2nd Semester MCA02005 Internet and Web Programming

L-T-P 3-0-0 3 CREDITS

Module I (8 Periods)
Internet Architecture: Internet overview, evolution of internet. Internet components: Local Area Networks, Access Networks, Core Networks, Routers, Transmission infrastructure, ISPs. TCP/IP model, TCP/IP vs OSI model. HTML: HTML Overview, Structure of HTML Documents, Document Types, HTML Elements and attributes. Anchor Attributes, Image Tag and its attributes, Image and Anchors, Table.
Module II (8 Periods)
Image Map: Attributes, Client Side Image Maps and Server Side Maps.
HTML Layout: Background, colors and text, Tables, Frames, Layers, Page content Division <Div>, <SPAN>. CSS: Style Sheet Basic, Properties, Positioning with Style Sheet.
Forms: <FORM> Elements, Form controls. Dynamic HTML.
Module III (8 Periods)
Java Script: Introduction, Client-Side JavaScript, Server-Side JavaScript, JavaScript Objects, JavaScript Security. Operators: Assignment Operators, Comparison Operators, Arithmetic Operators, Increment, Decrement, Unary Negation, Logical Operators, String Operators, Special Operators, Conditional operator, Comma operator, delete, new, this, void.
Statements: Break, comment, continue, delete, do ... while, export, for, for...in, function, if...else, import, labelled, return, switch, var, while.
Module IV (8 Periods)
JavaScript (Properties and Methods of Each) :Array, Boolean, Date, Function, Math, Number, Object, String, regExp. Document and its associated objects, document, Link, Area, Anchor, Image, Applet, Layer.
Events and Event Handlers: General Information about Events, Defining Event Handlers, event.
Module V (8 Periods)
Server Side Programming: Common Gateway Interface (CGI), Active Server Pages.
Internet applications: FTP, Telnet, Email, Chat. World Wide Web: HTTP protocol. Search Engines. E-commerce and security issues including symmetric and asymmetric key, encryption and digital signature, and authentication. Emerging trends, Internet telephony, and virtual reality over the web, etc. Intranet and extranet, firewall.
Books:
1. Computer Networking: A Top-Down Approach Featuring the Internet by Kurose and Ross, Pearson.
2. Web Design the Complete Reference by Thomas Powell, Tata McGrawHill.
3. HTML The Complete Reference by Thomas Powell, Tata McGrawHill.

JavaScript - Dialog Boxes / Popup Boxes

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

Alert Dialog Box

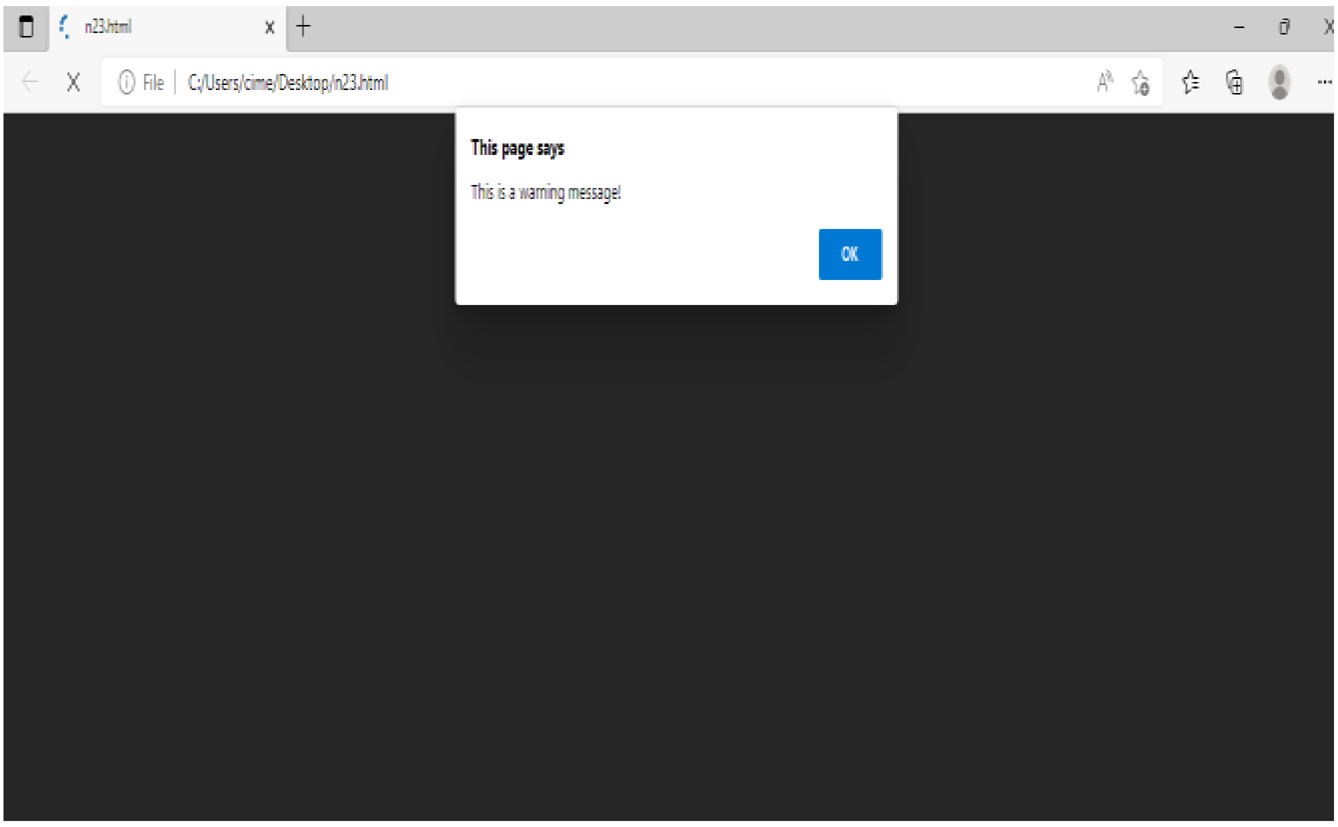
An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example

```
<html>
   <body>
    <script type = "text/javascript">
                    alert ("This is a warning message!");
      </script>
   </body>
</html>
```

O/P



This page says

This is a warning message!

Confirmation Dialog Box

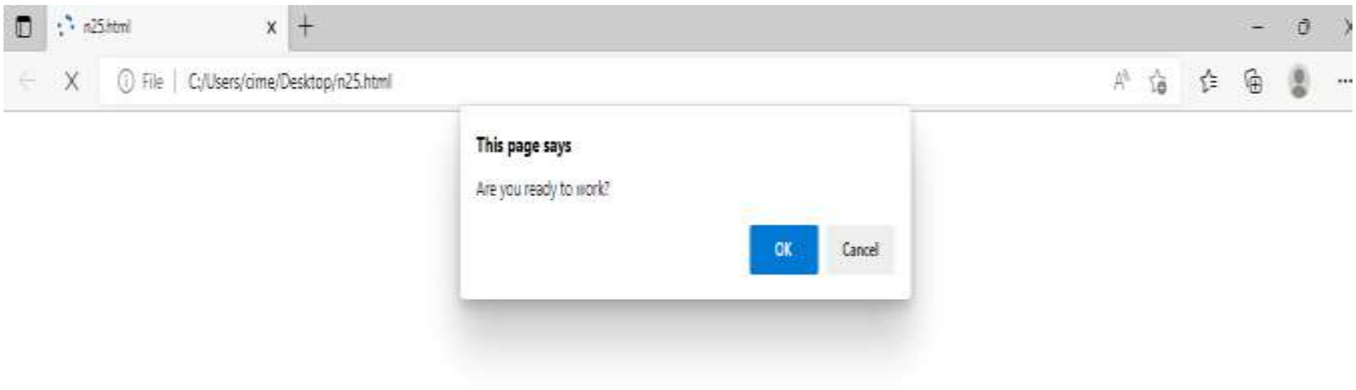
A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false.
Ex

```
<html>
  <body>
   <script type = "text/javascript">
var c=confirm("Are you ready to work");

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

O/P



This page says

Are you ready to work?

Prompt Dialog Box
The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

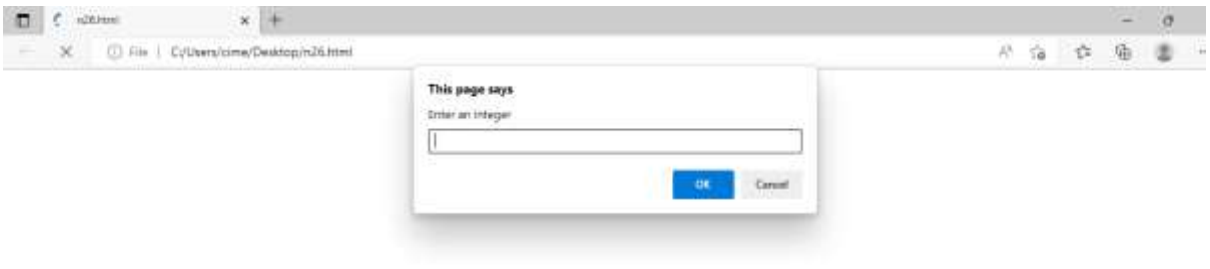
This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Ex

```
<html>
  <body>
   <script type = "text/javascript">
var c=prompt("Enter an Integer");

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

O/P



```
This page says
Enter an integer
```

Program to check if a number is prime or not

```
<!DOCTYPE html>
<html>
<body>
<script type = "text/javascript">
var n, i, flag = true;
var n=prompt("Enter an Integer");
for(i = 2; i <= n - 1; i++)
if (n % i == 0)
{
flag = false;
break;
}
if (flag == true)
alert(n + " is prime");
else
alert(n + " is not prime");
</script>
</body>
</html>
```

Program to calculate factorial of a number

```
<!DOCTYPE html>
<html>
<body>
<script >
```

```
var n, i, fact=1;
var n=prompt("Enter an Integer");
if(n>0)
{
        for(i=1;i<=n;i++)
        {
                fact = fact * i;

        }
}
else if(n==0)
{
fact=1;
}
alert(" Factorial of "+ n +" is "+ fact);

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

JavaScript Arrays
An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Why Use Arrays?
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array
Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the const keyword.

Ex
```
<!DOCTYPE html>
<html>
<body><script>
const cars = ["Saab", "Volvo", "BMW"];
for(i in cars)
{
document.write(cars[i]+ "<br />");
}
```

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

O/P

```
Saab
Volvo
BMW
```

Spaces and line breaks are not important. A declaration can span multiple lines:

```
<!DOCTYPE html>
<html>
<body>
<script>
const cars = new Array("Saab", "Volvo", "BMW");
for(i in cars)
{
document.write(cars[i]+ "<br />");
}
</script>

</body>
</html>
```

O/P

```
Saab
Volvo
BMW
```

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

```
<!DOCTYPE html>
<html>
<body>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.write(cars);
</script>
</body>
</html>
```

O/P

```
Saab,Volvo,BMW
```

Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

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



<script>
const person = ["John", "Doe",'g',46,6.7,true];
for(i in person)
{
document.write(person[i]+ "<br />");
}
</script>
</body>
</html>
```

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.length);
</script>
</body>
</html>
```

O/P

```
4
```

Looping Array Elements

One way to loop through an array, is using a for loop:

Example

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

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

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

<script>
```

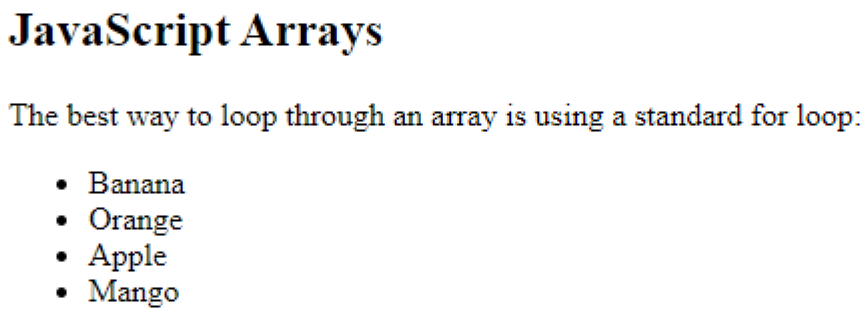
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Output



```
JavaScript Arrays

The best way to loop through an array is using a standard for loop:

  • Banana
  • Orange
  • Apple
  • Mango
```

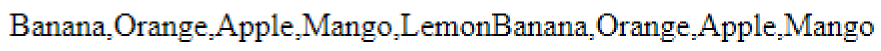
Adding Array Elements

The easiest way to add a new element to an array is using the push() method:

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Mango");
fruits.push("Lemon");
document.write(fruits);
fruits.pop("Lemon");
document.write(fruits);
</script>
</body>
</html>
```

Output



```
Banana,Orange,Apple,Mango,LemonBanana,Orange,Apple,Mango
```

Associative Arrays

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does not support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.

WARNING !!
If you use named indexes, JavaScript will redefine the array to an object.
After that, some array **methods and properties will produce incorrect results.**

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
person=[];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person[3] = 4.68;
person[4] = 48.68;
document.write(person);
</script>
</body>
</html>
```
O/P
```
John,Doe,46,4.68,48.68
```
Ex-
```
<!DOCTYPE html>
<html>
<body>
<script>
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.write(person.length);
</script>
</body>
</html>
```
O/P
```
3
```

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.write(person[1]);
</script>
</body>
</html>
```

O/P
undefined

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.write(person.length);
</script>
</body>
</html>
```
O/P

```
0
```

The Difference between Arrays and Objects

In JavaScript, arrays use numbered indexes.
In JavaScript, objects use named indexes.

Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

JavaScript does not support associative arrays.
You should use objects when you want the element names to be strings (text).
You should use arrays when you want the element names to be numbers.

```
<!DOCTYPE html>
<html>
<body>
<script>
var points = new Array(40, 100, 1);
document.write(points[1]);
</script>
</body>
</html>
```
O/P
```
100
```

```
<!DOCTYPE html>
```

```
<html>
<body>
<script>
var points = [40];
document.write(points.length);
</script>
</body>
</html>
```

O/P

```
1
```

```
<!DOCTYPE html>
<html>
<body>
<script>
var points = new Array(40);
document.write(points.length);
</script>
</body>
</html>
```

O/P

```
40
```

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.toString());
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
```

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write( fruits.join(" * "));
</script>
</body>
```

```
</html>
```
O/P
```
Banana * Orange * Apple * Mango
```
JavaScript Array push()
The push() method adds a new element to an array (at the end):
Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write((fruits)+"<br>");
fruits.push("Kiwi");
document.write(fruits);
</script>
</body>
</html>
```
O/P
```
Banana,Orange,Apple,Mango
Banana,Orange,Apple,Mango,Kiwi
```

JavaScript Array pop()
The pop() method removes the last element from an array.
```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write((fruits)+"<br>)");
fruits.pop();
document.write(fruits);
</script>
</body>
</html>
```
O/P
```
Banana,Orange,Apple,Mango
Banana,Orange,Apple
```
JavaScript Array shift()
The shift() method removes the first array element and "shifts" all other elements to a lower index.
Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
document.write(fruits);
</script>
</body>
```

```
</html>
```

O/P

```
Orange,Apple,Mango
```

JavaScript Array unshift()

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits + "<br>");
fruits.unshift("Lemon");
document.write(fruits);
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
Lemon,Banana,Orange,Apple,Mango
```

The unshift() method returns the new array length.

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.unshift("Lemon")+"<br>");
document.write(fruits.length+"<br>");
document.write(fruits);
</script>
</body>
</html>
```

O/P

```
5
5
Lemon,Banana,Orange,Apple,Mango
```

Changing Elements

Array elements are accessed using their index number:

Array indexes start with 0:

[0] is the first array element

[1] is the second

[2] is the third ...

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

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write((fruits)+"<br>");
fruits[0] = "Kiwi";
document.write(fruits);
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
Kiwi,Orange,Apple,Mango
```

JavaScript Array delete()
Warning !
Array elements can be deleted using the JavaScript operator delete.

Using delete leaves undefined holes in the array.
Use pop() or shift() instead.
Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write("The first fruit is: " + fruits[0]+"<br>");
delete fruits[0];
document.write("The first fruit is: " + fruits[0]);
</script>
</body>
</html>
```

O/P

```
The first fruit is: Banana
The first fruit is: undefined
```

Merging (Concatenating) Arrays
The concat() method creates a new array by merging (concatenating) existing arrays.

```
<!DOCTYPE html>
<html>
<body>
<script>
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);
document.write(myChildren);
</script>
</body>
</html>
```

O/P

```
Cecilie,Lone,Emil,Tobias,Linus
```

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments.

The concat() method can also take strings as arguments.

Example (Merging an Array with Values)
```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```

Splicing and Slicing Arrays
The splice() method adds new items to an array.

The slice() method slices out a piece of an array.
```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write((fruits)+"<br>");
fruits.splice(2, 0, "Lemon", "Kiwi");
document.write(fruits);
</script>
</body>
</html>
```
O/P
```
Banana,Orange,Apple,Mango
Banana,Orange,Lemon,Kiwi,Apple,Mango
```

The first parameter (2) defines the position where new elements should be added (spliced in).

The second parameter (0) defines how many elements should be removed.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

The splice() method returns an array with the deleted items.

```
<!DOCTYPE html>
<html>
<body>
<script>
fruits=[];
fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits+"<br>");
fruits.splice(4,2,"Lemon", "Kiwi");
document.write(fruits+"<br>");
</script>
```

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

O/P

```
Banana,Orange,Apple,Mango
Banana,Orange,Apple,Mango,Lemon,Kiwi
```

JavaScript Array slice()

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 2.

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
document.write(fruits.slice(2));
</script>
</body>
</html>
```

O/P

```
Lemon,Apple,Mango
```

Note

The slice() method creates a new array.
The slice() method does not remove any elements from the source array.

Sorting an Array

The sort() method sorts an array alphabetically:

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits+"<br>");
fruits.sort();
document.write(fruits+"<br>");
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
Apple,Banana,Mango,Orange
```

Reversing an Array

The reverse() method reverses the elements in an array.
You can use it to sort an array in descending order.

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits+"<br>");
fruits.reverse();
document.write(fruits+"<br>");
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
Mango,Apple,Orange,Banana
```

Numeric Sort

The Compare Function

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a - b}
```

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative a is sorted before b.
If the result is positive b is sorted before a.

If the result is 0 no changes are done with the sort order of the two values.

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
</script>
</body>
</html>
```

O/P

```
1,5,10,25,40,100
```

Use the same trick to sort an array descending:

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
document.write((points)+"<br>");
</script>
</body>
</html>
```
O/P
```
100,40,25,10,5,1
```

Sorting an Array in Random Order
```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(){return 0.5 - Math.random()});
document.write((points)+"<br>");
</script>
</body>
</html>
```
O/P
```
25,40,10,100,1,5
40,5,25,100,1,10
1,100,40,5,10,25
```

Find the Lowest Array Value

Example
```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40, 100, 13, 5, 25, 10];
points.sort(function(a, b){return a-b});
document.write(points[0]);
</script>
</body>
</html>
```
O/P
```
5
```

Example
```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40, 100,234, 15, 5, 25, 10];
document.write(Math.min.apply(null, points));
</script>
</body>
</html>
```

O/P

```
5
```

Find the Highest Array Value

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40,2023, 1005, 13, 5, 25, 10];
points.sort(function(a, b){return b - a});
document.write(points[0]);
</script>
</body>
</html>
```

O/P

```
2023
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
const points = [40, 100,234, 1, 5, 25, 10];
document.write(Math.max.apply(null, points));
</script>
</body>
</html>
```

O/P

```
234
```

JavaScript Array forEach()

The forEach() method calls a function (a callback function) once for each array element.

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
students = [12,54,87,54];
students.forEach(myFunction);
function myFunction(item) {
   document.write((item )+"<br>");
}
</script>
</body>
</html>
```

O/P

```
12
54
87
54
```

Note that the function takes 3 arguments:

The item value
The item index
The array itself

Elements Can be Reassigned
You can change the elements of a constant array:
Assigned when Declared
JavaScript const variables must be assigned a value when they are declared:
Meaning: An array declared with const must be initialized when it is declared.
Using const without initializing the array is a syntax error:

Example
This will not work:

```
const cars;
cars = ["Saab", "Volvo", "BMW"];
```

Arrays declared with var can be initialized at any time.

You can even use the array before it is declared:

Example
This is OK:

```
cars = ["Saab", "Volvo", "BMW"];
var cars;
```

Const Block Scope
An array declared with const has Block Scope.

An array declared in a block is not the same as an array declared outside the block:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
// Here cars[0] is "Saab"
{
  const cars = ["Toyota", "Volvo", "BMW"];
  // Here cars[0] is "Toyota"
}
// Here cars[0] is "Saab"
```

An array declared with var does not have block scope:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
// Here cars[0] is "Saab"
{
  var cars = ["Toyota", "Volvo", "BMW"];
  // Here cars[0] is "Toyota"
}
// Here cars[0] is "Toyota"
```

You can learn more about Block Scope in the chapter: JavaScript Scope.

Redeclaring Arrays

Redeclaring an array declared with var is allowed anywhere in a program:

Example

```
var cars = ["Volvo", "BMW"];   // Allowed
var cars = ["Toyota", "BMW"];  // Allowed
cars = ["Volvo", "Saab"];      // Allowed
```

Redeclaring or reassigning an array to const, in the same scope, or in the same block, is not allowed:

Example

```
var cars = ["Volvo", "BMW"];     // Allowed
const cars = ["Volvo", "BMW"];   // Not allowed
{
  var cars = ["Volvo", "BMW"];   // Allowed
  const cars = ["Volvo", "BMW"]; // Not allowed
}
```

Redeclaring or reassigning an existing const array, in the same scope, or in the same block, is not allowed:

Example

```
const cars = ["Volvo", "BMW"];   // Allowed
const cars = ["Volvo", "BMW"];   // Not allowed
var cars = ["Volvo", "BMW"];     // Not allowed
cars = ["Volvo", "BMW"];         // Not allowed

{
  const cars = ["Volvo", "BMW"]; // Allowed
  const cars = ["Volvo", "BMW"]; // Not allowed
  var cars = ["Volvo", "BMW"];   // Not allowed
  cars = ["Volvo", "BMW"];       // Not allowed
}
```

Redeclaring an array with const, in another scope, or in another block, is allowed:

Example
```
const cars = ["Volvo", "BMW"];   // Allowed
{
  const cars = ["Volvo", "BMW"]; // Allowed
}
{
  const cars = ["Volvo", "BMW"]; // Allowed
}
```

Creating Date Objects
Date objects are created with the new Date() constructor.
There are 4 ways to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order).

JavaScript counts months from 0 to 11:
January = 0.
December = 11.

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date());
</script>
</body>
</html>
```

O/P
```
Tue Oct 18 2022 11:19:21 GMT+0530 (India Standard Time)
```

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date(2022, 9, 18, 12, 15, 30, 0));
</script>
```

```
</body>
</html>
```
O/P
```
Tue Oct 18 2022 12:15:30 GMT+0530 (India Standard Time)
```

Specifying a month higher than 11, will not result in an error but add the overflow to the next year.

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date(2018, 15, 24, 10, 33, 30, 0));
</script>
</body>
</html>
```
O/P
```
Wed Apr 24 2019 10:33:30 GMT+0530 (India Standard Time)
```

Specifying a day higher than max, will not result in an error but add the overflow to the next month.

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date(2018, 05, 35, 10, 33, 30, 0));
</script>
</body>
</html>
```
O/P
```
Thu Jul 05 2018 10:33:30 GMT+0530 (India Standard Time)
```

Using 6, 4, 3, or 2 Numbers

6 numbers specify year, month, day, hour, minute, second:

Example
```
const d = new Date(2018, 11, 24, 10, 33, 30);
```
5 numbers specify year, month, day, hour, and minute:

Example
```
const d = new Date(2018, 11, 24, 10, 33);
```
4 numbers specify year, month, day, and hour:

Example
```
const d = new Date(2018, 11, 24, 10);
```
3 numbers specify year, month, and day:

Example
```
const d = new Date(2018, 11, 24);
```
2 numbers specify year and month:

Example

```
const d = new Date(2018, 11);
```
You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

Previous Century
One and two digit years will be interpreted as 19xx.
Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date(85, 5, 20));
</script>
</body>
</html>
```

O/P
```
Thu Jun 20 1985 00:00:00 GMT+0530 (India Standard Time)
```

new Date(dateString)
```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date("October 13, 2014 11:13:00"));
</script>
</body>
</html>
```
O/P
```
Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)
```

JavaScript Date Formats
JavaScript Date Input
There are generally 3 types of JavaScript date input formats:

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

The ISO format follows a strict standard in JavaScript.

The other formats are not so well defined and might be browser specific.

JavaScript Date Output
Independent of input format, JavaScript will (by default) output dates in full text string format:

```
Thu Oct 20 2022 11:54:04 GMT+0530 (India Standard Time)
```

JavaScript ISO Dates

ISO 8601 is the international standard for the representation of dates and times.

The ISO 8601 syntax (YYYY-MM-DD) is also the preferred JavaScript date format:

Example (Complete date)

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date("2015-03-25"));
</script>
</body>
</html>
```

O/P

```
Wed Mar 25 2015 05:30:00 GMT+0530 (India Standard Time)
```

ISO Dates (Year and Month)

ISO dates can be written without specifying the day (YYYY-MM).

Example

```
const d = new Date("2015-03");
```

Time zones will vary the result above between February 28 and March 01.

ISO Dates (Only Year)

ISO dates can be written without month and day (YYYY):

Example

```
const d = new Date("2015");
```

Time zones will vary the result above between December 31 2014 and January 01 2015.

ISO Dates (Date-Time)

ISO dates can be written with added hours, minutes, and seconds (YYYY-MM-DDTHH:MM:SSZ).

Example

```
const d = new Date("2015-03-25T12:00:00Z");
```

Date and time is separated with a capital T.

JavaScript Short Dates.

Short dates are written with an "MM/DD/YYYY" syntax like this:

Example

```
const d = new Date("03/25/2015");
```

JavaScript Long Dates.

Long dates are most often written with a "MMM DD YYYY" syntax like this:

Example

```
const d = new Date("Mar 25 2015");
```

Month and day can be in any order:

Example

```
const d = new Date("25 Mar 2015");
```

And, month can be written in full (January), or abbreviated (Jan):

Example

```
const d = new Date("January 25 2015");
```

Example

```
const d = new Date("Jan 25 2015");
```

Commas are ignored. Names are case insensitive:

Example

```
const d = new Date("JANUARY, 25, 2015");
```

JavaScript Get Date Methods

Method	Description
getFullYear()	Get the year as a four digit number (yyyy)
getMonth()	Get the month as a number (0-11)
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getHours()	Get the hour (0-23)
getMinutes()	Get the minute (0-59)
getSeconds()	Get the second (0-59)
getMilliseconds()	Get the millisecond (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(new Date().getFullYear());
</script>
</body>
</html>
```

O/P

```
2022
```

JavaScript Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

Set Date Methods

Set Date methods are used for setting a part of a date:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(Math.PI);
</script>
</body>
</html>
```

O/P

```
3.141592653589793
```

Number to Integer

There are 4 common methods to round a number to an integer:

Math.round(x)	Returns x rounded to its nearest integer
Math.ceil(x)	Returns x rounded up to its nearest integer
Math.floor(x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns the integer part of x

```
document.write(Math.round(3.69));O/P-4
document.write(Math.ceil(7.69));O/P-8
document.write(Math.floor(7.69));O/P-7
document.write(Math.trunc(81.69));O/P-81


document.write(Math.sign(4.78));O/P-1
document.write(Math.sign(-4.78));O/P- -1
document.write(Math.sign(0));O/P-0

document.write(Math.pow(9,2));  O/P-81
document.write(Math.sqrt(64));  O/P-8

document.write(Math.abs(-41));  O/P-41
document.write(Math.abs(41));  O/P- 41
document.write(Math.abs(0));  O/P-0
```

document.write(Math.min(0, 150, 30, 20, -8, -200)); O/P- -200
document.write(Math.max(0, 150, 30, 20, -8, -200)); O/P- 150

Math.random() returns a random number between 0 and 1.
document.write(Math.random()); O/P- 0.7396551108987746
document.write(Math.random()); O/P- 0.3947634951713139

Math.log() returns the natural logarithm of a number.
document.write(Math.log(0)); O/P - -Infinity
document.write(Math.log(-3));O/P -NaN
document.write(Math.log(3));O/P-1.0986122886681096
document.write(Math.log(5));O/P-1.6094379124341003
document.write(Math.log(10));O/P -2.302585092994046
document.write(Math.log(100));O/P- 4.605170185988092
document.write(Math.log(1000));O/P-6.907755278982137
document.write(Math.log(10000));O/P-9.210340371976184

NaN is short for "Not-a-Number". In JavaScript, NaN is a number that is not a legal number.

The Math.log2() Method
Math.log2(x) returns the base 2 logarithm of x.
document.write(Math.log2(8));O/P-3
document.write(Math.log2(16));O/P-4
document.write(Math.log2(10));O/P-3.321928094887362

The Math.log10() Method
Math.log10(x) returns the base 10 logarithm of x.
document.write(Math.log10(10));O/P-1
document.write(Math.log10(100));O/P-2
document.write(Math.log10(1000));O/P-3

document.write(Math.E);O/P-2.718281828459045

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

Number() method

The Number() method converts a value to a number.
If the value cannot be converted, NaN is returned.
For booleans, Number() returns 0 or 1.
For dates, Number() returns milliseconds since January 1, 1970 00:00:00.

For strings, Number() returns a number or NaN.

Syntax

```
Number(value)
```

If the value cannot be converted to a number, NaN is returned.

If no value is provided, 0 is returned.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write((Number(true))+"<br>");
document.write(Number(false)+"<br>");
document.write(Number("CIME")+"<br>");
document.write(Number(new Date())+"<br>");
document.write((Number(654))+"<br>");
document.write((Number(65.76))+"<br>");
document.write((Number("999"))+"<br>");
document.write((Number("999 675"))+"<br>");
</script>
</body>
</html>
```

O/P

```
1
0
NaN
1666426299307
654
65.76
999
NaN
```

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write((Number(654))+"<br>");
document.write((Number(654.56))+"<br>");
document.write((Number("654"))+"<br>");
document.write((Number(654,65))+"<br>");
</script>
</body>
</html>
```

O/P

```
654
654.56
654
654
```

Convert different arrays to a number.

Ex:

```
        <!DOCTYPE html>
        <html>
        <body>
        <script>
        document.write(Number([9])+"<br>");
        document.write(Number([91.9])+"<br>");
        document.write(Number([9,9])+"<br>");
        </script>
        </body>
        </html>
```

O/P

```
        9
        91.9
        NaN
```

JavaScript Booleans

A JavaScript Boolean represents one of two values: true or false.

Boolean Values

Very often, in programming, you will need a data type that can only have one of two values, like

YES / NO
ON / OFF
TRUE / FALSE

For this, JavaScript has a Boolean data type. It can only take the values true or false.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write((10 > 9)+"<br>");
document.write(Boolean(10 < 9)+"<br>");
document.write(Boolean(10==9)+"<br>");
document.write(Boolean(10>9)+"<br>");
</script>
</body>
</html>
```

O/P

```
true
false
false
true
```

JavaScript Events

HTML events are "things" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "react" on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:
An HTML web page has finished loading
An HTML input field was changed
An HTML button was clicked
Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:
```
<element event='some JavaScript'>
```
With double quotes:
```
<element event="some JavaScript">
```

Ex
```
<!DOCTYPE html>
<html>
<body>
<button onclick="document.write(Date())">Date</button>
</body>
</html>
```
O/P
```
Wed Oct 26 2022 11:10:50 GMT+0530 (India Standard Time)
```
Ex
```
<!DOCTYPE html>
<html>
<body>
<button onclick="displayDate()">Date</button>
<script>
function displayDate() {
  document.write(Date());
}
</script>
</body>
</html>
```
O/P
```
Wed Oct 26 2022 11:12:32 GMT+0530 (India Standard Time)
```

Common HTML Events
Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Event Handlers
Event handlers can be used to handle and verify user input, user actions, and browser actions:

Things that should be done every time a page loads
Things that should be done when the page is closed
Action that should be performed when a user clicks a button
Content that should be verified when a user inputs data

Many different methods can be used to let JavaScript work with events:
HTML event attributes can execute JavaScript code directly
HTML event attributes can call JavaScript functions
You can assign your own event handler functions to HTML elements
You can prevent events from being sent or being handled

JavaScript Objects
Objects are one of JavaScript's data types.
Objects are used to store key/value (name/value) collections.
A JavaScript object is a collection of named values.
Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

document.write(person.firstName + "<br>" );
document.write(person.lastName + "<br>" );
document.write(person.age + "<br>" );
document.write(person.eyeColor + "<br>");
</script>
</body>
</html>
```

O/P

```
John
```

```
Doe
50
blue
```

JavaScript Object Methods and Properties

Name	Description
constructor	Returns the function that created an object's prototype
keys()	Returns an Array Iterator object with the keys of an object
prototype	Let you to add properties and methods to JavaScript objects
toString()	Converts an object to a string and returns the result
valueOf()	Returns the primitive value of an object

JavaScript Object.keys()
Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = Object.keys(fruits);
document.write(keys);
</script>
</body>
</html>
```
O/P
```
0,1,2,3
```

```
const fruits ="Banana Mango";
```
O/P 0,1,2,3,4,5,6,7,8,9,10,11

```
const fruits ="Banana";
```
O/P 0,1,2,3,4,5

JavaScript Object.keys()
The Object.keys() method returns an Array Iterator object with the keys of an object.
The Object.keys() method does not change the original object.
Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
keys = Object.keys(person);
document.write(keys);
```

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

O/P

```
firstName,lastName,age,eyeColor
```

JavaScript Object prototype

Example

Use the prototype property to add a new property to all objects of a given type.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
function employee(name, jobtitle, born) {
  this.name = name;
  this.jobtitle = jobtitle;
  this.born = born;
}
employee.prototype.salary = 2000;
const fred = new employee("Fred Flintstone", "Caveman", 1970);
document.write(fred.salary);
</script>
</body>
</html>
```

O/P

```
2000
```

JavaScript Object toString()

Using toString() on an array.

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
myArray = fruits.toString();
document.write(myArray);
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
```

JavaScript Object valueOf()

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const myArray = fruits.valueOf();
document.write(myArray);
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
```

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
fruits = ["Banana", "Orange", "Apple", "Mango"];
myArray = fruits;
document.write(myArray);
</script>
</body>
</html>
```

O/P

```
Banana,Orange,Apple,Mango
```

JavaScript Strings
A JavaScript string stores a series of characters .
A string can be any text inside double or single quotes.
String indexes are zero-based:
The first character is in position 0, the second in 1, and so on.
Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
let carName1 = "Volvo XC60";  // Double quotes
let carName2 = 'Volvo XC60';  // Single quotes
document.write((carName1) + "<br>");
document.write((carName1) + (carName2));
</script>
</body>
</html>
```

O/P

```
Volvo XC60
Volvo XC60Volvo XC60
```

String Properties and Methods

Normally, strings like "John Doe", cannot have methods or properties because they are not objects.

But with JavaScript, methods and properties are also available to strings, because JavaScript treats strings as objects when executing methods and properties.

JavaScript String Methods

Name	Description
charAt()	Returns the character at a specified index (position)
charCodeAt()	Returns the Unicode of the character at a specified index
concat()	Returns two or more joined strings
constructor	Returns the string's constructor function
endsWith()	Returns if a string ends with a specified value
fromCharCode()	Returns Unicode values as characters
includes()	Returns if a string contains a specified value
indexOf()	Returns the index (position) of the first occurrence of a value in a string
lastIndexOf()	Returns the index (position) of the last occurrence of a value in a string
length	Returns the length of a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a value, or a regular expression, and returns the matches
prototype	Allows you to add properties and methods to an object
repeat()	Returns a new string with a number of copies of a string
replace()	Searches a string for a value, or a regular expression, and returns a string where the values are replaced
search()	Searches a string for a value, or regular expression, and returns the index (position) of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts a number of characters from a string, from a start index (position)
substring()	Extracts characters from a string, between two specified indices (positions)
toLocaleLowerCase()	Returns a string converted to lowercase letters, using the host's locale
toLocaleUpperCase()	Returns a string converted to uppercase letters, using the host's locale

toLowerCase()	Returns a string converted to lowercase letters
toString()	Returns a string or a string object as a string
toUpperCase()	Returns a string converted to uppercase letters
trim()	Returns a string with removed whitespaces
trimEnd()	Returns a string with removed whitespaces from the end
trimStart()	Returns a string with removed whitespaces from the start
valueOf()	Returns the primitive value of a string or a string object

Note
All string methods return a new value.
They do not change the original variable.

String HTML Wrapper Methods
HTML wrapper methods return a string wrapped inside an HTML tag.
These are not standard methods, and may not work as expected.

Method	Description
anchor()	Displays a string as an anchor
big()	Displays a string using a big font
blink()	Displays a blinking string
bold()	Displays a string in bold
fixed()	Displays a string using a fixed-pitch font
fontcolor()	Displays a string using a specified color
fontsize()	Displays a string using a specified size
italics()	Displays a string in italic
link()	Displays a string as a hyperlink
small()	Displays a string using a small font
strike()	Displays a string with a strikethrough
sub()	Displays a string as subscript text
sup()	Displays a string as superscript text

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
text1="CIME";
text2="BHUBANESWAR";
document.write(text1.concat(text2));
</script>
</body>
</html>
```

O/P
```
CIMEBHUBANESWAR
```

Ex
```
<!DOCTYPE html>
<html>
<body>
```

```
<script>
text1="CIME";
text2="BHUBANESWAR";
document.write(text1.concat(" ",text2));
</script>
</body>
</html>
```

O/P

```
CIME BHUBANESWAR
```

JavaScript String slice()

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
text = "Odisha";
document.write(text.slice(3));
</script>
</body>
</html>
```

O/P

```
sha
```

The first position is 0, the second is 1, ...
A negative number selects from the end of the string.

```
document.write(text.slice(-2));          O/P ha
document.write(text.slice(-1));          O/P a
```

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
let text = "     Hello World!     ";
let result = text.trim();
document.write(result);
</script>
</body>
</html>
```

O/P

```
Hello World!
```

Ex

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
text = "Hello World!";
document.write(text.bold());
</script>
</body>
</html>
```
O/P

Hello World!

```
document.write(text.italics());
```
O/P

Hello World!

```
document.write(text.sub());
```
O/P

Hello World!$_{\text{Hello World!}}$

JavaScript RegExp Reference

RegExp Object

A regular expression is a pattern of characters.
The pattern is used to do pattern-matching "search-and-replace" functions on text.
In JavaScript, a RegExp Object is a pattern with Properties and Methods.

Ex
```
<!DOCTYPE html>
<html>
<body>
<script>
let text = "WELCOME TO CIME";
//let pattern = /to cime/i;
let pattern = "CIME";
let result = text.match(pattern);
document.write(result);
</script>
</body>
</html>
```
O/P
```
CIME
```

to cime	The pattern to search for
/to cime/	A regular expression
/to cime/i	A case-insensitive regular expression

Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)

i	Perform case-insensitive matching
m	Perform multiline matching

Brackets

Brackets are used to find a range of characters:

Expression	**Description**
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	**Description**
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NULL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd

Quantifiers

Quantifier	**Description**
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X to Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

RegExp Object Properties

Property	**Description**

constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set
source	Returns the text of the RegExp pattern

RegExp Object Methods

Method	Description
compile()	Deprecated in version 1.5. Compiles a regular expression
exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false
toString()	Returns the string value of the regular expression

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<h2>JavaScript Validation</h2>
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Validation</h2>
<p>Please input a number between 1 and 10:</p>
<input id="numb">
```

```
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```