaScript?

- When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help.
- But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

Javascript History

- The ECMAScript specification is a standardized specification of a scripting language developed by Brendan Eich of Netscape; initially named Mocha, then "LiveScript", and finally JavaScript.
- In December 1995, Sun Microsystems and Netscape announced JavaScript in a press release.
- In November 1996, Netscape announced a meeting of the ECMA International standards organization to advance the standardization of JavaScript.

The first edition of ECMA-262 was adopted by the ECMA General Assembly in June 1997. Several editions of the language standard have been published since then.

ECMA Script

ECMAScript is a JavaScript standard meant to ensure the interoperability of web pages across different web browsers. It is standardized by ECMA International according to the document ECMA-262. ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used for writing server applications and services using Node.js.

Versions

There are eleven editions of ECMA-262 published. Work on version 12 of the standard was finalized in June 2022(13TH Edition –latest version).

Features of JavaScript

There are following features of JavaScript:

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

Application of JavaScript

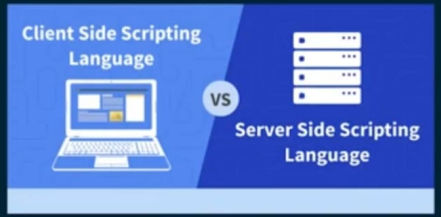
JavaScript is used to create interactive websites. It is mainly used for:

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

Difference between Java and Javascript

Java	JavaScript
Java is a strongly typed language and variables must be declared first to use in the program. In Java, the type of a variable is checked at compile-time.	JavaScript is a loosely typed language and has a more relaxed syntax and rules.
Java is an object-oriented programming language.	JavaScript is an object-based scripting language.
Java applications can run in any virtual machine(JVM) or browser.	JavaScript code used to run only in the browser, but now it can run on the server via Node.js.
Objects of Java are class-based even we can't make any program in java without creating a class.	JavaScript Objects are prototype-based.
Java program has the file extension ".Java" and translates source code into bytecodes which are executed by JVM(Java Virtual Machine).	JavaScript file has the file extension ".js" and it is interpreted but not compiled, every browser has the Javascript interpreter to execute JS code.
Java is a Standalone language.	contained within a web page and integrates with its HTML content.
Java has a thread-based approach to concurrency.	Javascript has an event-based approach to concurrency.
Java supports multithreading.	Javascript doesn't support multi-threading.
Java is mainly used for backend	Javascript is used for the frontend and backend both.
Java uses more memory	Javascript uses less memory.
Java requires a Java Development Kit(JDK) to run the code	Javascript requires any text editor or browser console to run the code

Types of Scripting Language



Types of Scripting Language

Basis for comparison	Client-side scripting Language	Server-side scripting Language
Basic	Works at the front end and script are visible among the users.	Works in the back end which could not be visible at the client end.
Processing	Does not need interaction with the server.	Requires server interaction.
Languages involved	HTML, CSS, JavaScript, etc.	PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etc...
Affect	Can reduce the load to the server.	Could effectively customize the web pages and provide dynamic websites.
Dependability	Client-side scripting depends upon the user's browser	Serve-side does not depend on the client.
Security	Insecure	Relatively secure.
Running	It runs on the end-users system.	It runs on the webserver.
Connectivity	The client-side does not connect to the database at the webserver.	The server-side helps connect with the database, which is already stored in the server database.
File Access	It does have any access to the files present in the web servers. But we have an option to upload files from the front end.	It has total access to the files which are stored in the web database server.

JavaScript can be added to your HTML file in two ways:

- Internal JS: We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or <body> tag according to the requirement.
- External JS: We can write JavaScript code in other file having an extension .js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

How to put a JavaScript into an HTML Page?

Syntax

```
<script type="text/javascript">
        (or)
<script language="JavaScript">
```

We can apply the script in both HEAD and BODY Tag. If we apply script in HEAD tag we can executing a script by calling a function. When we apply script to the BODY tag it shows the content of the web page.

The JavaScript command for writing some output to a page is document.write.

Syntax

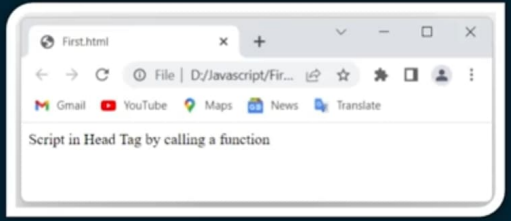
```
document.write("String")
```

If we no need to execute a script the <noscript> tag is used

Program(First.html)

```
<html>
 <head>
   <title>Script in Head Tag</title>
   <script type="text/javascript">
     function call()
     {
       document.write("Script in Head Tag by calling a function");
     }
   </script>
 </head>
 <body onload="call()">
 </body>
</html>
```

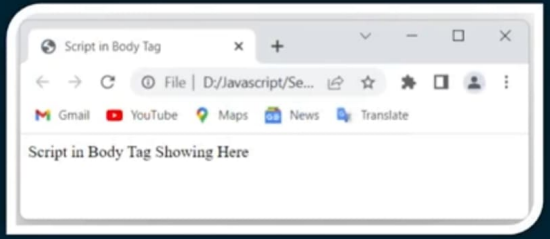
Output



Program(Second.html)

```
<html>
 <head>
   <title>Script in Head Tag</title>
 </head>
 <body>
   <script type="text/javascript">
       document.write("Script in Body Tag Showing Here");
   </script>
 </body>
</html
```

Output



JavaScript Output

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML -DOM changes.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

1. Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.

The id attribute defines the HTML element. The innerHTML property defines the HTML content.

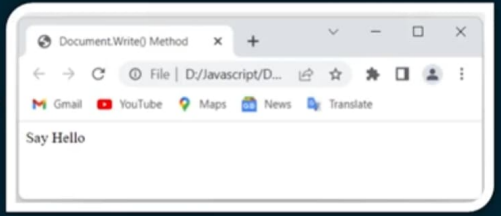
2. Using document.write()

For testing purposes, it is convenient to use document.write() method.

Program(DocumentDotWrite.html)

```
<!DOCTYPE html>
<html>
 <head>
   <title>Document.Write() Method</title>
 </head>
 <body>
  <script language="javascript">
    document.write("Say Hello");
  </script>
 </body>
</html>
```

Output:



Note: Using document.write() after an HTML document is loaded, will delete all existing HTML.

3. Using window.alert()

You can use an alert box to display data:

Program

```
<!DOCTYPE html>
<html>
 <head>
   <title>Alert Box</title>
 </head>
 <body>
  <script type="text/javascript">
    Window.alert("Say Hello");
  </script>
 </body>
</html>
```

Output:

4. Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

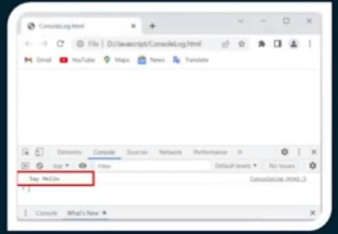
Program(ConsoleLog.html)

```
<!DOCTYPE html>
<html>
<body>
<script>
    console.log("Say Hello");
</script>
</body>
</html>
```

Open the Console in your browser

Right Click or button in your browser Click More Tools->Developer Tools->A Console Tab appear below the Browser.

Output



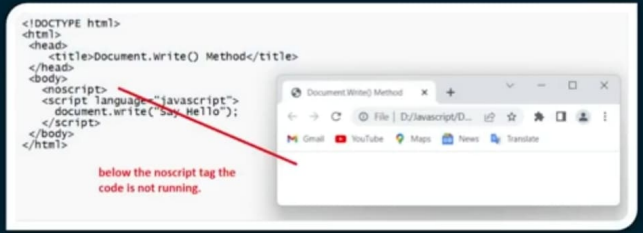
<noscript> tag

Suppose we does not want to run java script in the browser we apply <noscript> tag.

Example

```
<!DOCTYPE html>
<html>
 <head>
   <title> Document.Write() Method</title>
 </head>
 <body>
  <noscript>
  <script language="javascript">
    document.write("Say Hello");
  </script>
 </body>
</html>
```

Output



External Java Script

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

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

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

We apply external javascript in the html using src(source) attribute of the <script> tag.

Syntax

```
<script type="text/javascript" src="external_javascript_file">
  (or)
<script language="javascript" src="external_javascript_file">
```

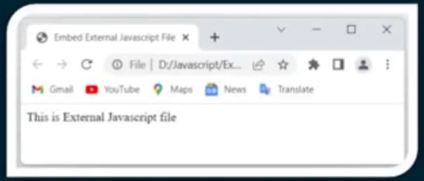
Program

```
<html>
 <head>
  <title>Embed External Javascript File</title>
 </head>
 <body>
  <script type="text/javascript" src="external.js">
  </script>
 </body>
</html>
```

external.js

```
  document.write("This is External Javascript file");
```

Output:



Advantages of External JavaScript

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

Disadvantages of External JavaScript

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

JavaScript Comment

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

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

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

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

Types of JavaScript Comments

There are two types of comments in JavaScript.

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

1. JavaScript Single line Comment

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

EXAMPLE

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

2. JavaScript Multi line Comment

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

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

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

EXAMPLE

```
<script>
  /* It is multi line comment.
  It will not be displayed */
  document.write("example of javascript multiline comment");
</script>
```

Variables

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

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

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

4 Ways to Declare a JavaScript Variable:

1. Using var
2. Using let
3. Using const
4. Using nothing

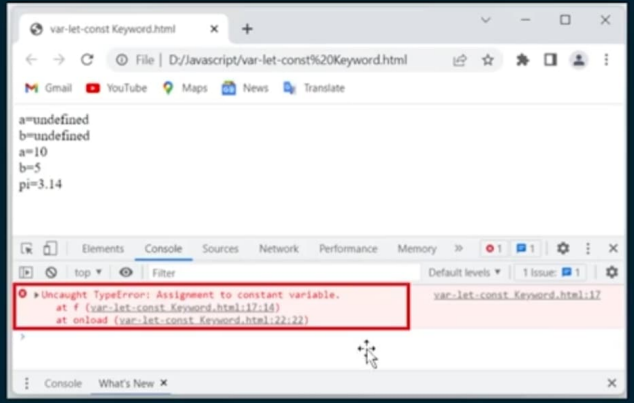
Differences between var, let, and const

Var	Let	const
The scope of a var variable is functional scope.	The scope of a let variable is block scope.	The scope of a const variable is block scope.
It can be updated and re-declared into the scope.	It can be updated but cannot be re-declared into the scope.	It cannot be updated or re-declared into the scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is "undefined".	It cannot be accessed without initialization, as it returns an error.	It cannot be accessed without initialization, as it cannot be declared without initialization.

Program

```
<html>
 <head>
  <title>var,let,const keyword</title>
  <script type="text/javascript">
     var a; //default value undefined [GLOBAL(or)FUNCTION SCOPE]
      function f()
      {
        let b; //default value undefined [LOCAL(or)BLOCK SCOPE]
        document.write("a="+a+"<br>");
        document.write("b="+b+"<br>");
        var a=10;  //can be redeclared and updated
        b=5;   //can be updated but not redeclared [Note: redeclare let b=5; means error]
        document.write("a="+a+"<br>");
        document.write("b="+b+"<br>");
        const pi=3.14;
        document.write("pi="+pi+"<br>");
        pi=8.14; //cannot be redeclared and updated
        document.write("pi="+pi+"<br>");
      }
   </script>
 </head>
 <body onload="f()">
 </body>
</html>
```

Output:


var-let-const Keyword.html
File | D:/Javascript/var-let-const%20Keyword.html
Gmail
YouTube
Maps
News
Translate
a=undefined
b=undefined
a=10
b=5
pi=3.14
Elements
Console
Sources
Network
Performance
Memory
top
Filter
Default levels
1 Issue: 1
Uncaught TypeError: Assignment to constant variable.
at f (var-let-const Keyword.html:17:14)
at onload (var-let-const Keyword.html:22:22)
var-let-const Keyword.html:17
Console
What's New

JavaScript Operators

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

For Example:

```
var sum=10+20;
```

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

There are following types of operators in JavaScript.

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

1. Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operand.

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

String Concatenation

The + operator performs concatenation operation when one of the operands is of string type. The following example demonstrates string concatenation even if one of the operands is a string.

Example: + Operator with String

```
var a = 5, b = "Hello ", c = "World!", d = 10;
a + b; //returns "5Hello "
b + c; //returns "Hello World!"
a + d; //returns 15
b + true; //returns "Hello true"
c - b; //returns NaN; - operator can only used with number
```

2. JavaScript Comparison(Relational) Operators

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

Operator	Description	Example
==	Is equal to	10==20 = false
===	strict equality comparison operator	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

Example

```
var a = 5, b = 10, c = "5";
var x = a;

a == c; // returns true

a === c; // returns false [strict equality comparison operator in JavaScript, which returns false for the values which are not of a similar type]

a == x; // returns true

a != b; // returns true

a > b; // returns false

a < b; // returns true

a >= b; // returns false

a <= b; // returns true
```

3. Logical Operators

In JavaScript, the logical operators are used to combine two or more conditions. JavaScript provides the following logical operators.

Operator	Description
&&	&& is known as AND operator. It checks whether two operands are non-zero or not (0, false, undefined, null or "" are considered as zero). It returns 1 if they are non-zero; otherwise, returns 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero or not (0, false, undefined, null or "" is considered as zero). It returns 1 if any one of of them is non-zero; otherwise, returns 0.
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition). !false returns true, and !true returns false.

Example: Logical Operators

```
var a = 5, b = 10;

(a != b) && (a < b); // returns true

(a > b) || (a == b); // returns false

(a < b) || (a == b); // returns true

!(a < b); // returns false

!(a > b); // returns true
```

4. Bitwise Operators

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

Operator	Description	Example	Same as	Result	Decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~ 5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2

5. 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

6. Ternary Operator

JavaScript provides a special operator called ternary operator ?: that assigns a value to a variable based on some condition. This is the short form of the if else condition.

Syntax:

<condition> ? <value1> : <value2>;

Example

var a = 10, b = 5;

var c = a > b? a : b; // value of c would be 10

var d = a > b? b : a; // value of d would be 5

7. 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.
,	The comma operator (,) evaluates each of its operands (from left to right) and returns the value of the last operand.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

Program (SpecialOperators.html)

```
<html>
 <head>
  <title>Special Operators</title>
 </head>
 <body>
  <script type="text/javascript">
   a=1;
   b=2.5;
   c=true;
   d="Hello";
   x={2,3};
   colors=new Array('red','green','blue');
   person = {firstName:"John",lastName:"Smith",age:50};
   document.write("type of a="+typeof(a)+"<br>");
   document.write("type of b="+typeof(b)+"<br>");
   document.write("type of c="+typeof(c)+"<br>");
   document.write("type of d="+typeof(d)+"<br>");
   document.write("type of x="+typeof(x)+"<br>");
   document.write("type of colors="+typeof(colors)+"<br>");
   document.write("'firstName' in person=");
   document.write('firstName' in person);
   delete person.age; //delete age in person
   document.write("<br>"+"'age' in person=");
   document.write('age' in person);
   document.write("<br>"+"'colors' instanceof Array=");
   document.write(colors instanceof Array);
   document.write("<br>"+"'person' instanceof Object=");
   document.write(person instanceof Object);
```

Control Statements

Conditional statements are used to perform different actions based on different conditions.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

1. The if Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

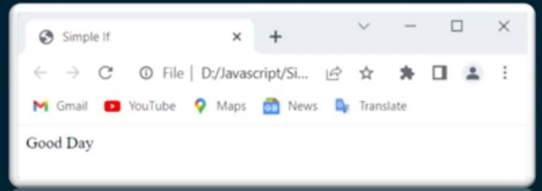
Syntax

```
if (condition) {
// block of code to be executed if the condition is true
}
```

Program(SimpleIf.html)

```
<html>
  <head>
    <title>Simple If</title>
  </head>
  <body>
    <script type="text/javascript">
      var hour=6;
      if(hour<18)
        document.write('Good Day');
    </script>
  </body>
</html>
```

Output:



2. The else Statement

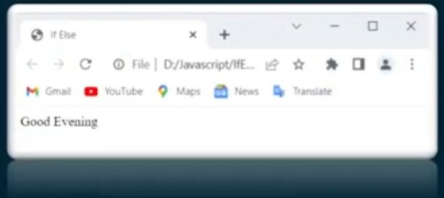
Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Program

```
<html>
  <head>
    <title>If Else</title>
  </head>
  <body>
    <script type="text/javascript">
      var hour=20;
      if(hour<18)
        document.write('Good Day');
      else
        document.write('Good Evening');
    </script>
  </body>
</html>
```

Output:



3. The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

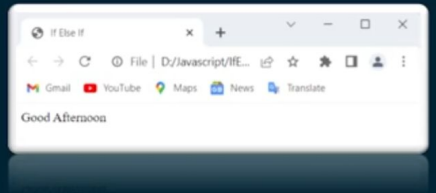
Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Program

```
<html>
 <head>
  <title>If Else If</title>
 </head>
 <body>
  <script type="text/javascript">
    const d = new Date();
    let hour = d.getHours();
    if(hour<=10)
      document.write('Good Morning');
    else if(hour>12 && hour<=16)
      document.write('Good Afternoon');
    else if(hour>16 && hour<=18)
     document.write('Good Evening');
    else
     document.write('Good Night');
  </script>
 </body>
</html>
```

Output:



4. Switch Statement

Use the switch statement to select one of many code blocks to be executed.

Syntax

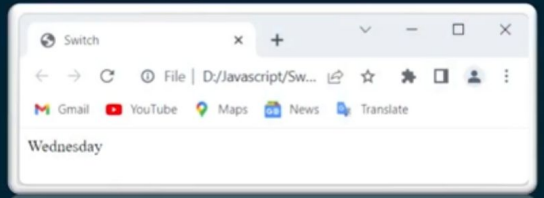
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

```
<html>
  <head>
    <title>Switch</title>
  </head>
  <body>
    <script type="text/javascript">
      const d = new Date();
      let day = d.getDay();
      switch(day)
      {
        case 0:
          document.write('Sunday');
          break;
        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;
        default:
          document.write('Invalid');
      }
    </script>
  </body>
</html>
```

OUTPUT:



JavaScript Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to do this.

In JavaScript we have the following looping statements:

1. while - loops through a block of code while a condition is true
2. do...while - loops through a block of code once, and then repeats the loop while a condition is true
3. for - run statements a specified number of times

1. while

The while statement will execute a block of code while a condition is true..

```
while (condition)
{
  code to be executed
}
```

2. do...while

The do...while statement will execute a block of code once, and then it will repeat the loop while a condition is true

```
do
{
  code to be executed
}while (condition)
```

3. for

The for statement will execute a block of code a specified number of times

```
for (initialization; condition; increment/decrement)
{
  code to be executed
}
```

```
<html>
 <head>
  <title>Loop</title>
 </head>
 <body>
  <script type="text/javascript">
   var x=1;
   document.write("Using While Loop"+"<br>");
   while(x<=5)
   {
    document.write(x+"<br>");
    x++;
   }
   x=10;
   document.write("Using Do...While Loop"+"<br>");
   do
   {
    document.write(x+"<br>");
    x++;
   }while(x<5);
   document.write("Using For Loop"+"<br>");
   for(x=1;x<=5;x++)
   {
    document.write(x+"<br>");
   }
  </script>
 </body>
</html>
```

JavaScript for...in Loop

Iterate (loop) over the properties of an object and values of array.

Syntax

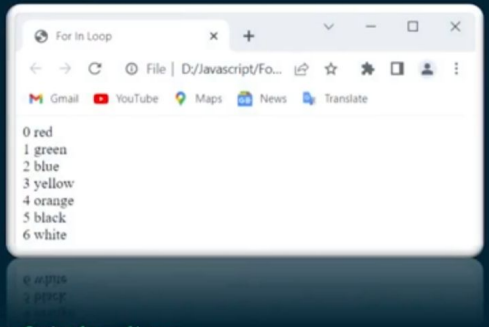
```
for (variable in object)
  statement
```

Program

```
<html>
 <head>
  <title>For In Loop</title>
 </head>
 <body>
  <script type="text/javascript">
   var colors=['red','green','blue','yellow','orange','black','white']
   for(i in colors)
   {
     document.write(i+" "+colors[i]+"<br>")
   }
  </script>
 </body>
</html>
```

Output:

OUTPUT:



JavaScript for...of Loop

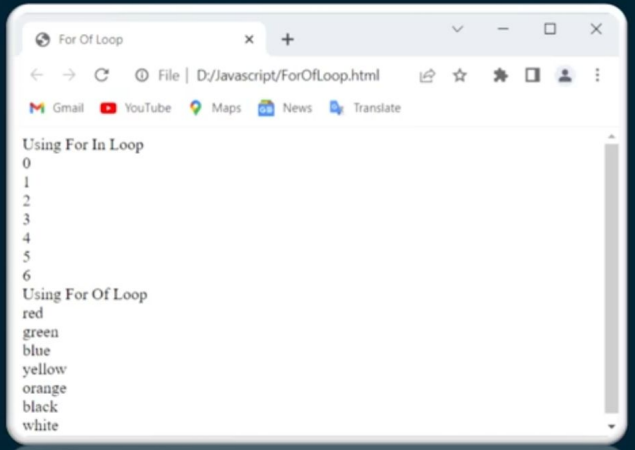
The for...of statement creates a loop iterating over iterable objects, including: built-in String, Array, array-like objects (e.g., arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. It invokes a custom iteration hook with statements to be executed for the value of each distinct property of the object.

Program

```
<html>
 <head>
  <title>For Of Loop</title>
 </head>
 <body>
  <script type="text/javascript">
    var colors=['red','green','blue','yellow','orange','black','white']
    document.write("Using For In Loop"+"<br>");
    for(i in colors)
    {
       document.write(i+"<br>")
    }
    document.write("Using For Of Loop"+"<br>");
    for(i of colors)
    {
       document.write(i+"<br>")
    }
  </script>
 </body>
</html>
```

Output:

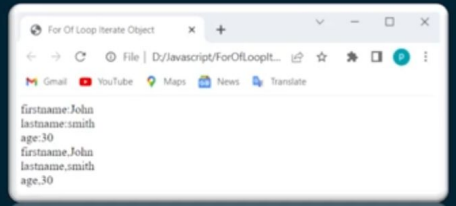
OUTPUT:


For Of Loop
File | D:/Javascript/ForOfLoop.html
Gmail
YouTube
Maps
News
Translate
Using For In Loop
0
1
2
3
4
5
6
Using For Of Loop
red
green
blue
yellow
orange
black
white

Iterating over an Object

```
<html>
 <head>
  <title>For Of Loop Iterate Object</title>
 </head>
 <body>
  <script type="text/javascript">
    const person={'firstname':'John','lastname':'smith','age':30};
    for(const i in person)
    {
      document.write(i+":"+person[i]+"<br>")
    }
    for(const i of Object.entries(person))
    {
      document.write(i+"<br>")
    }
  </script>
 </body>
</html>
```

Output:

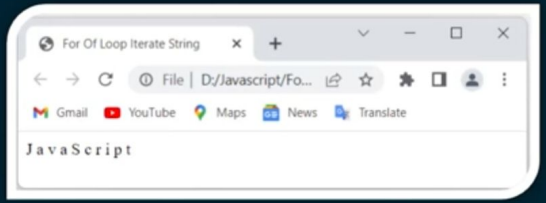


Iterating over a String

Program

```
<html>
  <head>
    <title>For Of Loop Iterate String</title>
  </head>
  <body>
    <script type="text/javascript">
      var msg="JavaScript";
      for(i of msg)
      {
        document.write(i+" ")
      }
    </script>
  </body>
</html>
```

Output

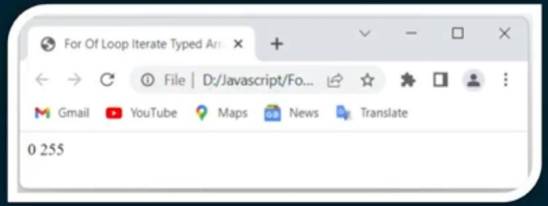


Iterating over a TypedArray

Program

```
<html>
  <head>
    <title>For Of Loop Iterate Typed Array</title>
  </head>
  <body>
    <script type="text/javascript">
      var msg=new Uint8Array([0x00, 0xff]);
      for(i of msg)
      {
         document.write(i+" ")
      }
    </script>
  </body>
</html>
```

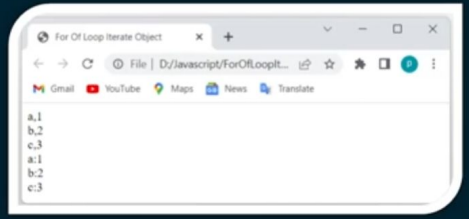
Output:



Iterating over a MAP

Program

```
<html>
 <head>
  <title>For Of Loop Iterate Object</title>
 </head>
 <body>
  <script type="text/javascript">
    const iterable = new Map([['a', 1], ['b', 2], ['c', 3]]);
    for(const entry of iterable)
    {
      document.write(entry+"<br>")
    }
    for (const [key, value] of iterable)
    {
      document.write(key+":"+value+"<br>");
    }
  </script>
 </body>
</html>
```

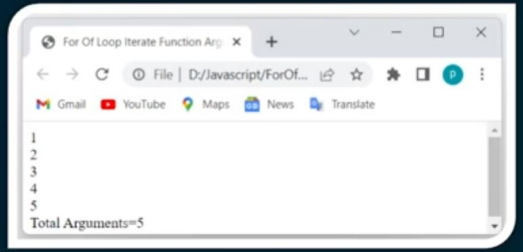


Output

Iterating over the arguments in a function

Program

```
<html>
 <head>
  <title>For Of Loop Iterate Function Arguments</title>
  <script type="text/javascript">
    function myfunction()
    {
      for (const arg of arguments)
      {
        document.write(arg+"<br>");
      }
      document.write("Total Arguments="+arguments.length);
    }
  </script>
 </head>
 <body>
  <script type="text/javascript">
    myfunction(1,2,3,4,5);
  </script>
 </body>
</html>
```



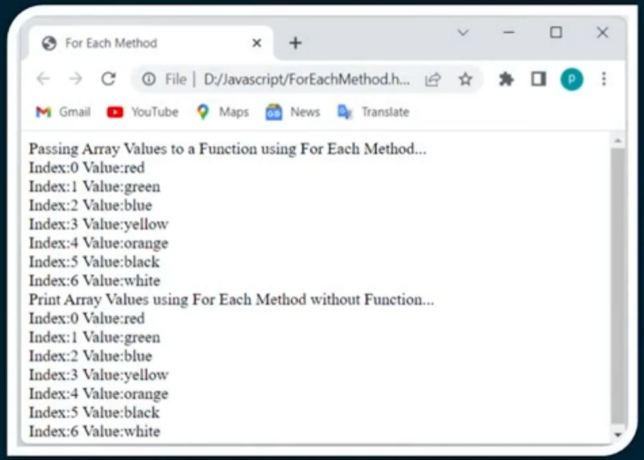
forEach with Arrays

The forEach() method is used to iterate over an array.

Program

```
<html>
  <head>
    <title>For Each Method</title>
  </head>
  <body>
    <script type="text/javascript">
      var colors=['red','green','blue','yellow','orange','black','white']
      document.write("Passing Array Values to a Function using For Each Method...<br>");
      colors.forEach(myFunction);
      function myFunction(value,index)
      {
           document.write('Index:'+index+' Value:'+value+'<br>');
      }
      document.write("Print Array Values using For Each Method without Function...<br>");
      colors.forEach((value,index)=>
      {
          document.write('Index:'+index+' Value:'+value+'<br>');
      });
    </script>
  </body>
</html>
```

Output:


For Each Method
File | D:/Javascript/ForEachMethod.h...
Gmail
YouTube
Maps
News
Translate
Passing Array Values to a Function using For Each Method...
Index:0 Value:red
Index:1 Value:green
Index:2 Value:blue
Index:3 Value:yellow
Index:4 Value:orange
Index:5 Value:black
Index:6 Value:white
Print Array Values using For Each Method without Function...
Index:0 Value:red
Index:1 Value:green
Index:2 Value:blue
Index:3 Value:yellow
Index:4 Value:orange
Index:5 Value:black
Index:6 Value:white

JavaScript Window Object Methods

The window object methods refer to the functions created inside the Window Object, which can be used to perform various actions on the browser window, such as how it displays a message or gets input from the user.

Below we have listed down some of the most commonly used window object methods:

Method	Description
alert()	specifies a method that displays an alert box with a message an OK button.
blur()	specifies a method that removes the focus from the current window.
clearInterval()	specifies a method that clears the timer, which is set by using setInterval() method.
close()	specifies a method that closes the current window.
confirm()	specifies a method that displays a dialogue box with a message and two buttons OK and Cancel
focus()	specifies a method that sets the focus on current window.
open()	specifies a method that opens a new browser window
moveTo()	specifies a method that moves a window to a specified position
moveBy()	specifies a Method that moves a window relative to its current position.
prompt()	specifies a method that prompts for input.
print()	specifies a method that sends a print command to print the content of the current window.
setTimeout()	specifies a method that evaluates an expression after a specified number of milliseconds.
setInterval()	specifies a method that evaluates an expression at specified time intervals (in milliseconds)
resizeBy()	specifies the amount by which the window will be resized
resizeTo()	used for dynamically resizing the window
scroll()	scrolls the window to a particular place in document
scrollBy()	scrolls the window by the given value
stop()	this method stops window loading

Window DialogBox

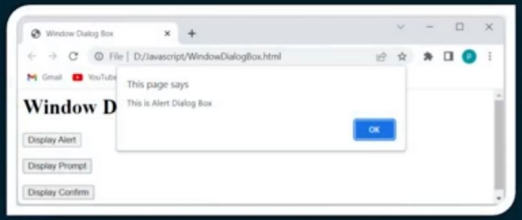
Method	Description
alert	Display an alert box
prompt	displays a dialog box that prompts the user for input. It returns the input value if the user clicks "OK", otherwise it returns null.
confirm	Display a confirmation box.

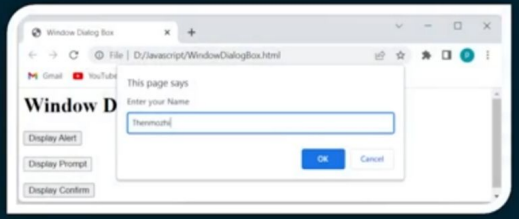
Program(WindowDialogBox.html)

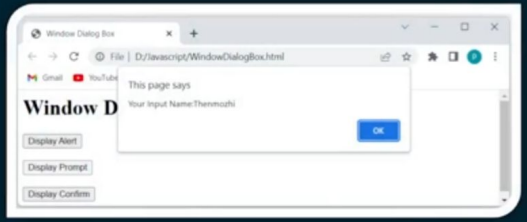
```
<html>
 <head>
  <title>Window Dialog Box</title>
 </head>
 <body>
  <h1>Window Dialog Box</h1>
  <form>
    <input type="button" id="button1" value="Display Alert" onclick="myFunction(1)"><br><br>
    <input type="button" id="button2" value="Display Prompt" onclick="myFunction(2)"><br><br>
    <input type="button" id="button3" value="Display Confirm" onclick="myFunction(3)">
  </form>
  <p id="demo"></p>
```

```
<script>
    function myFunction(buttonid)
    {
      switch(buttonid)
      {
        case 1:
            alert('This is Alert Dialog Box');
            break;
        case 2:
            var myName=prompt('Enter your Name',null);
            alert('Your Input Name:'+myName);
            break;
        case 3:
            var con=confirm('Would you like to confirm?');
            if(con==true)
              document.getElementById('demo').innerHTML='You Press OK Button';
            else
              document.getElementById('demo').innerHTML='You Press Cancel Button';
            break;
      }
    }
  </script>
 </body>
</html>
```

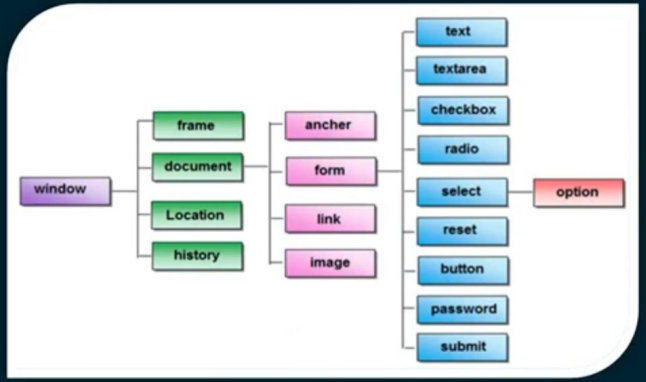
Output


Window Dialog Box
File | D:/Javascript/WindowDialogBox.html
Gmail
YouTube
Window D
This page says
This is Alert Dialog Box
OK
Display Alert
Display Prompt
Display Confirm


Window Dialog Box
File | D:/Javascript/WindowDialogBox.html
Gmail
YouTube
Window D
This page says
Enter your Name
Thenmozhi
OK
Cancel
Display Alert
Display Prompt
Display Confirm


Window Dialog Box
File | D:/Javascript/WindowDialogBox.html
Gmail
YouTube
Window D
This page says
Your Input Name:Thenmozhi
OK
Display Alert
Display Prompt
Display Confirm

Window Object Hierarchy



FRAME OBJECT

The Window frames property in HTML DOM is used to return the frame element in the form of an array object. This property represents all <iframe> elements in the current window. DOM Windowframe is a read-only property.

Syntax:

window.frames

Return Value: It returns a reference to the Window object, representing all the frames in the current window.

Properties:

- length property: It returns the number of iframe elements in the current window.

Syntax:

window.length

- location property: It changes the location of iframe.

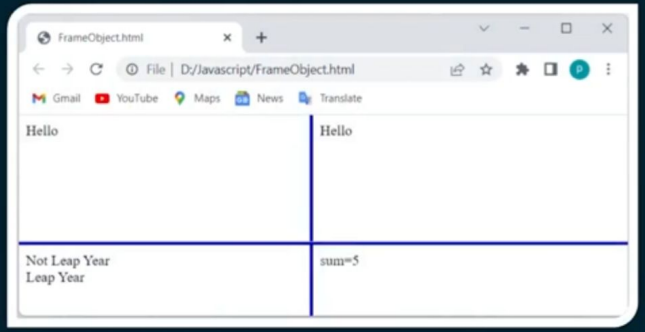
Syntax:

window.frames[index_no].location

Program(FrameObject.html)

```
<html>
 <head>
   <script>
      function displayFrames()
      {
        var frames=window.frames;
        alert('Frame Length:'+frames.length);
        for(i=0;i<frames.length;i++)
         alert(document.getElementsByTagName('frame')[i].name);
      }
   </script>
 </head>
   <frameset rows="*,*" cols="*,*" border=5 bordercolor="blue" onload="displayFrames()">
    <frame src="file:///D:/Javascript/ArrowFunction1.html" name="topleft">
    <frame src="file:///D:/Javascript/ArrowFunction2.html" name="topright">
    <frame src="file:///D:/Javascript/ArrowFunction3.html" name="bottomleft">
    <frame src="file:///D:/Javascript/ArrowFunction4.html" name="bottomright">
   </frameset>
   </noframe>
</html>
```

Output


FrameObject.html
File | D:/Javascript/FrameObject.html
Gmail
YouTube
Maps
News
Translate
Hello
Hello
Not Leap Year
Leap Year
sum=5

JavaScript Arrays

An array is an object that can store multiple values at once. For example,

```
const words = ['hello', 'world', 'welcome'];
```

Create an Array

You can create an array using two ways:

1. Using an array literal

The easiest way to create an array is by using an array literal []. For example,

```
const array1 = ["eat", "sleep"];
```

2. Using the new keyword

You can also create an array using JavaScript's new keyword.

```
const array2 = new Array("eat", "sleep");
```

Here are more examples of arrays:

```
// empty array
const myList = [ ];
// array of numbers
const numberArray =new Array [ 2, 4, 6, 8];
// array of strings
const stringArray = [ 'eat', 'work', 'sleep'];
// array with mixed data types
const newData = ['work', 'exercise', 1, true];
```

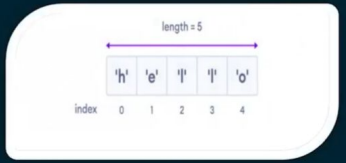
You can also store arrays, functions and other objects inside an array. For example,

```
const newData = [
   [1, 2 ,3],
   function hello() {console.log('hello')},
   {'task1': 'exercise'}
];
```

Access Elements of an Array

You can access elements of an array using indices (0, 1, 2 ...). For example,

```
const myArray = ['h', 'e', 'l', 'l', 'o'];
// first element
console.log(myArray[0]);  // "h"
// second element
console.log(myArray[1]); // "e"
```



Array length

You can find the length of an element (the number of elements in an array) using the length property. For example,

```
const dailyActivities = [ 'eat', 'sleep'];
// this gives the total number of elements in an array
console.log(dailyActivities.length); // 2
```

Array Methods

In JavaScript, there are various array methods available that makes it easier to perform useful calculations. Some of the commonly used JavaScript array methods are:

Method	Description
concat()	joins two or more arrays and returns a result
indexOf()	searches an element of an array and returns its position
find()	returns the first value of an array element that passes a test
findIndex()	returns the first index of an array element that passes a test
forEach()	calls a function for each element
includes()	checks if an array contains a specified element
push()	adds a new element to the end of an array and returns the new length of an array
unshift()	adds a new element to the beginning of an array and returns the new length of an array
pop()	removes the last element of an array and returns the removed element
shift()	removes the first element of an array and returns the removed element
sort()	sorts the elements alphabetically in strings and in ascending order
slice()	selects the part of an array and returns the new array
splice()	removes or replaces existing elements and/or adds new elements

```
<html>
  <head>
    <title>Example for Array methods</title>
  </head>
  <body>
     <script type="text/javascript">
       var bikes=['Bajaj','TVS','Honda Splender'];
       var cars=['Swift','Audi','Ford','BMW'];
       document.write('<br>Length of bikes='+bikes.length);
       document.write('<br><br>Length of cars='+cars.length);
       document.write('');
       //create empty array
       var vehicles=[];
       //concat two array
       vehicles=bikes.concat(cars);
       document.write('<br><br>Vehicles='+vehicles);
       document.write('<br><br>Vehicles='+bikes);
       document.write('<br><br>Vehicles='+cars);
       vehicles=bikes.join('-');
       document.write('<br><br>Join Vehicles='+vehicles);
       document.write('<br><br>Index of TVS='+bikes.indexOf('TVS'));
       bikes.push('Activa');
       bikes.unshift('KTM');
       document.write('<br><br> Bikes='+bikes);
       document.write('<br><br> Pop Last Element from Bikes='+bikes.pop());
       document.write('<br><br> Remove First Element from Bikes='+bikes.shift());
       document.write('<br><br> Sort Bikes='+bikes.sort());
       document.write('<br><br> Slice(1,3) from cars='+cars.slice(1,3));
       document.write('<br><br> Print Bikes using For each method');
       bikes.forEach((b)=>{
           document.write('<br>'+b);
       });
     </script>
  </body>
</html>
```

Working with Objects

JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method.

Two types of objects:

1. Predefined objects -predefined in browser and
2. user defined objects.

Objects Overview

In JavaScript, an object is a standalone entity, with properties and/or funtionalities. Compare with a Person, for example. Person is an Object, with properties. A Person has a Name,DOB,Address,Age etc. The same way, JavaScript objects can have properties, which define their characteristics.

Objects and properties

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables. You access the properties of an object with a simple dot-notation(.).

Syntax

```
objectName.propertyName
```

property name are case sensitive. You can define a property by assigning it a value.

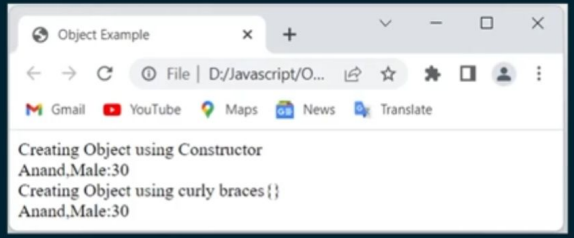
How to Create a Object

1) Using Object Constructor

2) Using Curly braces {}

Example(ObjectExample.html)

```
<html>
 <head>
  <title>Object Example</title>
 </head>
 <body>
  <script type="text/javascript">
     document.write('Creating Object using Constructor <br>');
     person=new Object();
     person.Name='Anand';
     person.gender='Male';
     person.age=30;
     document.write(person.Name+','+person.gender+':'+person.age);
     document.write('<br>Creating Object using curly braces{} <br>');
     person=
     {
       Name:'Anand',
       gender:'Male',
       age:30
     };
     document.write(person.Name+','+person.gender+':'+person.age);
  </script>
 </body>
</html>
```



Properties

Properties of JavaScript objects can also be accessed or set using a bracket notation. Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it. An object property name can be any valid JavaScript string This notation is also very useful when property names are to be dynamically determined.

Property accessors

Property accessors provide access to an object's properties by using the dot notation or the bracket notation.

There are two ways to access properties: dot notation and bracket notation.

Syntax

object.property

object['property']

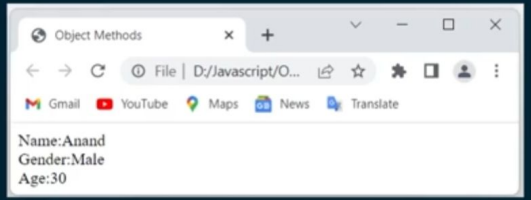
One can think of an object as an associative array (like map, dictionary, hash, lookup table). The keys in this array are the names of the object's properties.

Methods

An Object also having Methods -Program(Object Methods)

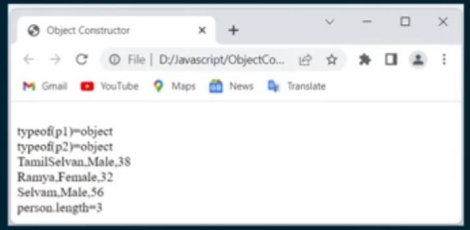
```
<html>
 <head>
  <title>Object Methods</title>
 </head>
 <body>
  <script type="text/javascript">

     person=
     {
        Name:'Anand',
        gender:'Male',
        age:30,
        getName()
        {
         return this.Name;
        },
        getGender()
        {
         return this.gender;
        },
        getAge()
        {
         return this.age;
        }
     }
     document.write('Name:'+person.getName()+'<br>');
     document.write('Gender:'+person.getGender()+'<br>');
     document.write('Age:'+person.getAge()+'<br>');
  </script>
 </body>
```

Output



Object Constructor --Program

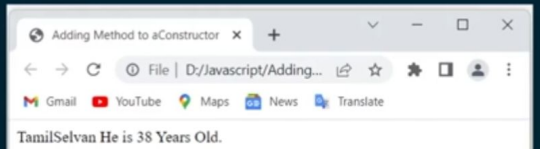
```
<html>
 <head>
  <title>Object Constructor</title>
 </head>
 <body>
  <script type="text/javascript">
     function person(Name,gender,age)//constructor
     {
        this.Name=Name;
        this.gender=gender;
        this.age=age;
     }
     p1=new person('TamilSelvan','Male',38); //constructor calling
     document.write('<br>typeof(p1)='+typeof(p1));
     p2=new person('Ramya','Female',32); //constructor calling
     document.write('<br>typeof(p2)='+typeof(p2));
     document.write('<br>'+p1.Name+','+p1.gender+','+p1.age);
     document.write('<br>'+p2.Name+','+p2.gender+','+p2.age);
     //Assigning new Property
     person['Name']='Selvam';
     person['gender']='Male';
     person['age']=56;
     document.write('<br>'+person.Name+','+person.gender+','+person.age);
     document.write('<br>person.length='+person.length); //length determine the number of
     properties(Name,gender,age)
  </script>
```



Function in Object Constructor

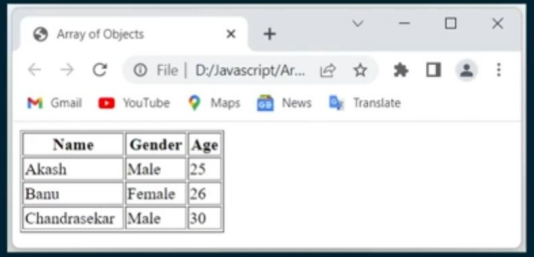
Program(AddingMethodToConstructor.html)

```
<html>
 <head>
  <title>Adding Method to a Constructor</title>
 </head>
 <body>
  <script type="text/javascript">
    function person(Name,gender,age)   //Constructor
    {
      this.Name=Name;
      this.gender=gender;
      this.age=age;
      this.details=function()  //function
      {
        return this.Name+' He is '+this.age+' Years Old.'
      };
    }
    p1=new person('TamilSelvan','Male',38); //constructor calling
    document.write(p1.details()); //function calling
  </script>
 </body>
</html>
```



Array of Objects -Program

```
<html>
 <head>
  <title>Array of Objects</title>
 </head>
 <body>
  <script type="text/javascript">
   const persons=[
     {
      'Name':'Akash',
      'gender':'Male',
      'age':25
     },
     {
      'Name':'Banu',
      'gender':'Female',
      'age':26
     },
     {
      'Name':'Chandrasekar',
      'gender':'Male',
      'age':30
     }];

   document.write('<table border=1 width=200>');
   document.write('<tr><th>Name</th><th>Gender</th><th>Age</th></tr>');
   for(i=0;i<persons.length;i++)
   {
     document.write('<tr><td>'+persons[i].Name+'</td><td>'+persons[i].gender+'</td><td>'+persons[i].age+'</tr>');
   }
   document.write('</table>');
  </script>
 </body>
</html>
```



this keyword

- In JavaScript, the **this** keyword refers to an object.
- Which object depends on how this is being invoked (used or called).
- The this keyword refers to different objects depending on how it is used:
- In an object method, this refers to the object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In a function, in strict mode, this is undefined.
- In an event, this refers to the element that received the event.
- Methods like call(), apply(), and bind() can refer this to any object.
- ES2020 introduced the globalThis object that provides a standard way to access the global object across environments

Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

In the previous chapter we learned how to use an object constructor:

Example

```
function person(Name,gender,age)   //Constructor
{
   this.Name=Name;
   this.gender=gender;
   this.age=age;
}
p1=new person('TamilSelvan','Male',38); //constructor calling
```

We also learned that you can not add a new property to an existing object constructor:

Example

```
person.city='Chennai';
```

To add a new property to a constructor, you must add it to the constructor function:

Example

```
function person(Name,gender,age)   //Constructor
{
   this.Name=Name;
   this.gender=gender;
   this.age=age;
   this.city='Chennai';
}
```

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

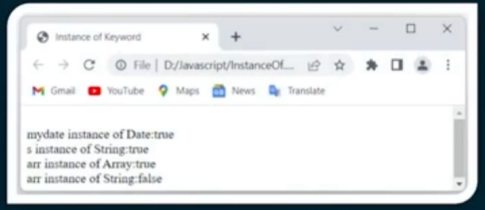
- Date objects inherit from Date.prototype
- Array objects inherit from Array.prototype
- person objects inherit from person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and person objects inherit from Object.prototype.

Checking the Prototype using instanceof keyword

```
<html>
 <head>
   <title>Instance of Keyword</title>
 </head>
 <body>
   <script type="text/javascript">
     mydate=new Date();
     s=new String("Hello");
     arr=[1,2,3];
     document.write('<br>mydate instance of Date:'+(mydate instanceof Date));
     document.write('<br>s instance of String:'+(s instanceof String));
     document.write('<br>arr instance of Array:'+(arr instanceof Array));
     document.write('<br>arr instance of String:'+(arr instanceof String));
   </script>
 </body>
</html>
```

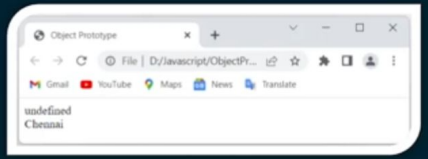


Using the prototype Property

The JavaScript prototype property allows you to add new properties to object constructors:

Program

```
<html>
 <head>
  <title>Object Prototype</title>
 </head>
 <body>
  <script type="text/javascript">
     function person(Name,gender,age)   //Constructor
     {
        this.Name=Name;
        this.gender=gender;
        this.age=age;
     }
     person.city='Chennai';
     p1=new person('TamilSelvan','Male',38); //constructor calling
     document.write(p1.city); //undefined

     //add prototype here
     person.prototype.city='Chennai';
     document.write('<br>'+p1.city);
  </script>
 </body>
</html>
```



Built-in Objects

1) Math Object
2) String Object
3) Date Object
4) Array Object

1) Math Object

Methods

Property	Description
abs(x)	Absolute value of x
acos(x)	Arccosine of x, in radian
asin(x)	Arcsine of x, in radian
atan(x)	Arctangent of x, a numeric value between -PI/2 and PI/2 radian
atan2(y, x)	Arctangent of the quotient of its arguments
ceil(x)	Value of x rounded up to the nearest integer
cos(x)	Cosine of x (x in radians)
exp()	Value of E^x
floor()	Value of x rounded below to the nearest integer
log()	Natural logarithm (base E) of x
max(a, b, ...)	Highest value
min(a, b, ...)	Lowest value
pow(x, y)	Value of x to power of y
random()	Random number between 0 and 1
round(x)	Value of x rounded to the nearest integer
sin(x)	Sine of x (x in radians)
sqrt(x)	Square root of x
tan(x)	Tangent of angle

2. String Object

Definition

A String is a collection of characters (or) sequence of characters, which is always enclosed with a single (') or double quotes (").

Creating a String

I) By using String literal

```
var stringname="String value";
```

II) By using String Object

```
var stringname = new string("String value");
```

String Object Properties and Methods

String property

- Length property – returns the length of the string.

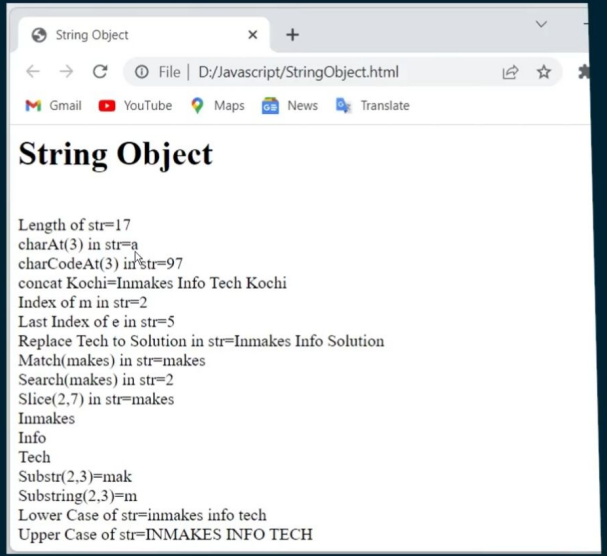
String Methods

String Methods	Description
1.charAt()	It returns a character at the specified position.
2.charCodeAt()	It returns the Unicode of a specific character.
3.concat()	Concatenation, joining the one or more string.
4.indexOf()	The given string or word, to find the position of the specified value in a sting
5.lastindexOf()	To find the last position of the specified value in a string.
6.LocaleCompare()	Comparing the two words, which is equal means returns 1, it is greater means returns 0, lesser means returns -1.
7.Match()	The given string matches the sentence means, returns the position.
8.Replace()	The specified string will be converted to a new string.
9.Search()	search the specific value or string, in a given paragraph, and return the position of the match.
10.Slice()	The given particular index, in the index indexed with particular index and returns the position of the match.
11.Split()	It will split a string into an array of substring.
12.Substr()	Returns the characters in a string starting at the specified location by a specified number of characters.
13.subString()	It returns characters from a string, between two specified indexes.
14.tostring()	Returns the value of the specified string object.
15.Trim()	Eliminates the space between the two ends of a string.
16.toUpperCase()	It converts a string to uppercase (capitalize) letters.
17.toLowerCase()	It converts a string to Lowercase letters.
18.valueOf()	it will display the total string value or primitive value of a string object.

Program

```
<html>
 <head>
  <title>String Object</title>
 </head>
 <body>
  <h1>string Object</h1>
  <script type="text/javascript">
    var str='Inmakes Info Tech';
    document.write('<br> Length of str='+str.length);
    document.write('<br> charAt(3) in str='+str.charAt(3));
    document.write('<br> charCodeAt(3) in str='+str.charCodeAt(3));
    document.write('<br> concat Kochi='+str.concat(' Kochi'));
    document.write('<br> Index of m in str='+str.indexOf('m'));
    document.write('<br> Last Index of e in str='+str.indexOf('e'));
    document.write('<br> Replace Tech to Solution in str='+str.replace('Tech','Solution'));
    document.write('<br> Match(makes) in str='+str.match('makes'));
    document.write('<br> Search(makes) in str='+str.search('makes'));
    document.write('<br> Slice(2,7) in str='+str.slice(2,7));
    arr=str.split(' ');
    for(i=0;i<arr.length;i++)
    document.write('<br>'+arr[i]);
    document.write('<br> Substr(2,3)='+str.substr(2,3));
    document.write('<br> Substring(2,3)='+str.substring(2,3));
    document.write('<br> Lower Case of str='+str.toLowerCase());
    document.write('<br> Upper Case of str='+str.toUpperCase());
  </script>
 </body>
</html>
```

Output


String Object
File | D:/Javascript/StringObject.html
Gmail
YouTube
Maps
News
Translate
String Object
Length of str=17
charAt(3) in str=a
charCodeAt(3) in str=97
concat Kochi=Inmakes Info Tech Kochi
Index of m in str=2
Last Index of e in str=5
Replace Tech to Solution in str=Inmakes Info Solution
Match(makes) in str=makes
Search(makes) in str=2
Slice(2,7) in str=makes
Inmakes
Info
Tech
Substr(2,3)=mak
Substring(2,3)=m
Lower Case of str=inmakes info tech
Upper Case of str=INMAKES INFO TECH

3. Date Object

The Date object is an inbuilt datatype of JavaScript language. It is used to work with dates and times. The Date object is created by using new keyword, i.e. new Date().

There are four different way to declare a date, the basic things is that the date objects are created by the new Date() operator.

Syntax:

new Date()

new Date(milliseconds)

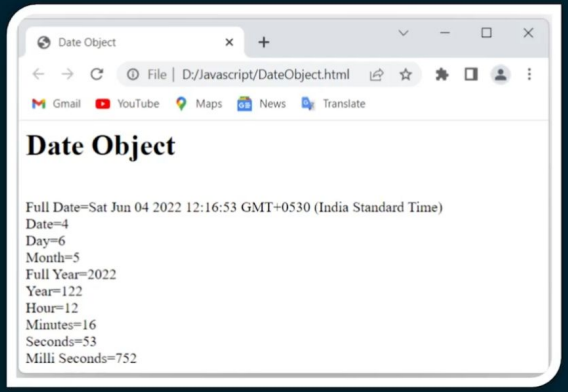
new Date(dataString)

new Date(year, month, date, hour, minute, second, millisecond)

Program

```
<html>
 <head>
  <title>Date Object</title>
 </head>
 <body>
  <h1>Date Object</h1>
  <script type="text/javascript">
    var myDate=new Date();
    document.write('<br> Full Date='+myDate);
    document.write('<br> Date='+myDate.getDate());
    document.write('<br> Day='+myDate.getDay());
    document.write('<br> Month='+myDate.getMonth());
    document.write('<br> Full Year='+myDate.getFullYear());
    document.write('<br> Year='+myDate.getYear());
    document.write('<br> Hour='+myDate.getHours());
    document.write('<br> Minutes='+myDate.getMinutes());
    document.write('<br> Seconds='+myDate.getSeconds());
    document.write('<br> Milli Seconds='+myDate.getMilliseconds());
  </script>
 </body>
</html>
```

Output


Date Object
File | D:/Javascript/DateObject.html
Gmail
YouTube
Maps
News
Translate
Date Object
Full Date=Sat Jun 04 2022 12:16:53 GMT+0530 (India Standard Time)
Date=4
Day=6
Month=5
Full Year=2022
Year=122
Hour=12
Minutes=16
Seconds=53
Milli Seconds=752

Functions

Function is used to perform operation and return a value. It accept parameters.

Types of Functions

1) Built-in Functions

2) User-defined Functions

1) Built-in Functions

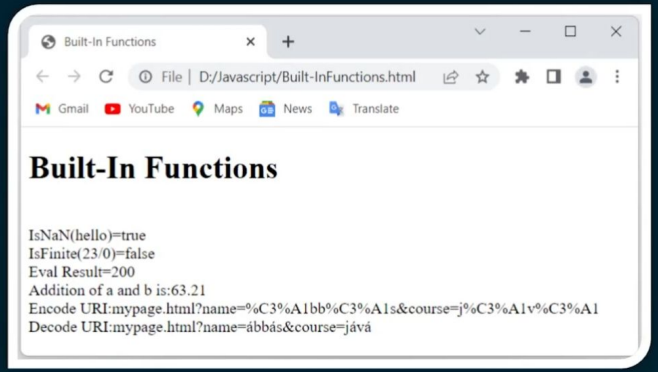
These Built-in Functions are available in Javascript Objects such as Array,String,Math,Date etc. These built in functions also known as global functions.

Function	Description
isNaN	This function is intended to determines whether value is a legal number or not.
isFinite	This function is intended to find whether a number is a finite legal number.
eval	This function is intended to execute Javascript source code.
number	This function is intended to converts object to the corresponding number value.
string	This function is intended to converts object to the corresponding string value.
parseInt	This function is intended to converts string value to integer.
parseFloat	This function is intended to converts string value to floating point number.
escape	This function is intended to encodes the string value into world wide acceptable format.
unescape	Deprecated. Use decodeURI() or decodeURIComponent() instead.
encodeURI	This function is intended to encode URI.
decodeURI	This function is intended to decode URI.
encodeURIComponent	This function is intended to encode URI component.
decodeURIComponent	This function is intended to decode URI component.

Example(Built-InFunctions.html)

```
<!DOCTYPE html>
<html>
  <head>
     <title>Built-In Functions</title>
  </head>
<body>
   <h1>Built-In Functions</h1>
   <script>
       let x = 10;
       let y = 20;
       let text = "x * y";
       let result = eval(text);
       document.write('<br>IsNaN(hello)='+isNaN('hello'));
       document.write('<br>IsFinite(23/0)='+isFinite(23/0));
       document.write('<br>Eval Result='+result);
       let a=parseInt('25');
       let b=parseFloat('38.21');
       document.write('<br>Addition of a and b is:'+(a+b));
       let uri="mypage.html?name=ábbás&course=jává"
       let encoded = encodeURI(uri);
       document.write('<br>Encode URI:'+encoded);
       let decoded = decodeURI(encoded);
       document.write('<br>Decode URI:'+decoded);
   </script>
</body>
</html>
```

Output


Built-In Functions
File | D:/Javascript/Built-InFunctions.html
Gmail
YouTube
Maps
News
Translate
Built-In Functions
IsNaN(hello)=true
IsFinite(23/0)=false
Eval Result=200
Addition of a and b is:63.21
Encode URI:mypage.html?name=%C3%A1bb%C3%A1s&course=j%C3%A1v%C3%A1
Decode URI:mypage.html?name=ábbás&course=jává

User-Defined Functions

- User-Defined Functions are created by user.

Create User-Defined Functions

- User-Defined Function prefix with function keyword.

Example

```
function showArea()
{
    area=length*breath;
}
```

- Function accept arguments

Example

```
function showArea(length,breath)
{
    this.length=length;
    this.breath=breath;
    area=length*breath;
}
```

- Function should return a value to the caller program

Example

```
function showArea(length,breath)
{
    this.length=length;
    this.breath=breath;
    area=length*breath;
    return area;
}
```

Calling Function

We can call a function directly in a script or by the HTMLEvent.

```
rect1=showArea(10,20);
```

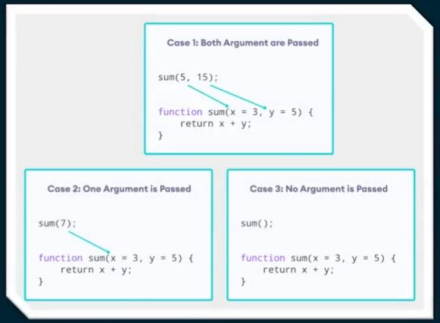
Program

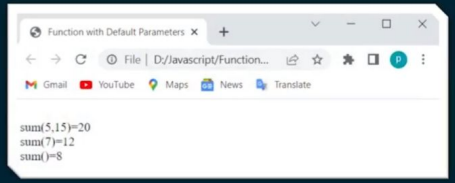
```
<html>
 <head>
  <title>User-Defined Function</title>
 </head>
 <body>
  <h1>User-Defined Function</h1>
  <script type="text/javascript">
   function showArea(length,breath)
   {
        this.length=length;
        this.breath=breath;
        area=length*breath;
        return area;
   }
   rect1=showArea(2,3); //calling function
   document.write('Area Of Rectangle1='+rect1);
   rect2=showArea(6.5,3.8); //calling function
   document.write('<br> Area Of Rectangle2='+rect2);
  </script>
 </body>
</html>
```

Output

Javascript Default Parameters

```
<html>
 <head>
  <title>Function with Default Parameters</title>
 </head>
 <body>
  <script type="text/javascript">
   function sum(x = 3, y = 5)
   {
    // return sum
    return x + y;
   }

   document.write('<br>sum(5,15)='+sum(5,15));
   document.write('<br>sum(7)='+sum(7));
   document.write('<br>sum()='+sum());
  </script>
 </body>
</html>
```

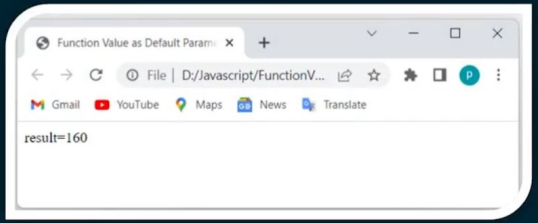




Passing Function Value as Default Value

```
<html>
  <head>
    <title>Function Value as Default Parameters</title>
  </head>
  <body>
    <script type="text/javascript">
     const sum=()=>15;
     const calculate=function(x,y=x*sum()){return x+y;}
     const result=calculate(10);
     document.write('result='+result);
    </script>
  </body>
</html>
```

Output



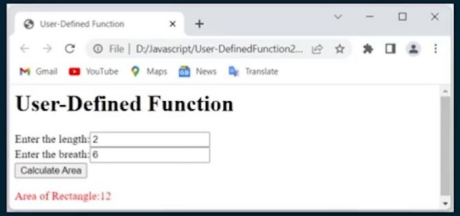
Arrow function

```
let myFunction = (a, b) => a * b;

hello = function() {
  return "Hello World!";
}
hello = () => {
  return "Hello World!";
}
hello = () => "Hello World!";
```

Program(Calling a Function using HTMLEvent)

```
<html>
 <head>
  <title>User-Defined Function</title>
 </head>
 <body>
  <h1>User-Defined Function</h1>
  <script type="text/javascript">
   function showArea(length,breath)
   {
        this.length=length;
        this.breath=breath;
        area=length*breath;
        document.getElementById("p1").innerHTML='Area of Rectangle:'+area;
   }
  </script>
  <form>
   Enter the length:<input type="number" name="length"><br>
   Enter the breath:<input type="number" name="breath"><br>
   <input type="button" value="Calculate Area" onClick="showArea(length.value,breath.value)">
   <p id="p1" style="color:red;"></p>
  </form>
 </body>
</html>
```

Output



JavaScript Arrow Function

Arrow function is one of the features introduced in the ES6 version of JavaScript, similar to lambda expression in java. Arrow functions provide a shorthand way of declaring anonymous functions. For example,

This function

```
// function expression
let x = function(x, y)
{
  return x * y;
}
```

can be written as using an Arrow Function(=>).

```
// using arrow functions
let x = (x, y) => x * y;
```

Arrow Function Syntax

The syntax of the arrow function is:

```
let myFunction = (arg1, arg2, ...argN) =>
{
   statement(s)
}
```

- myFunction is the name of the function
- arg1, arg2, ...argN are the function arguments
- statement(s) is the function body

If the body has single statement or expression, you can write arrow function as:

```
let myFunction = (arg1, arg2, ...argN) => expression
```

Advantages of using Arrow Function

- This arrow function reduces lots of code and makes the mode more readable.
- Arrow function syntax automatically binds “this” to the surrounding code’s context.
- Writing the arrow => is more flexible as compared with the writing function keyword.

Disadvantages of using Arrow Function

- An arrow function doesn’t have its own bindings with this or super.
- An Arrow function should not be used as methods.
- An arrow function can not be used as constructors.
- An arrow function can not use yield within its body.
- Arrow function cannot be suitable for call apply and bind methods.

Difference Between Regular and Arrow Functions

	Regular Function	Arrow Function
Syntax	Explicitly mention with function keyword and function Name	By Writing a few lines of code a programmer can get the same result as ordinary Functions
Arguments Binding	arguments inside the regular functions contains the list of arguments.	The arrow function, on the opposite, doesn't define arguments i.e. they do not have arguments binding.
this keyword	This value is dynamic a standard Javascript Function. Because of the dynamic context, the value of the variable changes depending on how the function is called.	arrow function lacks its own "this" keyword, the behaviour of this inside an arrow function differs significantly from that of an ordinary function.
new keyword	Regular functions can be built and called. The new keyword can be used to call them	the arrow functions are only callable and not constructible, i.e., arrow functions can never be used as constructor functions

Forms and Inputs The <form> Tag

Forms are used to submit information from the user, by way of the browser, to a server somewhere.

<form> is the tag to start a form.

</form> is the tag to end a form.

Without the form tag, you can still make something that looks like a form, but it will not communicate with a server. The form has several commonly used attributes.

- method= can be get or post.
- action= specifies the URL of a program that will process the inputs if the form is submitted.

Within the form, there can be any number of input fields. In order to be transmitted, each has a name and a value. There are three major types of input fields: input, textarea, and select.

It can also be designated by autofocus to receive the cursor when the page is loaded. And it can (probably) participate in taborder, which tells which field you go to next when you press the tab key.

- name=value

When your form is submitted, each input field is either transmitted as part of the form.

- The name= Parameter

The name parameter is used in every field. The name parameter is not displayed to the user, but they can see it with a "show page source" command.

The <input> tag

input covers most of the cases. It is a void tag. (No end tag is allowed.) For each <input> tag in valid HTML5, is a separate closing tag is forbidden.
Within the input tag category, there are many types. They are identified by including a type attribute within the input tag. is the tag for an input field.
<input type=xxx ...> is the tag for an input field.

type attribute

- text
- password
- hidden
- checkbox
- radio
- button
- submit
- reset

The default type is text.

The value= Parameter

- The value parameter is used in every field.
- The value parameter may or may not be openly displayed to the user, but they can see it with a "show page source" command.
- The value parameter may or may not be easily changed by your user.
- Hackers always have the ability to change any parameter.

type=password

<input type=password...> is the tag for a password field.

- It acts almost exactly like a text field, but it obscures the thing typed in, replacing each character with a dot or other symbol. It provides one line of space for the user to type in something.
- The most important parameters are: name, size.
- The name parameter is normal.
- The size parameter the number of characters that should be visible.
- Other useful parameters are: value.
- The value parameter will be displayed to the user as a series of dots and can be easily changed by them. If your user is a hacker, they can see the value in that field.

type=checkbox

<input type=checkbox ...> is the tag for a checkbox.

- This is generally a square box in which there is either a checkmark or not.
- Normally each checkbox has a different name.
- The name and value are transmitted to your CGI program if and only if
- the box is checked.
- You can pre-check one or more the spots by using the checked parameter.
- checked=checked or simply checked with no equals sign.

type=radio

<input type=radio ...> is the tag for a radio button. This is generally a round space in which there is either a dot or not.

- Radio buttons are for selecting among mutually exclusive alternatives, like yes/no, male/female, red/green/yellow.
- Several related radio buttons are grouped together by having the same name.
- You can pre-select one of the spots by using the checked parameter. checked=checked or simply checked with no equals sign.

type=file

<input type=file ...> is the tag for uploading a file.
The entire file will be uploaded when the form is submitted.

type=hidden

- is the tag for a hidden field.
- Hidden fields and cookies are the main ways that the server can keep track of many conversations (sessions) going on with many browsers all at the same time.
- The only useful parameters are: name, value.
- The name=value will be transmitted when your form is submitted.
- The name parameter is normal.
- The value parameter is not displayed to the user. It is hidden. That's the whole point. But it is transmitted when the form is submitted.

HTML5 Input Types

There are several new HTML5 type= attributes. These may become more useful in the future. They may not work in any special way, but they are always safe to use because when a browser does not understand one of these, it will use type=text instead.

- For dates and times: type=datetime, type=datetime-local, type=date, type=month, type=time, and type=week.
- For numbers: type=number and type=range.
- For colors: type=color.
- For special types of data: type=email, type=url, type=tel, and type=search.

The <textarea> Tag

textarea provides a rectangular space where several lines of information can be entered. It has a required end tag. The textarea tag has a name but the value is not specified inside the tag.
The value is whatever appears between the textarea tag and its ending tag,or whatever is entered into that space by the user. For each <textarea> tag in valid HTML5,is a separate closing tag requied.
CSS can be used to make the textarea the particular size you want. You can also use the rows= and cols= parameters to get a particular size. The user can often resize the textarea box by grabbing (mouse-down) the lower right corner and stretching the box or shrinking it.

The <Select> and <Option> Tags

select and option work together to create a drop-down list. The select tag has a name attribute but no value. The option tag has a value attribute but no name.
For each <select> tag in valid HTML5, is a separate closing tag is required.
For each <option> tag in valid HTML5, is a separate closing tag is optional.

Auto Focus and Tab Index

The autofocus= attribute causes the focus to begin on a certain element.
Normally it would be used with an <input ...> element. This causes the cursor to rest inside that field when the page is first loaded.

The value of autofocus does not matter, so it can be left off. In that case, the equals sign can also be left off. When the user pressed the tab key, the tabindex= attribute tells the browser where to send the cursor next. If tabindex is not specified, the browser normally sends the cursor to the next input field it can find in the HTML.
We can control tab order using the tabindex attribute.
<input ... tabindex=1 ...>

Advanced Concepts

Input Patterns

It is possible to specify a pattern for what the user is allowed to type into an input field.
For example, if you wanted to force the user to type a five-digit number, like for a US zip code, you could say:

```
<input ... pattern=[0-9]{5} ...>
```

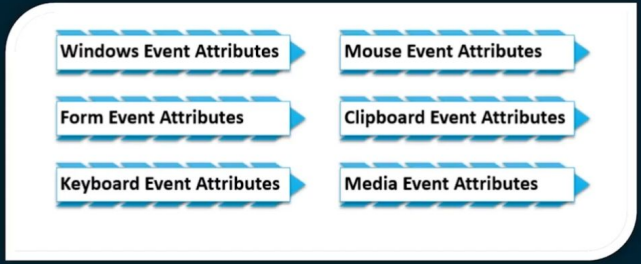
The [0-9] part means a character in the range of zero through nine. The {5} part means five of them.
For a ten-digit telephone number using the xxx-xxx-xxxx format you could say:

```
<input ... pattern=[0-9]{3}-[0-9]{3}-[0-9]{4} ...>
```

Patterns use something called a "regular expression." The rules for patterns
are not difficult to learn, and are very powerful. Using a pattern does not guarantee that the browser will enforce it, but most modern browsers do.

HTML Events

In a web page or a website created using the HTML script (HyperText Markup Language), every action performed by the user and the web tool itself is termed an HTML Event. The description contains six types of attributes for every Event: window event attribute, form event attribute, keyboard event attribute, mouse event attribute, media event attribute, and clipboard event attribute. Each of these event attributes will have specific dedicated attributes that are used to fetch the details of the HTML Event.



- **Windows Event Attributes**: This is provided for the action of the windows object. It works in body tag <body>.
- **Form Event Attributes**: If the user performs some action in the form like input data, cancel, submit, then these event attributes works.
- **Keyboard Event Attributes**: This keyboard event attributes used for keyboard action and user interaction.
- **Mouse Event Attributes**: Mouse event attribute used for mouse action that is move, click, wheel, etc.
- **Clipboard Event Attributes**: This event attributes used for clipboard action: example, cut, copy, paste.
- **Media Event Attributes**: This event attribute works on media files like video etc.

Description of Event Attributes

1. Windows Event Attributes

Attribute	Description
onafterprint	This script act, though, after the document printed.
onbeforeprint	This script act, though, before a document printed.
onbeforeunload	Whereas before the unloaded document, this Script works.
onerror	In the document occurs the Error then this event executed.
onhashchange	The anchor part of URL change in the document that time event executed.
onload	When the first Web page is loaded completely, then this event executed.
onmessage	In the document, the message that occurred at that time event executed.
onoffline	If the network connection is unavailable and the browser says offline, then the event executed.
ononline	When the network available in the browser, then the event executed.
onpagehide	This script act; if the user not working on a current webpage, a then-current page can be hidden.
onpageshow	This script act at that time the current webpage is load.
onpopstate	This script automatically works on the browser for a history state change.
onresize	This script act when the browser of the window changes the size.
onstorage	When users web storage updated, then the event executed.
onunload	The user's current web page is not loaded or the window is closed, then the event is executed.

2. Form Event Attributes

Attribute	Description
onblur	Some form validation object loos the focus, then event fired.
onchange	The value change in the form, then event fired.
onfocus	In the form <input>, <a> , <select> object has focus. Working on this object then event fired.
oninput	The user gives input of value in the form then this event fired.
oninvalid	The event works on when the element does not satisfy its predefined constraints.
onreset	User reset the form information, then event fired.
onsearch	Users search the required field, then event fired.
onselect	The user selects the text or text area in form, then event fired.
onsubmit	The user submits the form at the end then the event fired.

3. Keyboard Event Attributes

Attribute	Description
onkeydown	Using a keyboard, the user press the key down at that point event works
onkeypress	Using the keyboard, users press the key and display characters at that point event works.
onkeyup	After the press, the key user releases the key then the event works.

4. Mouse Event Attributes

Attribute	Description
onclick	The user clicks the mouse on the button then an event occurred.
ondblclick	Users double click the mouse then the event occurred.
onmousedown	The user presses the mouse button on the element then the event occurred.
onmousemove	The user moves the mouse pointer over the element then the event occurred.
onmouseout	The user moves the mouse outside of the element then the event occurred.
onmouseover	The user moves the mouse over the element then the event occurred.
onmouseup	The user released the mouse button then the event occurred.
onmousewheel	Using the mouse wheel user rolls the up and down on element then the event occurred.
onwheel	Using a mouse wheel user roll them up and down then the event occurred.

5. Clipboard Event Attributes

Attribute	Description
oncopy	Using mouse users to copy the content, then the event occurred.
oncut	Using a mouse, users cut the content then the event occurred.
onpaste	Using a mouse user, paste the content, then an event occurred.

6. Media Event Attributes

Attribute	Description
onabort	When media files aborted for download and play again, then an event occurs.
oncanplay	When any media file ready for play, then this trigger is fired.
oncanplaythrough	Media file ready to play without buffering and loading.
oncuechange	Element changes the cue of <track> then event fired.
ondurationchange	The Media file changes the length of time then the trigger is fired.
onemptied	If the Media file unavailable and come fatal error, then the trigger is fired.
onended	The Media file comes on endpoint then the trigger is fired.
onerror	When an error occurred to get the media file, the trigger is fired
onloadeddata	The Media file loads the data then the trigger is fired.
onloadedmetadata	The Media file loads the metadata then the trigger is fired.
onloadstart	The Media file starts to load then the trigger is fired.
onpause	The Media file paused to play again then the trigger is fired.
onplay	Media file ready to play, then trigger is fired.
onplaying	The Media file starts to play when the trigger is fired.
onprogress	This script act when the browser is working on connecting with the media data.
onratechange	If the videos playback speed is changed, then the trigger is fired.
onseeked	Users completed moving; otherwise, skip the new position of video. this attribute set as false.
onseeking	The user wants to move; otherwise, skip the new position of the video. this attribute set as true.
onstalled	When the browser suddenly stops to the connection of data, then the event works.
onsuspend	When the web Browser on purpose does not get media data, then events work.
ontimeupdate	When a user changes the video play position like forward and backward.
onvolumechange	To change media volume low to high.
onwaiting	If the data load the information, current video stop with buffering then event works.

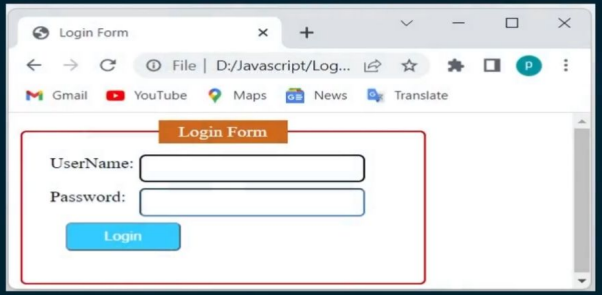
Program

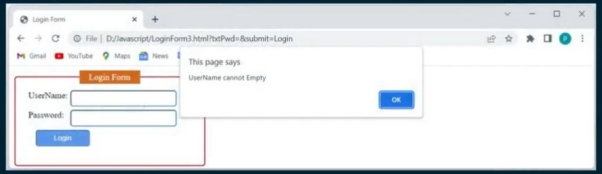
```
<html>
 <head>
   <title>Login Form</title>
   <style>
    label
    {
     float:left;
     width:4em;
    }
    input
    {
      margin-left:1em;
      margin-bottom:.5em;
      padding: 5px 15px;
      border-radius:5px;
      border-color:#4682B4;
    }
    legend
     {
        background-color: #D2691E;
        color: #fff;
        padding: 3px 6px;
        width:100px;
        text-align:center;
     }
    fieldset
   {
     padding: 12px 24px;
     width:200px;
     border-color:red;
     border-radius:5px;
    }
```

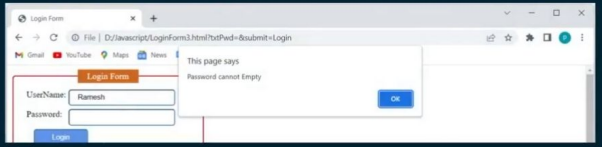
```
#submit1
    {
      color:white;
      border-radius:5px;
      width:100px;
      height:30px;
      background-color:#00BFFF;
      opacity:0.8;
                          transition: 0.5s;
    }
    #submit1:hover
    {
      background-color:#4169E1;
    }

   </style>
   <script type="text/javascript">
    function validate()
    {

      if(document.forms[0].elements[0].value.length<1)
       {

         alert('UserName cannot Empty');
         document.forms[0].elements[0].focus();

       }
      else if(document.forms[0].elements[1].value.length<1)
       {
         alert('Password cannot Empty');
         document.forms[0].elements[1].focus();
       }
```

```
else if(document.forms[0].elements[0].value=='Ramesh'&&document.forms[0].elements[1].value=='Inmakes@123')
          alert('Login Successful');
        else
          alert('Invalid UserId or Password');
}
   </script>
 </head>
 <body>
   <fieldset style="width:300px">
    <legend>Login Form</legend>
    <form>
      <label for="txtUser">UserName:</label>
      <input type="text" autofocus><br>
      <label for="txtPwd">Password:</label>
      <input type="password" name="txtPwd"><br>
      <input type="submit" name="submit" id="submit1" value="Login" onClick="validate()">
    </form>
   </fieldset>
 </body>
</html>
```

Output


Login Form
File | D:/Javascript/Log...
Gmail
YouTube
Maps
News
Translate
Login Form
UserName:
Password:
Login


Login Form
File | D:/Javascript/LoginForm3.html?txtPwd=&submit=Login
This page says
UserName cannot Empty
OK
Login Form
UserName:
Password:
Login


Login Form
File | D:/Javascript/LoginForm3.html?txtPwd=&submit=Login
This page says
Password cannot Empty
OK
Login Form
UserName:
Ramesh
Password:
Login

Javascript Basic Form Validation

Program(SignUp.html)

```
<html>
 <head>
  <title>Sign Up</title>
  <style>
     .box
     {  position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%,-50%);   /*for align browser center*/
        display:flex;
        justify-content:center;
     }
     fieldset
     {
        width:300px;
        height:350px;
        border-color:blue;
        border-radius:5px;
        background-color:#B041FF;
        padding: 12px 24px;
        font-size:20px;
     }
```

```
legend
    {
      background-color:        #ff00ff; /*#C45AEC;tyrian purple*/
      color:#fff;
      width:200px;
      font-size:20px;
      padding: 3px 6px;
      text-align:center;
      border-radius:5px;
    }
    label
    {
      color:white;
    }
    input
    {
      width:400px;
      height:30px;
      font-size:20px;
    }
    #submit1
    {
      background-color:cornflowerblue;
      color:white;
      border:none;
      border-radius:5px;
      outline: none;
```

```
 #submit1:hover
      {
        //background-color:#00ffff;
        background-color:orange;
        //transition:0.5s;
      }
      .error
      {
        color:#ff0000;
      }
    </style>
  </head>
<body>
   <div class="box">
    <fieldset>
     <legend>SignUp</legend>
      <form>
       <div>
         <label for="txtUser">Username:</label>
       </div>
       <div id="input">
         <input type="text" id="txtUser" autocomplete=off>
       </div>
       <div id="error1" class="error"></div>
              <div>
         <label for="txtEmail">Email:</label>
       </div>
```

```
<div id="input">
        <input type="text" id="txtEmail" autocomplete=off>
</div>
<div id="error2" class="error"></div>
<div>
    <label for="txtPass">Password:</label>
</div>
<div id="input">
    <input type="password" id="txtPass" autocomplete=off>
</div>
<div id="error3" class="error"></div>
<div>
    <label for="txtConfirmPass">Confirm Password:</label>
</div>
<div id="input">
    <input type="password" id="txtConfirmPass" autocomplete=off>
</div>
<div id="error4" class="error"></div>
    <input type="submit" id="submit1" value="Register" onclick="return validate()">
</form>
  </fieldset>
   </div>
    <script type="text/javascript">
     var user=document.getElementById("txtUser");
     var email=document.getElementById("txtEmail");
     email.addEventListener('focusout',validateEmail);
     var pwd=document.getElementById("txtPass");
```

```
function validate()
    {
      if(user.value=='')
      {
        document.getElementById("error1").innerHTML='Username cannot Empty';
        user.focus();
        return false;
      }
      else
      {
                document.getElementById("error1").innerHTML='';
      }
      if(email.value=='')
      {
        document.getElementById("error2").innerHTML='Email cannot Empty';
        email.focus();
        return false;
      }
      else
      {

                document.getElementById("error2").innerHTML='';
      }
      if(pwd.value!='')
      {
        if(pwd.value.length<7)
        {
          document.getElementById("error3").innerHTML='Password require minimum 8 characters';
          pwd.focus();
          return false;
        }
```

```
else
        {
                document.getElementById("error3").innerHTML='';
        }
      }
      else
      {
        document.getElementById("error3").innerHTML='Password cannot Empty';
        pwd.focus();
        return false;
      }
            if(confirmpwd.value!='')
      {
        if(confirmpwd.value.length<7)
        {
          document.getElementById("error4").innerHTML='Confirm Password require minimum 8 characters';
          confirmpwd.focus();
          return false;
        }
                        else
        {
                document.getElementById("error4").innerHTML='';
        }
      }
      else
      {
        document.getElementById("error4").innerHTML='Confirm Password cannot Empty';
              confirmpwd.focus();
        return false;
      }
```

```
if(pwd.value!=confirmpwd.value)
        {
          document.getElementById("error3").innerHTML='Password Mismatch';
          document.getElementById("error4").innerHTML='Confirm Password Mismatch';
          pwd.focus();
          return false;
        }
        else
        {
          document.getElementById("error3").innerHTML='';
          document.getElementById("error4").innerHTML='';
        }
        alert('Success');
       return true;
      }
```

```
function validateEmail()
      {
       emailId=email.value;
       atpos = emailId.indexOf("@");
       dotpos = emailId.lastIndexOf(".");
       /*alert('@ position='+atpos);
       alert('. position='+dotpos);
       alert('. - @='+(dotpos-atpos));*/
       if (atpos < 1 || ( dotpos - atpos < 2 ))
       {
         document.getElementById("error2").innerHTML='Please Enter Correct Emailid';
         email.focus() ;
         return false;
       }
       else
       {
        document.getElementById("error2").innerHTML='';
        return( true );
       }
     }
   </script>
  </body>
</html>
```

Output:


SignUp
Username:
Username cannot Empty
Email:
Password:
Confirm Password:
Register


SignUp
Username:
Ramesh
Email:
Email cannot Empty
Password:
Confirm Password:
Register

Output


SignUp
Username:
Ramesh
Email:
ramesh123@gmail.com
Password:
Password cannot Empty
Confirm Password:
Register


SignUp
Username:
Ramesh
Email:
ramesh123@gmail.com
Password:
Password require minimum 8 characters
Confirm Password:
Register

Output


SignUp
Username:
Ramesh
Email:
ramesh123@gmail.com
Password:
Confirm Password:
Confirm Password cannot Empty
Register


SignUp
Username:
Ramesh
Email:
ramesh123@gmail.com
Password:
Confirm Password:
Confirm Password require minimum 8 characters
Register

JAVASCRIPT WINDOW OBJECT

JavaScript Window is a global Interface (object type) which is used to control the browser window lifecycle and perform various operations on it.

A global variable window, which represents the current browser window in which the code is running, is available in our JavaScript code and can be directly accessed by using window literal.

It represents a window containing a webpage which in turn is represented by the document object. A window can be a new window, a new tab, a frame set or individual frame created with JavaScript.

In case of multi tab browser, a window object represents a single tab, but some of its properties like innerHeight, innerWidth and methods like resizeTo() will affect the whole browser window.

Whenever you open a new window or tab, a window object representing that window/tab is automatically created.

The properties of a window object are used to retrieve the information about the window that is currently open, whereas its methods are used to perform specific tasks like opening, maximizing, minimizing the window, etc.

Using JavaScript Window Object

JavaScript window object to create a new window, using the open() method. This method creates a new window and returns an object which further can be used to manage that window.

Following is the syntax for the window object open() method:

let new Window = window.open(url, windowName, [windowFeatures]);

We can provide, as many properties as we want, while creating a new window.

When the open() method is executed, it returns the reference of the window object for the new window created, which you can assign to a variable, like we have done in the code above. We have assigned the value returned by the window.open() method to the newWindow variable.

We have used the newWindow variable to access the new window, like getting the name of the window, getting location of the window which opened the new window etc.

A table of most popular window object properties is given below:

Property	Description
closed	returns a boolean value that specifies whether a window has been closed or not
document	specifies a document object in the window.
history	specifies a history object for the window.
frames	specifies an array of all the frames in the current window
defaultStatus	specifies the default message that has to be appeared in the status bar of Window.
innerHeight	specifies the inner height of the window's content area.
innerWidth	specifies the inner width of the window's content area.
length	specifies the number of frames contained in the window.
location	specifies the location object for window
name	specifies the name for the window
top	specifies the reference of the topmost browser window.
self	returns the reference of current active frame or window.
parent	returns the parent frame or window of current window.
status	specifies the message that is displayed in the status bar of the window when an activity is performed on the Window.
screenleft	specifies the x coordinate of window relative to a user's monitor screen
screenTop	Specifies the y coordinate of window relative to a user's monitor screen
screenX	specifies the x coordinate for window relative to a user's monitor screen
screenY	Specifies the y coordinate for window relative to a user's monitor screen

Program(WindowObject1.html)

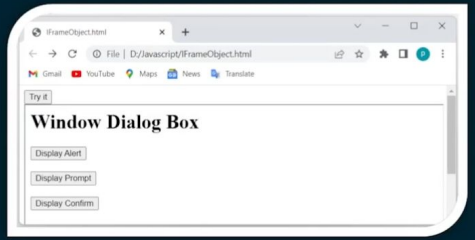
```
<html>
  <head>
    <title>Window Object</title>
  </head>
  <body>
    <h1>Window Methods</h1>
    <form>
       <input type="button" name="button1" value="Create New Window" onclick="openNewWindow()"><br><br>
       <input type="button" name="button2" value="Close New Window" onclick="closeNewWindow()"><br><br>
       <input type="button" name="button3" value="Close the Existing Window" onclick="window.close()"></div>
    </form>
    <script>
      var win;
      function openNewWindow()
      {
        win=window.open('',"MyNewWindow","width=500px,height=200px,screenX=100,screenY=100");
        /*get the Dimension of the window*/
        let h = win.outerHeight;
        let w = win.outerWidth;
        win.document.write("<br>Window Name: "+win.name);
        win.document.write("<br>Height and width of the window are: "+w+" and "+h);      }
      function closeNewWindow()
      {
         win.close();
      }
    </script>
  </body>
</html>
```

Location Property using Iframe

Program

```
<html>
  <head>
    <script>
     function myFunction()
     {
        window.frames[0].location = "file:///D:/Javascript/WindowDialogBox.html";
     }
</script>
  </head>
  <body>
     <button onclick="myFunction()">Try it</button>
     <iframe src="" name="myFrame" style="width:100%;height:300px" ></iframe>
  </body>
</html>
```

Output



DOCUMENT OBJECT

The current web page loaded into each window is modelled using a document object. The title property of the document object tells you what is between the opening<title> and closing </title> tag for that web page, and the lastModified property of the document object tells you the date this page was last updated.

Document Object Properties and Methods (contd...)

Property / Method	Description
activeElement	Returns the currently focused element in the document
addEventListener()	Attaches an event handler to the document
baseURI	Returns the absolute base URI of a document
body	Sets or returns the document's body (the <body> element)
characterSet	Returns the character encoding for the document
close()	Closes the output stream previously opened with document.open()
cookie	Returns all name/value pairs of cookies in the document
createAttribute()	Creates an attribute node
createComment()	Creates a Comment node with the specified text
createDocumentFragment()	Creates an empty DocumentFragment node
createElement()	Creates an Element node
createEvent()	Creates a new event
createTextNode()	Creates a Text node
defaultView	Returns the window object associated with a document, or null if none is available.
designMode	Controls whether the entire document should be editable or not.
doctype	Returns the Document Type Declaration associated with the document
documentElement	Returns the Document Element of the document (the <html> element)
documentURI	Sets or returns the location of the document
domain	Returns the domain name of the server that loaded the document

Document Object Properties and Methods (contd...)

embeds	Returns a collection of all <embed> elements the document
forms	Returns a collection of all <form> elements in the document
getElementById()	Returns the element that has the ID attribute with the specified value
getElementsByClassName()	Returns an HTMLCollection containing all elements with the specified class name
getElementsByName()	Returns an live NodeList containing all elements with the specified name
getElementsByTagName()	Returns an HTMLCollection containing all elements with the specified tag name
hasFocus()	Returns a Boolean value indicating whether the document has focus
head	Returns the <head> element of the document
images	Returns a collection of all <img> elements in the document
implementation	Returns the DOMImplementation object that handles this document
importNode()	Imports a node from another document
lastModified	Returns the date and time the document was last modified
links	Returns a collection of all <a> and <area> elements in the document that have a href attribute
normalize()	Removes empty Text nodes, and joins adjacent nodes

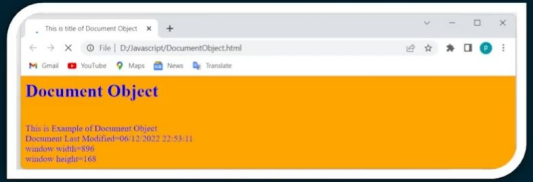
The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.
title	Sets or returns the title of the document
forms	Returns a collection of all <form> elements in the document
lastModified	Returns the date and time the document was last modified

Program

```
<html>
 <head>
  <title>Document Object</title>
 </head>
 <body>
    <h1>Document Object</h1>
    <script language="javascript">
     document.title='This is title of Document Object';
     document.bgColor='orange';
     document.fgColor='blue';
     document.write('<br>This is Example of Document Object');
     document.write('<br>Document Last Modified='+document.lastModified);
     document.write('<br>window width='+window.innerWidth);
     document.write('<br>window height='+window.innerHeight);
     window.location.reload(true);
    </script>
 </body>
</html>
```

Output



LOCATION OBJECT

- The location object contains information about the current URL.
- The location object is a property of the window object.
- The location object is accessed with:

window.location or just location

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

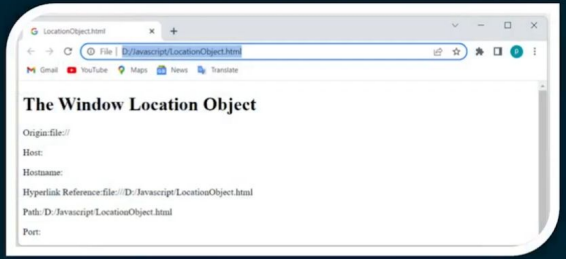
Location Object Methods

Method	Description
assign()	Loads a new document
reload()	Reloads the current document
replace()	Replaces the current document with a new one

Program

```
<!DOCTYPE html>
<html>
<body>
<h1>The Window Location Object</h1>
<p id="origin"></p>
<p id="host"></p>
<p id="hostname"></p>
<p id="href"></p>
<p id="pathname"></p>
<p id="port"></p>
<script>
 location.assign('file:///D:/Javascript/LocationObject.html');
 let origin = window.location.origin;
 document.getElementById("origin").innerHTML = 'Origin:'+origin;
 let host = window.location.host;
 document.getElementById("host").innerHTML = 'Host:'+host;
 let hostname = window.location.hostname;
 document.getElementById("hostname").innerHTML = 'Hostname:'+hostname;
 let href = window.location.href;
 document.getElementById("href").innerHTML = 'Hyperlink Reference:'+href;
 let pathname = window.location.pathname;
 document.getElementById("pathname").innerHTML = 'Path:'+pathname;
 let port = window.location.port;
 document.getElementById("port").innerHTML = 'Port:'+port;
</script>
</body>
</html>
```

Output



The Window History Object

The history object contains the URLs visited by the user (in the browser window).

The history object is a property of the window object.

The history object is accessed with: window.history or just history:

History Object Properties and Methods

Property/Method	Description
back()	Loads the previous URL (page) in the history list
forward()	Loads the next URL (page) in the history list
go()	Loads a specific URL (page) from the history list
length	Returns the number of URLs (pages) in the history list

Call() method in Javascript

The Function call is a predefined javascript method, which is used to write methods for different objects. It calls the method, taking the owner object as an argument. The keyword this refers to the "owner" of the function or the object it belongs to. All the functions in javascript are considered as object methods. So we can bound a function to a particular object by using 'call()'. A function will be the global object if the function is not considered as a method of a JavaScript object.

Syntax:

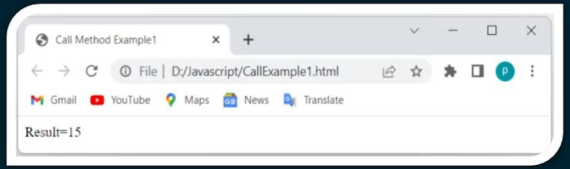
call()

Return Value: It calls and returns a method with the owner object being the argument.

Program(CallExample1.html)

```
<html>
  <head>
    <title>Call Method Example1</title>
  </head>
  <body>
    <script type="text/javascript">
       function sum(a,b)
       {
         return(a+b);
       }
       var result=sum.call(this,5,10);
       document.write('Result='+result);
    </script>
  </body>
</html>
```

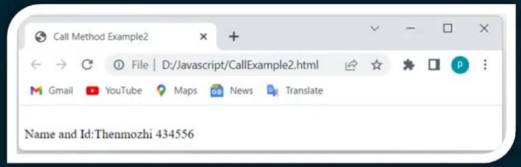
Output



Program(CallExample2.html)

```
<html>
  <head>
    <title>Call Method Example2</title>
  </head>
  <body>
    <script type="text/javascript">
       var employee =
       {
          details: function()
          {
            return this.name + " " + this.id;
          }
       }
       var emp1 =
       {
         name: "Karthick",
         id: 234412,
       }
       var emp2 =
       {
         name: "Thenmozhi",
         id: 434556,
       }
       var x = employee.details.call(emp2);
       document.write('<br> Name and Id:'+x);
    </script>
  </body>
</html>
```

Output

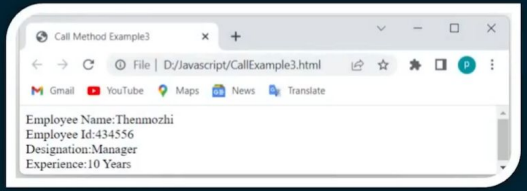


The example below describes the use of function call with arguments.

Program(CallExample3.html)

```
<html>
 <head>
  <title>Call Method Example3</title>
 </head>
 <body>
  <script type="text/javascript">
     var employee =
     {
       details: function(designation, experience)
       {
         return "Employee Name:"+this.name + "<br>" +
              "Employee Id:"+this.id+"<br>"+
              "Designation:"+designation +"<br>"+
              "Experience:"+experience;
       }
     }
     var emp1 =
     {
       name: "Karthick",
       id: 234412,
     }
     var emp2 =
     {
       name: "Thenmozhi",
       id: 434556,
     }
     var x = employee.details.call(emp2,"Manager","10 Years");
     document.write(x);
  </script>
 </body>
</html>
```

OUTPUT

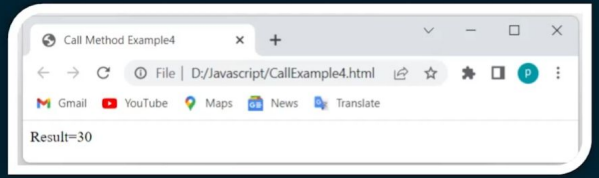


The example below describe bound a function to an object.

Program(CallExample4.html)

```
<html>
 <head>
  <title>Call Method Example4</title>
 </head>
 <body>
  <script type="text/javascript">
    var obj={a:10,b:20};
    function sum()
    {
      return this.a+this.b;
    }
    var result=sum.call(obj);
    document.write('Result='+result);
  </script>
 </body>
</html>
```

Output

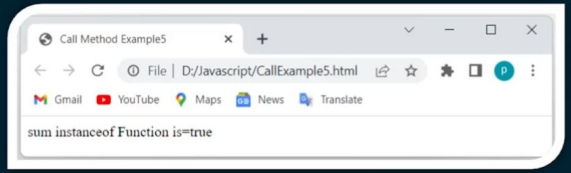


In JavaScript, a function is an instance of the Function type.

Program(CallExample5.html)

```
<html>
 <head>
  <title>Call Method Example5</title>
 </head>
 <body>
  <script type="text/javascript">
     function sum(a,b)
     {
       return(a+b);
     }
     document.write('sum instanceof Function is='+(sum instanceof Function));
  </script>
 </body>
</html>
```

Output

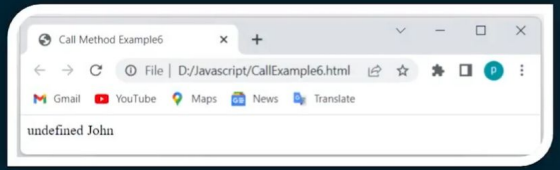


Note:that in the strict mode, the this inside the function is set to undefined instead of the global object.

Program(CallExample6.html)

```
<html>
 <head>
  <title>Call Method Example6</title>
 </head>
 <body>
  <script type="text/javascript">
     //var greeting = 'Hi';
              var messenger = {
                        greeting: 'Hello'
              }
              function say(name){
              document.write(this.greeting + ' ' + name);
            }
      say.call(this,'John');
  </script>
 </body>
</html>
```

OUTPUT

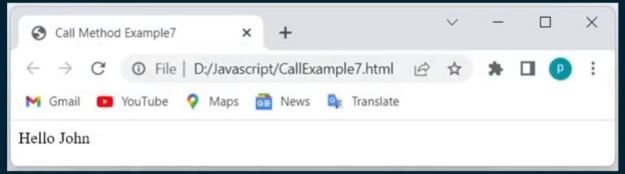


However, when you invoke the call() method of say function object and pass the messenger object as the this value:

Program(CallExample7.html)

```
<html>
 <head>
  <title>Call Method Example7</title>
 </head>
 <body>
  <script type="text/javascript">
     var greeting = 'Hi';
          var messenger = {
              greeting: 'Hello'
          }
          function say(name){
              document.write(this.greeting + ' ' + name);
          }
    say.call(messenger,'John');
  </script>
 </body>
</html>
```

Output



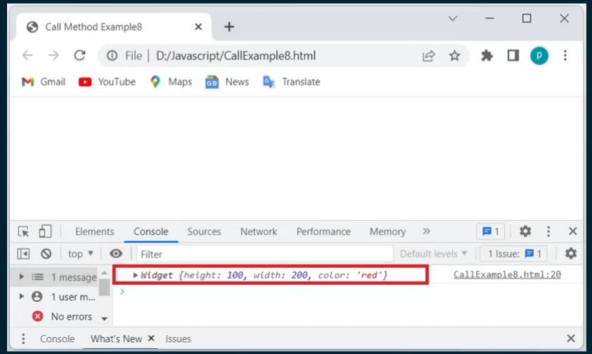
Using the JavaScript call() method to chain constructors for an object

You can use the call() method for chaining constructors of an object. Consider the following example:

Program(CallExample8.html)

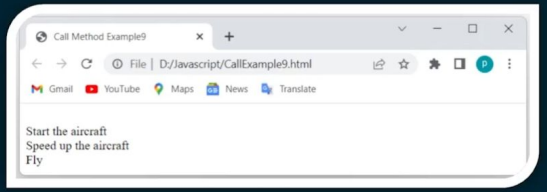
```
<html>
 <head>
  <title>Call Method Example8</title>
 </head>
 <body>
  <script type="text/javascript">
    function Box(height, width)
    {
      this.height = height;
      this.width = width;
    }
    function Widget(height, width, color)
    {
      Box.call(this, height, width);
      this.color = color;
    }
    let widget = new Widget(100, 200,'red');
    console.log(widget);
  </script>
 </body>
</html>
```

Output



Using the JavaScript call() method for function borrowing: Program(CallExample9.html)

```
<html>
 <head>
  <title>Call Method Example9</title>
 </head>
 <body>
  <script type="text/javascript">
    const car = {
     name: 'car',
     start() {
      document.write('<br>Start the ' + this.name);
     },
     speedUp() {
      document.write('<br>Speed up the ' + this.name);
     },
     stop() {
      document.write('<br>Stop the ' + this.name);
     },
    };
    const aircraft = {
     name: 'aircraft',
     fly() {
      document.write('<br>Fly');
     },
    };
    car.start.call(aircraft);
    car.speedUp.call(aircraft);
    aircraft.fly();
  </script>
 </body>
</html>
```

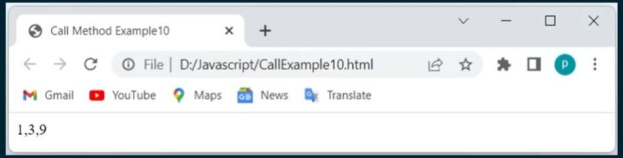


The following example illustrates how the arguments object borrows the filter() method of the Array.prototype via the call() function:

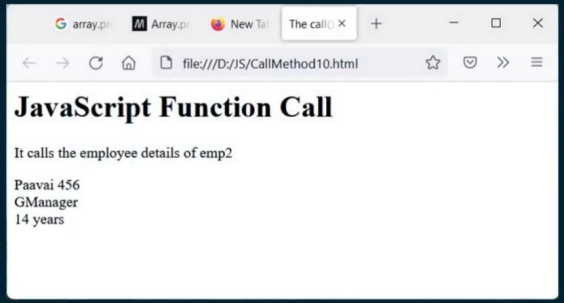
Program

```
<html>
  <head>
    <title>Call Method Example10</title>
  </head>
  <body>
    <script type="text/javascript">
      function isOdd(number)
      {
        return number % 2;
      }
      function getOddNumbers()
      {
        return Array.prototype.filter.call(arguments, isOdd);
      }
      let results = getOddNumbers(10, 1, 3, 4, 8, 9);
      document.write(results);
    </script>
  </body>
</html>
```

Output



```html
<!DOCTYPE html>
<html>
 <head>
   <title> The call() Method with Arguments </title>
 </head>
 <body>
   <h1>JavaScript Function Call</h1>
   <p> It calls the employee details of emp2 </p>
   <p id="demo"></p>
  <script>
   var employee = {
   details: function(designation, experience) {
          return this.name
          + " "
          + this.id + "<br>"
          + designation
          + "<br>"
          + experience;
       }
   }
   var emp1 = {
      name: "Aadhirai",
      id: "123",
   }
   var emp2 = {
      name: "Paavai",
      id: "456",
   }
   var x = employee.details.call(emp2, "GManager", "14 years");
   document.getElementById("demo").innerHTML = x;
  </script>
 </body>
</html>
```



Class in Javascript

JavaScript is prototype based functional language, classes were introduced in EcmaScript 2015 (ES6) to provide a cleaner way to follow object-oriented programming patterns.

JavaScript still follows a prototype-based inheritance model. Classes in JavaScript are the prototype-based inheritance model which we use to implement OOP concepts.

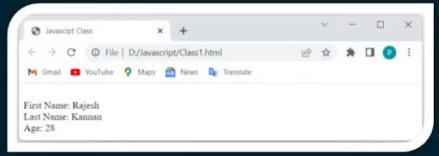
Thus the introduction of classes in JS made it easier for developers to build software around OOP concepts. It also brought in similarities to different OOP-based programming languages such as C++ and Java.

Before classes, we used constructor functions to do OOP in JavaScript. Have a look at the example below:

```
function Person(firstname, lastname, age) //constructor function
{
    this.firstname = firstname;
    this.lastname = lastname;
    this.age = age;
}
const p1 = new Person("Rajesh", "Kannan", 28); //constructor calling
console.log(p1);
```

Program with classes

```
<html>
  <head>
    <title>Javascipt Class</title>
  </head>
  <body>
    <script type="text/javascript">
      class Person
      {
        constructor(firstname, lastname, age) //constructor function
        {
          this.firstname = firstname;
          this.lastname = lastname;
          this.age = age;
        }
        showPersonDetails() //ordinary function
        {
          document.write(`<br>First Name: ${this.firstname}`);
          document.write(`<br>Last Name: ${this.lastname}`);
          document.write(`<br>Age: ${this.age}`);
        }
      }
      const p1 = new Person("Rajesh", "Kannan", 28); //constructor calling [p1 - object]
      p1.showPersonDetails(); //calling function
      //document.write(typeof Person); [Function]
    </script>
  </body>
</html>
```



JavaScript Class Inheritance

Inheritance enables you to define a class that takes all the functionality from a **parent class** and allows you to add more.

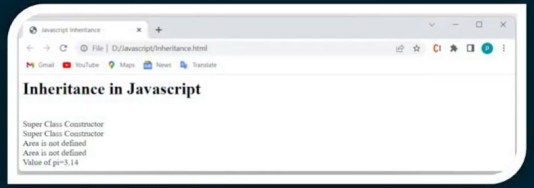
Using class inheritance, a class can inherit all the **methods** and **properties** of another class.

Inheritance is a useful feature that allows code **reusability**.

To use class inheritance, you use the **extends** keyword.

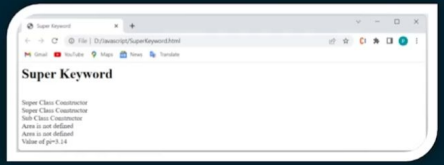
Program(Inheritance.html)

```
<html>
  <head>
    <title>Javascript Inheritance</title>
  </head>
  <body>
    <h1>Inheritance in Javascript</h1>
    <script language="javascript">
    class Area
    { _pi=3.14;  //protected field
       constructor()
       {
         document.write("<br>Super Class Constructor");
       }
       showArea()
       {
         document.write("<br>Area is not defined");
       }
    }
    class Circle extends Area
    {
      getpi()
      {
        document.write("<br>Value of pi="+this._pi);
      }
    }
    a=new Area(); //Constructor calling
    c=new Circle(); //super class Constructor calling
    a.showArea();
    c.showArea();
    c.getpi();
   </script>
  </body>
</html>
```



The super keyword used inside a child class denotes its parent class.

```
<html>
 <head>
  <title>Super Keyword</title>
 </head>
 <body>
  <h1>Super Keyword</h1>
  <script language="javascript">
  class Area
  { _pi=3.14;  //protected field
     constructor()
     {  document.write("<br>Super Class Constructor");
     }
     showArea()
     { document.write("<br>Area is not defined");
     }
  }
  class Circle extends Area
  {  constructor()
    {  super(); //calling super class constructor
       document.write("<br>Sub Class Constructor");
    }
    getpi()
    { document.write("<br>Value of pi="+this._pi);
    }
  }
  a=new Area(); //Constructor calling
  c=new Circle(); //super class Constructor calling
  a.showArea();
  c.showArea();
  c.getpi();
  </script>
 </body>
</html>
```



Overriding Method or Property

If a child class has the same method or property name as that of the parent class, it will use the method and property of the child class. This concept is called method overriding.

Program(MethodOverriding.html)

```
<html>
 <head>
   <title>Method Overriding</title>
 </head>
 <body>
   <h1>Method Overriding</h1>
   <script language="javascript">
   class Area
   {
      _pi=3.14;  //protected field
      _area=0;  //protected field
      constructor()
      {
        document.write("<br>Super Class Constructor");
      }
      showArea()
      {
        document.write("<br>Area is not defined");
      }
   }
```

Hierarichal Inheritance -Program(Inheritance3.html)

```
<html>
  <head>
    <title>Javascript Inheritance</title>
  </head>
  <body>
     <h1>Hierarichal Inheritance in Javascript</h1>
     <script language="javascript">
     class Area
     {
        _pi=3.14;  //protected field
        _area=0;   //protected field
        constructor()
        {
          document.write("<br>Area is the Super Class");
        }
        showArea()
        {
          document.write("<br>Area is not defined");
        }
     }
     class Circle extends Area
     {
        #radius=0;   //private field
        constructor(value)
        {
          super();
          this.#radius=value;
          document.write("<br>Circle extends Area");
        }
```

```
/*method overriding here*/
        showArea()
        {
          this._area=this._pi*this.#radius*this.#radius;
          document.write("<br>Area of Circle="+this._area);
        }
    }
    class Square extends Area
    {
        #a=0;   //private field
        constructor(value)
        {
          super();
          this.#a=value;
          document.write("<br>Square extends Area");
        }
        /*method overriding here*/
        showArea()
        {
          this._area=this.#a*this.#a;
          document.write("<br>Area of Square="+this._area);
        }    }
```

```
class Rectangle extends Area
  {
    #l=0;  //private field
    #b=0;  //private field
    constructor(value1,value2)
    {
      super();
      this.#l=value1;
      this.#b=value2;
      document.write("<br>Rectangle extends Area");
    }
    /*method overriding here*/
    showArea()
    {
      this._area=this.#l*this.#b;
      document.write("<br>Area of Rectangle="+this._area);
    }
  }
  //a=new Area(); //constructor calling
  c=new Circle(7); //constructor calling
  c.showArea();
  s=new Square(5); //constructor calling
  s.showArea();
  r=new Rectangle(8,10); //constructor calling
  r.showArea();
 </script>
 </body>
</html>
```

Output

Using call() to chain constructors for an object -You can use call to chain constructors for an object (similar to Java).

Program(CallMethod3.html)

```
<html>
  <head>
    <title>Call Method in Javascript</title>
  </head>
  <body>
    <h1>Call Method in Javascript</h1>
    <script type="text/javascript">
      function Product(name, price)
      {
        this.name = name;
        this.price = price;
      }
      function Food(name, price)
      {
        Product.call(this, name, price);
        this.category = 'food';
      }
      function Toy(name, price)
      {
        Product.call(this, name, price);
        this.category = 'toy';
      }
      obj1=new Food("Cheese",5);
      obj2=new Toy("Robot",40);
      document.write('<br>Food Name='+obj1.name);
      document.write('<br>Food Price='+obj1.price);
      document.write('<br>Toy Name='+obj2.name);
      document.write('<br>Toy Price='+obj2.price);
    </script>
  </body>
</html>
```

Output

Using call() to invoke an anonymous function

Here, we create an anonymous function and use call to invoke it on every object in an array. The purpose of the anonymous function is to add print function to every object, which is able to print the correct index of the object in the array.

Program(CallMethod4.html)

```
<html>
 <head>
  <title>Call Method in Javascript</title>
 </head>
 <body>
  <h1>Call Method in Javascript</h1>
  <script type="text/javascript">
    const animals =
    [
      {species: 'Lion', name: 'King' },
      {species: 'Whale', name: 'Fail' }
    ];
    for(let i = 0; i < animals.length; i++)
    {
     (function(i)
     {
      this.print = function()
      {
        document.write('<br>#' + i + ' ' + this.species+ ': ' + this.name);
      }
      this.print();
    }).call(animals[i], i);
    }
  </script>
 </body>
</html>
```

Using call() to invoke a function and specifying the context for 'this'

Program(CallMethod5.html)

```
<html>
 <head>
  <title>Call Method in Javascript</title>
 </head>
 <body>
  <h1>Call Method in Javascript</h1>
  <script type="text/javascript">
    function greet()
    {
     const reply = [this.animal, 'typically sleep between', this.sleepDuration].join(' ');
     document.write(reply);
    }
    const obj =
    {
     animal: 'cats', sleepDuration: '12 and 16 hours'
    };
    greet.call(obj);  // cats typically sleep between 12 and 16 hours
  </script>
 </body>
</html>
```

Output

Document Object Model(DOM)

- The Document Object Model (DOM) is an application programming interface (API) for HTML -CSS and XML documents.
- It defines the logical structure of documents and the way a document is accessed and manipulated - allows us to create, change, or remove elements from the document.
- We can also add events to these elements to make our page more dynamic.
- The DOM views an HTML document as a tree of nodes. A node represents an HTML element.
- When the HTML document is loaded into the browser, there is will be another representation to the document that is called as DOM document.
- With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object.
- Using DOM, the JavaScript gets access to HTML as well as CSS of the web page and can also add behavior to the HTML elements. so basically Document Object Model is an API that represents and interacts with HTML or XML documents.

The model is called a DOM tree, and it is stored in the browsers' memory. It consists of four main types of nodes.

1. Document Node
2. Element Nodes
3. Attribute Nodes
4. Text Nodes

Each node is an object with methods and properties. Scripts access and update this DOM tree (not the source HTML file). Any changes made to the DOM tree are reflected in the browser.

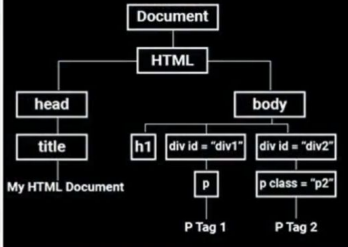
What is Document Object Model ?

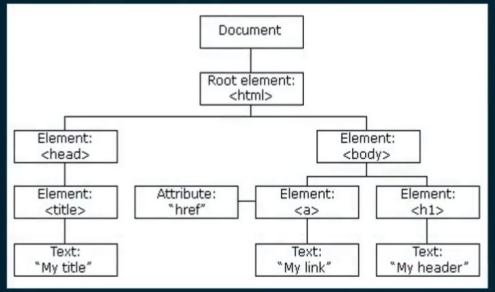
HTML Document

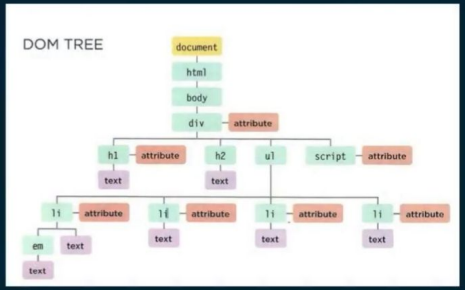
index.html

```
<html>
    <head>
        <title>My HTML Document</title>
    </head>

    <body>
        <h1>Heading</h1>
        <div id="div1">
            <p>P Tag 1</p>
        </div>
        <div id="div2">
            <p class="p2">P Tag 2</p>
        </div>
    </body>
</html>
```

Document Object Model (DOM)







1. Document Node

- As you see the above DOM tree that every element, attribute, and piece of text in the HTML is represented by its own DOM node.
- At the top of the tree a document node is added; it represents the entire page (and also corresponds to the document object).
- When you access any element, attribute, or text node, you navigate to it via the document node. It is the starting point for all visits to the DOM tree.

2. Element Nodes

- HTML elements describe the structure of an HTML/web page.
- (The <h1>-<h6> elements describe what parts are headings; the <p> tags indicate where paragraphs of text start and finish; and so on.)
- To access the DOM tree, you start by looking for elements. Once you find the element you want, then you can access its text and attribute nodes if you want to. This is why you start by learning methods that allow you to access element nodes, before learning to access and alter text or attributes
- Relationships between the document and all of the element nodes are described using the same terms as a family tree: parents, children, siblings, ancestors, and descendants. (Every node is a descendant of the document node.)

3. Attribute Nodes

- The opening tags of HTML elements can carry attributes and these are represented by attribute nodes in the DOM tree.
- Attribute nodes are not children of the element that carries them; they are part of that element. Once you access an element, there are specific JavaScript methods and properties to read or change that element's attributes.
- For example, it is common to change the values of class attributes to trigger new CSS rules that affect their presentation.

4. TEXT NODES

- Once you have accessed an element node, you can then reach the text within that element. This is stored in its own text node.
- Text nodes cannot have children. If an element contains text and another child element, the child element is not a child of the text node but rather a child of the containing element.

WORKING WITH THE DOM TREE

Accessing and updating the DOM tree involves two steps:

1. Locate the node that represents the element you want to work with.
2. Use its text content, child elements, and attributes.

The search methods available for DOM include:

- **getElementById**

This method returns a reference to the first element that has the specified ID.

- **getElementsByTagName**

This method returns the active NodeList, which has the specified TAG name.

- **getElementsByName**

This method returns the active NodeList, which has the specified NAME. This is a preferred method for option buttons.

- **getElementsByClass**

This method returns the active NodeList, which has the specified CLASS name.

- **querySelector**

This method accepts CSS selector as parameter and return the first matched element.

- **querySelectorAll**

This method accepts CSS selector as parameter and return all the matched elements. Therefore, it returns a static NodeList.

- **parentNode**

Selects the parent of the current element node (which will return just one element).

- **previousSibling / nextSibling**

Selects the previous or next sibling from the DOM tree.

- **firstChild / lastChild**

Program(Find the Siblings of tags)

```
<html>
  <head>
    <title>Query Selector</title>
  </head>
  <body>
      <h1>Query Selector</h1>
      <ul>
        <li class='item1'>HTML</li>
        <li class='item2'>CSS</li>
        <li class='item3' style='color:red'><b>Javascript</b></li><br>
        <li class='item4' style='color:red'>JQuery</li>
        <li class='item5'>Bootstrap</li>
      </ul>
      <script type="text/javascript">
      var x=document.querySelector('li'); //first element
      document.write('First Item using Query Selector='+x.innerHTML);

      document.write('<br><br><b>Display All the items using QuerySelector All</b>');
      let listItems = document.querySelectorAll('ul > li');
      for(i=0;i<listItems.length;i++)
      {
        document.write('<br>'+listItems[i].innerHTML);
      }
      var startItem = document.getElementsByTagName('ul')[0] ;
      var firstChild = startItem.firstElementChild.innerHTML;
      var lastChild = startItem.lastElementChild.innerHTML;
```

```
    document.write('<br>Last Child of &lt;ul&gt; Tag='+lastChild);
    document.write('<br><br><b>Display list having style attribute</b>');
    var x=document.querySelectorAll('li[style]'); //select elements having class name item
    for(i=0;i<x.length;i++)
    {
      document.write('<br>'+x[i].innerHTML);
    }
    var startItem = document.querySelector('.item1');
    document.write('<br>First Sibling='+startItem.innerHTML);
    var nextSibling = startItem.nextElementSibling;
    document.write('<br><b><u>Forward Siblings</u></b>');
    while(nextSibling)
    {
      document.write('<br>'+nextSibling.innerHTML);
      nextSibling = nextSibling.nextElementSibling;
    }
    var lastItem = document.querySelector('ul').lastElementChild;
    document.write('<br>Last Item='+lastItem.innerHTML);
    var prevSibling = lastItem.previousElementSibling;
    document.write('<br><b><u>Backward Siblings</u></b>');
   while(prevSibling)
    {
      document.write('<br>'+prevSibling.innerHTML);
      prevSibling = prevSibling.previousElementSibling;
    }
</script>
  </body>
```

Output

Query Selector

- HTML
- CSS
- **Javascript**

- JQuery
- Bootstrap

First Item using Query Selector=HTML

Display All the items using QuerySelector All
HTML
CSS
Javascript
JQuery
Bootstrap
First Child of <ul> Tag=HTML
Last Child of <ul> Tag=Bootstrap

Display list having style attribute
Javascript
JQuery
First Sibling=HTML
Forward Siblings
CSS
Javascript

JQuery
Bootstrap
Last Item=Bootstrap
Backward Siblings
JQuery

Javascript
CSS
HTML

Program

```
<html>
 <head>
   <title>DOM Example</title>
 </head>
 <body>
   <h1>Web Programming Language</h1>
          <ul>
             <li>HTML</li>
             <li>CSS</li>
             <li>JavaScript</li>
          </ul>
          <p id="demo"></p>
   <script language="javascript">
     var x = document.getElementsByTagName("li");
     document.getElementById("demo").innerHTML = '1st list item:' +x[0].innerHTML+
     '<br>2nd list item: ' + x[1].innerHTML+
     '<br>3rd list item: ' + x[2].innerHTML;
       document.getElementById("demo").style.color = "red";
   </script>
 </body>
```

output

The HTML DOM – Changing Styles

Program

```
<html>
 <head>
  <title>DOM Changing Style</title>
 </head>
 <body>
  <h1>Web Programming Language</h1>
          <ul>
            <li>HTML</li>
            <li>CSS</li>
            <li>JavaScript</li>
          </ul>
          <p id="demo"></p>
  <script language="javascript">
   var x = document.getElementsByTagName("li");
   document.getElementById("demo").innerHTML = "1st list item:" +x[0].innerHTML+
   "<br>2nd list item: " + x[1].innerHTML+
   "<br>3rd list item: " + x[2].innerHTML;
     document.getElementById("demo").style.color = "red";
    x[0].style.color="red";
    x[1].style.color="blue";
    x[2].style.color="orange";
    x[2].style.fontSize="16pt";
    x[2].style.borderStyle="dashed";
    x[2].style.borderSize="100px";
    x[2].style.borderWidth="2pt";
  </script>
 </body>
```

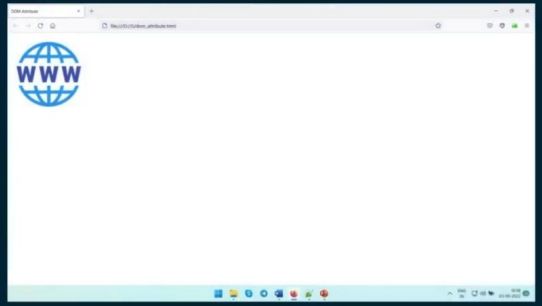
output

The HTML DOM – Changing Attributes

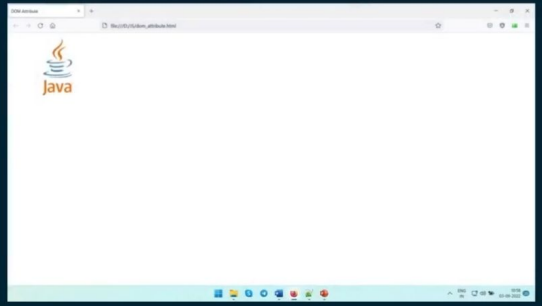
Program

```
<html>
  <head>
    <title>DOM Example3</title>
  </head>
  <body>
     <img id="image1" src="/images/web.png" onmousemove="MouseMove()"
onmouseout="MouseOut()"></image>
     <script language="javascript">
        function MouseMove()
        {
          document.getElementById("image1").src="/images/java.jpg";
        }
        function MouseOut()
        {
          document.getElementById("image1").src="web.png";
        }
     </script>
  </body>
</html>
```

Output(Before MouseMove)



Output(After MouseMove)



The HTML DOM – Building HTML: Program

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>DOM Example5</title>
</head>
<body>
<div id="theDiv">
<h1>Hello, World! !</h1>
<p id="one">This is text from paragraph 1</p>
<p id="two">This is text from paragraph 2</p>
<p id="four">This is text from paragraph 4</p>
<button onclick="remv(document.getElementById('two'))">Click to remove p2</button>
</div>
<script src="BuildDOM.js"></script>
</body>
</html>
```

BuildDOM.js

```
var newPara = document.createElement("p");
var content = document.createTextNode("This is a new paragraph.");
newPara.appendChild(content);
var divElem = document.getElementById("theDiv");
divElem.appendChild(newPara);
function remv(element)
{
  element.parentNode.removeChild(element);
}
```

Cookies

A cookie is a piece of data that is stored on your computer to be accessed by your browser.

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

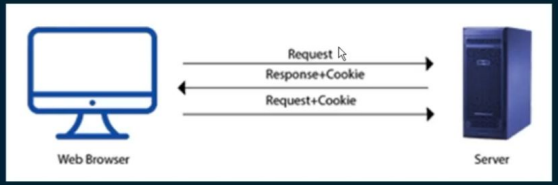
- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

Why do you need a Cookie?

The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

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.



Javascript Set Cookie

You can create cookies using document.cookie property like this.
document.cookie = "cookiename=cookievalue"

You can even add expiry date to your cookie so that the particular cookie will be removed from the computer on the specified date. The expiry date should be set in the UTC/GMT format. If you do not set the expiry date, the cookie will be removed when the user closes the browser.
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2022 20:00:00 UTC"

You can also set the domain and path to specify to which domain and to which directories in the specific domain the cookie belongs to. By default, a cookie belongs to the page that sets the cookie.
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2022 20:00:00 UTC; path=/ "

JavaScript get Cookie

You can access the cookie like this which will return all the cookies saved for the current domain.
var x = document.cookie

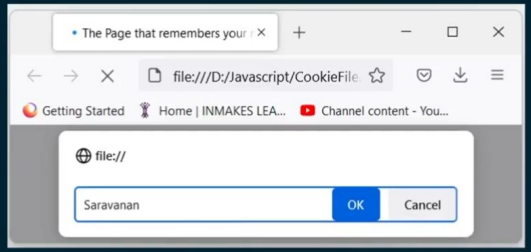
JavaScript Delete Cookie

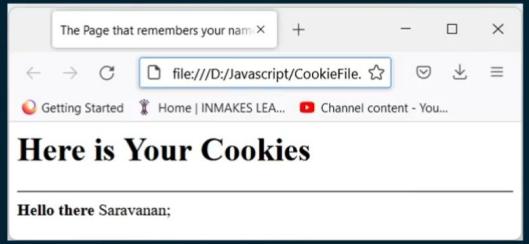
To delete a cookie, you just need to set the value of the cookie to empty and set the value of expires to a passed date.
: "cookiename= ; expires = Thu, 05 Jan 2022 00:00:00 GMT"

Program

```
<html>
 <head>
  <title>The Page that remembers your name</title>
  <script>
     if(document.cookie.substring(0,2)!="n=")
            {
              nam=window.prompt("Enter your name");
              document.cookie=" "+nam+";";
              document.cookie+="expires=Sunday, 12-Jun-2022 06:05:00 GMT";
     }
  </script>
 </head>
 <body>
  <h1>Here is Your Cookies</h1>
          <hr>
          <script>
             indx=document.cookie.indexOf(";");
          nam=document.cookie.substring(0,indx+1);
       document.write("<b>Hello there </b>"+nam);
          </script>
 </body>
```

Output


The Page that remembers your
file:///D:/Javascript/CookieFile
Getting Started
Home | INMAKES LEA...
Channel content - You...
file://
Saravanan
OK
Cancel


The Page that remembers your nam
file:///D:/Javascript/CookieFile.
Getting Started
Home | INMAKES LEA...
Channel content - You...
Here is Your Cookies
Hello there Saravanan;