

## **UNIT II CLIENT SIDE PROGRAMMING**

### CLIENT SIDE PROGRAMMING

#### JAVASCRIPT:

- Inserting JavaScript into a webpage is much like inserting any other HTML content. The tags used to add JavaScript in HTML are `<script>` and `</script>`.
- The code surrounded by the `<script>` and `</script>` tags is called a script block.
- 'type' attribute was the most important attribute of `<script>` tag. However, it is no longer used. Browser understands that `<script>` tag has JavaScript code inside it.
- It is a client-side scripting language.
- JavaScript is used in Web site development to such things as:
  - check or modify the contents of forms
  - change images
  - open new windows and write dynamic page content.
- `<script></script>`

#### **Java Script Vs Java**

JavaScript	Java
Interpreted (not compiled) by client.	Compiled on server before execution on client.
Object-based. Code uses built-in, extensible objects, but no classes or inheritance.	Object-oriented. Applets consist of object classes with inheritance.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
Secure. Cannot write to hard disk.	Secure. Cannot write to hard disk.

#### FEATURES-JAVA SCRIPT

- Browser support
  - Syntax
  - Dynamic typing
- Java Script can enhance the dynamics and interactive features of your page by allowing you to perform calculations, check forms, write interactive games, add special effects, customize graphics selections, create security passwords and more.
- Run time evaluation
  - Support for object
  - Regular expression
  - Functions in program

#### DIRECTLY EMBEDDED JAVASCRIPT

Javascript directly embedded within HTML document

- Inside the head tag
- Within the body tag

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

...  
...  
...

```
</script>
```

### Inside HEAD Tag:

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

### Within BODY Tag:

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

### Program:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Internal Script</title>
    <script type = "text/JavaScript">
      function Hello()
      {
        alert("Hello, World");
      }
    </script>
  </head>
  <body>
    <input type = "button" onclick = "Hello();" name = "ok" value = "Click Me" />
  </body>
</html>
```

### INDIRECTLY EMBEDDED JAVASCRIPT

- If you are going to define a functionality which will be used in various HTML documents then it's better to keep that functionality in a separate JavaScript file and then include that file in your HTML documents. A JavaScript file will have extension as .js and it will be included in HTML files using <script> tag.

#### Advantage

Script hidden from browser user

#### Disadvantage

Small script-separate file is meaningless

Complicated

### FirstPg.htmlLive Demo

```
<!DOCTYPE html>
<html>
  <head>
    <title>Javascript External Script</title>
    <script src = " my_script.js" type = "text/javascript"/></script>
```

```

</head>
<body>
  <input type = "button" onclick = "Hello();" name = "ok" value = "Click Me" />
</body>
</html>

```

```

my_script.js
function Hello()
{
  alert("Hello, World");
}

```

---

## **Datatypes in JavaScript**

There are majorly two types of languages. First, one is **Statically typed language** where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types. Example: C, C++, Java.

```

// Java(Statically typed)
int x = 5 // variable x is of type int and it will not store any other type.
string y = 'abc' // type string and will only accept string values

```

Other, **Dynamically typed languages**: These languages can receive different data types over time. For example- Ruby, Python, JavaScript etc.

```

// Javascript(Dynamically typed)
var x = 5; // can store an integer
var name = 'string'; // can also store a string.

```

JavaScript is dynamically typed (also called loosely typed) scripting language. That is, in javascript variables can receive different data types over time. Datatypes are basically typed of data that can be used and manipulated in a program.

### **seven data types:**

Out of which six data types are Primitive(predefined).

- **Numbers**: 5, 6.5, 7 etc.
- **String**: "Hello Guys" etc.
- **Boolean**: Represent a logical entity and can have two values: true or false.
- **Null**: This type has only one value : *null*.
- **Undefined**: A variable that has not been assigned a value is *undefined*.
- **Object**: It is the most important data-type and forms the building blocks for modern JavaScript. We will learn about these data types in details in further articles.

---

## **Variables in JavaScript:**

Variables in JavaScript are containers which hold reusable data. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In JavaScript, all the variables must be declared before they can be used.

It is declared using the *var* keyword followed by the name of the variable and semi-colon. Below is the syntax to create variables in JavaScript:

```

var var_name;
var x;

```

The *var\_name* is the name of the variable which should be defined by the user and should be unique. These type of names are also known as **identifiers**. The rules for creating an identifier in JavaScript are, the name of the identifier should not be any pre-defined word(known as keywords), the first character must be a letter, an underscore (\_), or a dollar sign (\$). Subsequent characters may be any letter or digit or an underscore or dollar sign.

## Variable Scope

Scope of a variable is the part of the program from where the variable may directly be accessible. In JavaScript, there are two types of scopes:

1. **Global Scope** – Scope outside the outermost function attached to Window.
2. **Local Scope** – Inside the function being executed.

```
var globalVar = "This is a global variable";
function fun()
{
var localVar = "This is a local variable";
document.out(globalVar);
document.out (localVar);
}
fun();
```

### output:

This is a global variable  
This is a local variable

---

## concepts of JSON function, JSON array with example.

**JSON** or **JavaScript Object Notation** is a format for structuring data. Like XML, it is one of the way of formatting the data. Such format of data is used by web applications to communicate with each other.

### Characteristics of JSON

- JSON stands for JavaScript Object Notation.
- It is a lightweight text-based interchange format.
- JSON is language independent.
- JSON is easy to read and write than XML.
- JSON supports array, object, string, number and values.
- It has been extended from the JavaScript scripting language.

The JSON file must be save with .json extension.

### JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).

Square brackets hold arrays and values are separated by ,(comma).

#### **Example:**

```
{ "name": "Sonoo" }
```

- Data is separated by commas

#### **Example:**

```
{ "name": "Sonoo", "email": "sonoojaiswal1987@gmail.com" }
```

- Curly braces hold objects

#### **Example:**

```
var person={ "name": "Sonoo", "email": "sonoojaiswal1987@gmail.com" }
```

Here person is the object.

Square brackets hold arrays

#### **Example1:**

```
var person= { "name": "radha", "Occupation": "private", "arrange": [ "a1", "a2", "a3" ] }
```

arrange is an array. person is a object

### Example2:

```
{ "employees": [
  { "name": "Sonoo", "email": "sonoojaiswal1987@gmail.com" },
  { "name": "Rahul", "email": "rahul32@gmail.com" },
  { "name": "John", "email": "john32bob@gmail.com" }
]}
```

## Creating Simple Objects

JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript

- Creation of an empty Object

```
var JSONObj = {};
```

- Creation of a new Object

```
var JSONObj = new Object();
```

- Creation of an object with attribute bookname with value in string, attribute price with numeric value. Attribute is accessed by using '.' Operator

```
var JSONObj = { "bookname": "VB BLACK BOOK", "price": 500 };
```

This is an example that shows creation of an object in javascript using JSON, save the below code as json\_object.htm

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language = "javascript" >
  var JSONObj = { "name" : "tutorialspoint.com", "year" : 2005 };
  document.write("<h1>JSON with JavaScript example</h1>");
  document.write("<br>");
  document.write("<h3>Website Name = "+JSONObj.name+"</h3>");
  document.write("<h3>Year = "+JSONObj.year+"</h3>");
</script>
</head>
<body>
</body>
</html>
```

It produces the following result

### JSON with JavaScript example

Website Name=tutorialspoint.com

Year=2005

## Creating Array Objects

The following example shows creation of an array object in javascript using JSON, save the below code as json\_array\_object.html

```
<html>
<head>
<title>Creation of array object in javascript using JSON</title>
<script language = "javascript" >
  document.write("<h2>Mammals cost</h2>");
  var ani = { "mammals": [
    { "Name" : "goat", "price" : 700 },
    { "Name" : "cow", "price" : 1400 }] }
  var i = 0
```

```

document.write("<table border = '2'><tr>");
for(i = 0;i<ani.mammals.length;i++) {
    document.write("<td>");
    document.write("<table border = '1' width = 100 >");
    document.write("<tr><td><b>Name</b></td><td width = 50>" + ani.mammals[i].Name+"</td></tr>");
    document.write("<tr><td><b>Price</b></td><td width = 50>" + ani.mammals[i].price + "</td></tr>");
    document.write("</table>");
    document.write("</td>");
}
    document.writeln("</tr></table>");
</script>
</head>
<body>
</body>
</html>
Output:

```

## Mammals cost

<b>Name</b>	goat	<b>Name</b>	cow
<b>Price</b>	700	<b>Price</b>	1400

## JSON Function Files

- A common use of JSON is to read data from a web server, and display the data in a web page.
- We have written the student function in the file named fun.js
- Function defines an array of two fields “name” and “year”.there are three elements in the array.
- In our HTML file “function-array.html” , the JSON code refers to the external js file as shown below.  

```
<script src="fun.js"></script>
```
- Then the elements of an array are accessed and displayed on the browser

### function-array.html

```

<!DOCTYPE html>
<html>
<head>
    <title> JSON function</title>
    <script language = "javascript" >
        document.write("<h3>JSON function file</h3>");
        function student(arr)
        {
            for(var i=0;i<arr.length;i++)
            {
                document.write("<h3>Website Name = "+arr[i].name+"</h3>");
                document.write("<h3>Year = "+arr[i].year+"</h3>");
            }
            document.write("<br/>");
        }
    </script>
</head>
<body>
    <script src="fun.js"></script>
</body>
</html>

```

### fun.js

```
student([ { "name" : "tutorialspoint.com", "year" : 2005 }, { "name" : "aaa.com", "year" : 2010 }, { "name" : "bbb.com", "year" : 2015 }])
```

**output:**

**JSON function file**

**Website Name = tutorialspoint.com**

**Year = 2005**

**Website Name = aaa.com**

**Year = 2010**

**Website Name = bbb.com**

**Year = 2015**

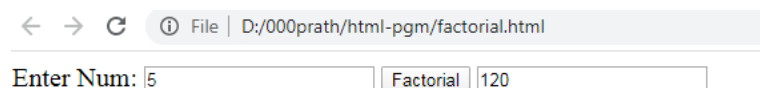
---

**Write a javascript to find factorial of a given number.**

**Program:**

```
<!doctype html>
<html>
<head>
    <script>
    function show()
    {
    var i, no, fact;
    fact=1;
    no=Number(document.getElementById("num").value);
    for(i=1; i<=no; i++)
    {
    fact= fact*i;
    }
    document.getElementById("answer").value= fact;
    }
    </script>
</head>
<body>
    Enter Num:
    <input id="num">
    <button onclick="show()">Factorial</button>
    <input id="answer">
</body>
</html>
```

**Output:**

A screenshot of a web browser window. The address bar shows the file path "D:/000prath/html-pgm/factorial.html". The page content displays "Enter Num:" followed by an input field containing the number "5". To the right of the input field is a button labeled "Factorial". Further right is another input field containing the number "120".

← → ↻ ① File | D:/000prath/html-pgm/factorial.html

Enter Num:

**Code Explanation**

- **no=Number(document.getElementById("num").value);**  
This code is used for receive input value form input field which have id num.
- **document.getElementById("answer").value= fact;**  
This code is used for receive calculated value of factorial and display in input field which have id answer
- **<button onclick="show()">Factorial</button>**  
This code is used for call show function when button clicked.



---

**Write a HTML program for the registration of new customer to online banking system (customer data collected using a form, after submitting account number and type of account and username, password is displayed as output)**

**Program:**

```
<html>
<head>
<title>
    Online Banking System
</title>
<script type="text/javascript">
    function fun()
    {
        alert("Name:"+form1.username.value);
        alert("Password:"+form1.pwd.value);
        alert("Password:"+form1.mymenu.value);
        alert("Password:"+form1.accno.value);
    }
</script>
</head>
    <body bgcolor="pink">
        <form name="form1">
<table>
    <tr>
        <td><b>Name:</b></td>
        <td><input type="text" name="username"></td>
    </tr>
    <tr>
        <td><b>Password:</b></td>
        <td><input type="password" name="pwd"></td>
    </tr>
    <tr>
        <td><b>Account Number:</b></td>
        <td>
            <select name="mymenu">
                <option value="saving account">Saving Account</option>
                <option value="current account">Current Account</option>
                <option value="zero account">Zero Account</option>
            </select>
        </td>
    </tr>
    <tr>
        <td><b>Account Number:</b></td>
        <td><input type="number" name="accno"></td>
    </tr>
</table>
<br/>
<br/>
<center>
    <input type="submit" value="submit" onclick="fun()">
    <input type="reset" value="Clear">
</center>
</form>
</body>
</html>
```

**Output:**

**Name:**   
**Password:**   
**Account Number:**   
**Account Number:**

This page says  
 Name:Pratheeba. R.S

---

### Write a javascript to find the prime number between 1 and 100

**Output:**

**PRIME NUMBERS BETWEEN 1 AND 100**

1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

**Program:**

```

<html>
<head>
    <title>JavaScript Prime</title>
</head>
<body bgcolor="lightgreen">
<center><h4>PRIME NUMBERS BETWEEN 1 AND 100</h4></center><br/>
<script>
    for (var limit = 1; limit <= 100; limit++)
    {
        var a = false;
        for (var i = 2; i <= limit; i++)
        {
            if (limit%i===0 && i!==limit)
            {
                a = true;
            }
        }
        if (a === false)
        {
            document.write("\t"+limit);
        }
    }
</script>
</body>
</html>

```

---

### Write a XHTML program to display data in table

		Meals		
		Breakfast	Lunch	Dinner
Foods	Bread	100gms	200gms	175gms
	Main Course	100gms	200gms	175gms
	Vegetable	100gms	200gms	175gms
	Dessert	100gms	200gms	175gms

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>CSE Program</title>
</head> <body bgcolor="powderblue">

<table border="4" bordercolor="purple" >
<tr> <td rowspan="2"></td>
<th></th>
<th colspan="3">Meals</th>
</tr>
<tr>
<th></th>
<th>Breakfast</th>
<th>Lunch</th>
<th>Dinner</th>

</tr>
<tr>
<th rowspan="5">Foods</th>
<th>Bread</th>
<td>100gms</td>
<td>200gms</td>
<td>175gms</td>

</tr> <tr>
<th>Main Course</th>
<td>100gms</td>
<td>200gms</td>
<td>175gms</td>

</tr> <tr>
<th>Vegetable</th>
<td>100gms</td>
<td>200gms</td>
<td>175gms</td>

</tr> <tr>
<th>Dessert</th>
<td>100gms</td>
<td>200gms</td>
<td>175gms</td>

</tr>
</table>

```

</body>  
</html>

---

## **Regular expression**

### **Regular Expression**

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects.

These patterns are used with the

- `exec`
- test methods

With the `match`, `replace`, `search`, and `split` methods of `String`.

### **Creating a regular expression**

You construct a regular expression in one of two ways:

Using a regular expression literal, as follows:

```
var re = /ab+c/;
```

Regular expression literals provide compilation of the regular expression when the script is loaded.

When the regular expression will remain constant, use this for better performance. Or calling the constructor function of the

`RegExp` object, as follows:

```
var re = new RegExp("ab+c");
```

Using the constructor function provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.

### **Writing a regular expression pattern**

A regular expression pattern is composed of simple characters, such as `/abc/`, or a combination of simple and special characters, such as `/ab*c/` or `/Chapter (\d+)\. \d*/`.

The last example includes parentheses which are used as a memory device.

The match made with this part of the pattern is remembered for later use, as described in [Using parenthesized substring matches](#).

### **Using simple patterns**

Simple patterns are constructed of characters for which you want to find a direct match.

For example, the pattern `/abc/` matches character combinations in strings only when exactly the characters 'abc' occur together and in that order.

Such a match would succeed in the strings "Hi, do you know your abc's?" and "The latest airplane designs evolved from slabcraft." In both cases the match is with the substring 'abc'.

There is no match in the string 'Grab crab' because while it contains the substring 'ab c', it does not contain the exact substring 'abc'.

### **Using special characters**

When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, the pattern includes special characters.

For example, the pattern `/ab*c/` matches any character combination in which a single 'a' is followed by zero or more 'b's (\* means 0 or more occurrences of the preceding item) and then immediately followed by 'c'.

In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'.

The following table provides a complete list and description of the special characters that can be used in

regular expressions.

Special characters in regular expressions.	
Character	Meaning
\	<p>Matches according to the following rules:</p> <p>A backslash that precedes a non-special character indicates that the next character is special and is not to be interpreted literally. For example, a 'b' without a preceding '\' generally matches lowercase 'b's wherever they occur. But a '\b' by itself doesn't match any character; it forms the special word boundary character.</p> <p>A backslash that precedes a special character indicates that the next character is not special and should be interpreted literally. For example, the pattern /a*/ relies on the special character '*' to match 0 or more a's. By contrast, the pattern /a\*/ removes the specialness of the '*' to enable matches with strings like 'a*'. </p> <p>Do not forget to escape \ itself while using the RegExp("pattern") notation because \ is also an escape character in strings.</p>
^	<p>Matches beginning of input. If the multiline flag is set to true, also matches immediately after a line break character.</p> <p>For example, /^A/ does not match the 'A' in "an A", but does match the 'A' in "An E".</p> <p>The '^' has a different meaning when it appears as the first character in a character set pattern. See complemented character sets for details and an example.</p>
\$	<p>Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character.</p> <p>For example, /t\$/ does not match the 't' in "eater", but does match it in "eat".</p>
*	Matches the preceding character 0 or more times. Equivalent to {0,}.
	For example, /bo*/ matches 'boooo' in "A ghost boooed" and 'b' in "A bird warbled", but nothing in "A goat grunted".
+	Matches the preceding character 1 or more times. Equivalent to {1,}.
	For example, /a+/ matches the 'a' in "candy" and all the a's in "caaaaaaandy", but nothing in "cndy".
?	Matches the preceding character 0 or 1 time. Equivalent to {0,1}.
	<p>For example, /e?le?/ matches the 'el' in "angel" and the 'le' in "angle" and also the 'l' in "oslo".</p> <p>If used immediately after any of the quantifiers *, +, ?, or {}, makes the quantifier non-greedy (matching the fewest possible characters), as opposed to the default, which is greedy (matching as many characters as possible). For example, applying /\d+/ to "123abc" matches "123". But applying /\d+?/ to that same string matches only the "1".</p> <p>Also used in lookahead assertions, as described in the x(?=y) and x(!y) entries of this table.</p>

.	<p>(The decimal point) matches any single character except the newline character.</p> <p>For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.</p>
(x)	<p>Matches 'x' and remembers the match, as the following example shows. The parentheses are called capturing parentheses.</p> <p>The '(foo)' and '(bar)' in the pattern /(foo) (bar) \1 \2/ match and remember the first two words in the string "foo bar foo bar". The \1 and \2 in the pattern match the string's last two words. Note that \1, \2, \n are used in the matching part of the regex. In the replacement part of a regex the syntax \$1, \$2, \$n must be used, e.g.: 'bar foo'.replace( /(...) (...)/, '\$2 \$1' ).</p>
(?:x)	<p>Matches 'x' but does not remember the match. The parentheses are called non- capturing parentheses, and let you define subexpressions for regular expression operators to work with. Consider the sample expression /(?:foo){1,2}/. If the expression was /foo{1,2}/, the {1,2} characters would apply only to the last 'o' in 'foo'. With the non-capturing parentheses, the {1,2} applies to the entire word 'foo'.</p>
<b>Special characters in regular expressions</b>	
<b>Character</b>	<b>Meaning</b>
x(?=y)	<p>Matches 'x' only if 'x' is followed by 'y'. This is called a lookahead.</p> <p>For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat Frost)/ matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results.</p>
x(?!y)	<p>Matches 'x' only if 'x' is not followed by 'y'. This is called a negated lookahead.</p> <p>For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'.</p>
x y	<p>Matches either 'x' or 'y'.</p> <p>For example, /green red/ matches 'green' in "green apple" and 'red' in "red apple."</p>
{n}	<p>Matches exactly n occurrences of the preceding character. N must be a positive integer.</p> <p>For example, /a{2}/ doesn't match the 'a' in "candy," but it does match all of the a's in "caandy," and the first two a's in "caaandy."</p>
{n,m}	<p>Where n and m are positive integers and n &lt;= m. Matches at least n and at most m occurrences of the preceding character. When m is omitted, it's treated as ∞.</p> <p>For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy". Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.</p>

[xyz]	<p>Character set. This pattern type matches any one of the characters in the brackets, including escape sequences. Special characters like the dot(.) and asterisk (*) are not special inside a character set, so they don't need to be escaped. You can specify a range of characters by using a hyphen, as the following examples illustrate.</p> <p>The pattern [a-d], which performs the same match as [abcd], matches the 'b' in "brisket" and the 'c' in "city". The patterns /[a-z.]+/ and /[\\w.]+/ match the entire string "test.i.ng".</p>
[^xyz]	<p>A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. Everything that works in the normal character set also works here.</p> <p>For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop."</p>
[\\b]	<p>Matches a backspace (U+0008). You need to use square brackets if you want to match a literal backspace character. (Not to be confused with \\b.)</p>
\\b	<p>Matches a word boundary. A word boundary matches the position where a word character is not followed or preceded by another word-character. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero. (Not to be confused with [\\b].)</p> <p>Examples:</p> <p>\\bm/ matches the 'm' in "moon" ;</p> <p>/oo\\b/ does not match the 'oo' in "moon", because 'oo' is followed by 'n' which is a word character;</p> <p>/oon\\b/ matches the 'oon' in "moon", because 'oon' is the end of the string, thus not followed by a word character;</p> <p>/\\w\\b\\w/ will never match anything, because a word character can never be followed by both a non-word and a word character.</p> <p>Note: JavaScript's regular expression engine defines a specific set of characters to be "word" characters. Any character not in that set is considered a word break. This set of characters is fairly limited: it consists solely of the Roman alphabet in both upper- and lower-case, decimal digits, and the underscore character. Accented characters, such as "é" or "ü" are, unfortunately, treated as word breaks.</p>
\\B	<p>Matches a non-word boundary. This matches a position where the previous and next character are of the same type: Either both must be words, or both must be non- words. The beginning and end of a string are considered non-words.</p> <p>For example, \\B./ matches 'oo' in "noonday", and /y\\B./ matches 'ye' in "possibly yesterday."</p>
\\cX	<p>Where X is a character ranging from A to Z. Matches a control character in a string.</p> <p>For example, \\cM/ matches control-M (U+000D) in a string.</p>
\\d	<p>Matches a digit character. Equivalent to [0-9].</p> <p>For example, \\d/ or /[0-9]/ matches '2' in "B2 is the suite number."</p>
\\D	<p>Matches any non-digit character. Equivalent to [^0-9].</p> <p>For example, \\D/ or /^[^0-9]/ matches 'B' in "B2 is the suite number."</p>

\f	Matches a form feed (U+000C).
\n	Matches a line feed (U+000A).
\r	Matches a carriage return (U+000D).
\s	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to [ \f\n\r\t\v\u00a0\u1680\u180e\u2000-\u200a\u2028\u2029\u202f\u205f\u3000].  For example, /\s\w*/ matches ' bar' in "foo bar."
\S	Matches a single character other than white space. Equivalent to [^\f\n\r\t\v\u00a0\u1680\u180e\u2000-\u200a\u2028\u2029\u202f\u205f\u3000].  For example, /\S\w*/ matches 'foo' in "foo bar."
\t	Matches a tab (U+0009).
\v	Matches a vertical tab (U+000B).
\w	Matches any alphanumeric character including the underscore. Equivalent to [A-Za-z0-9_].  For example, /\w/ matches 'a' in "apple," '5' in "\$5.28," and '3' in "3D."
\W	Matches any non-word character. Equivalent to [^A-Za-z0-9_].  For example, /\W/ or /[^\w]/ matches '%' in "50%."
\n	Where n is a positive integer, a back reference to the last substring matching the n parenthetical in the regular expression (counting left parentheses).  For example, /apple(,)\sorange\1/ matches 'apple, orange,' in "apple, orange, cherry, peach."
\0	Matches a NULL (U+0000) character. Do not follow this with another digit, because \0<digits> is an octal escape sequence.
\xhh	Matches the character with the code hh (two hexadecimal digits)
\uhhhh	Matches the character with the code hhhh (four hexadecimal digits).

Escaping user input to be treated as a literal string within a regular expression can be accomplished by simple replacement:

```
function escapeRegExp(string){
    return string.replace(/[.*+?^${}()|[\]\\]/g, "\\$&");
}
```

### Using parentheses

Parentheses around any part of the regular expression pattern cause that part of the matched substring to be remembered. Once remembered, the substring can be recalled for other use, as described in Using Parenthesized Substring Matches.

For example,

The pattern **/Chapter (\d+)\.d\*/** illustrates additional escaped and special characters and indicates that part of the pattern should be remembered.

It matches precisely the characters 'Chapter ' followed by one or more numeric characters (\d means any numeric character and + means 1 or more times), followed by a decimal point (which in itself is a special character; preceding the decimal point with \ means the pattern must look for the literal character '.'), followed by any numeric character 0 or more times (\d means numeric character, \* means 0 or more times). In addition, parentheses are used to remember the first matched numeric characters.

This pattern is found in "Open Chapter 4.3, paragraph 6" and '4' is remembered. The pattern is not found in "Chapter 3 and 4", because that string does not have a period after the '3'.

To match a substring without causing the matched part to be remembered, within the parentheses preface the pattern with ?. For example, (?.d+) matches one or more numeric characters but does not remember the matched characters.



## Working with regular expressions

Regular expressions are used with the RegExp methods `test` and `exec` and with the String methods `match`, `replace`, `search`, and `split`. These methods are explained in detail in the JavaScript reference.

Methods that use regular expressions	
Method	Description
<code>exec</code>	A RegExp method that executes a search for a match in a string. It returns an array of information.
<code>test</code>	A RegExp method that tests for a match in a string. It returns true or false.
<code>match</code>	A String method that executes a search for a match in a string. It returns an array of information or null on a mismatch.
<code>search</code>	A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
<code>replace</code>	A String method that executes a search for a match in a string, and replaces the matched substring with a replacement substring.
<code>split</code>	A String method that uses a regular expression or a fixed string to break a string into an array of substrings.

When you want to know whether a pattern is found in a string, use the **test** or **search** method; for more information (but slower execution) use the **exec** or **match** methods.

If you use `exec` or `match` and if the match succeeds, these methods return an array and update properties of the associated regular expression object and also of the predefined regular expression object.

If the match fails, the *exec* method returns null

In the following example, the script uses the `exec` method to find a match in a string.

```
var myRe = /d(b+)d/g;
var myArray = myRe.exec("cdbbdsbz");
```

If you do not need to access the properties of the regular expression, an alternative way of creating `myArray` is with this script:

```
var myArray = /d(b+)d/g.exec("cdbbdsbz");
```

If you want to construct the regular expression from a string, yet another alternative is this script:

```
var myRe = new RegExp("d(b+)d", "g");
var myArray = myRe.exec("cdbbdsbz");
```

With these scripts, the match succeeds and returns the array and updates the properties shown in the following table.

Object	Property or index	Description	In this example
myArray		The matched string and all remembered substrings.	["dbbd", "bb"]
	index	The 0-based index of the match in the input string.	1
	input	The original string.	"cdbbdsbz"
	[0]	The last matched characters.	"dbbd"

myRe	lastIndex	The index at which to start the next match. (This property is set only if the regular expression uses the g option, described in Advanced Searching With Flags.)	5
	source	The text of the pattern. Updated at the time that the regular expression is created, not executed.	

We can use a regular expression created with an object initializer without assigning it to a variable. If you do, however, every occurrence is a new regular expression. For this reason, if you use this form without assigning it to a variable, you cannot subsequently access the properties of that regular expression. For example, assume you have this script:

```
var myRe = /d(b+)d/g;
var myArray = myRe.exec("cdbbdsbz");
console.log("The value of lastIndex is " + myRe.lastIndex);

// "The value of lastIndex is 5"
```

However, if you have this script:

```
var myArray = /d(b+)d/g.exec("cdbbdsbz");
console.log("The value of lastIndex is " + /d(b+)d/g.lastIndex);

// "The value of lastIndex is 0"
```

The occurrences of /d(b+)d/g in the two statements are different regular expression objects and hence have different values for their lastIndex property. If you need to access the properties of a regular expression created with an object initializer, you should first assign it to a variable.

### Using parenthesized substring matches

Including parentheses in a regular expression pattern causes the corresponding submatch to be remembered. For example, /a(b)c/ matches the characters 'abc' and remembers 'b'. To recall these parenthesized substring matches, use the Array elements [1], ..., [n].

The number of possible parenthesized substrings is unlimited. The returned array holds all that were found. The following examples illustrate how to use parenthesized substring matches.

The following script uses the replace() method to switch the words in the string. For the replacement text, the script uses the \$1 and \$2 in the replacement to denote the first and second parenthesized substring matches.

```
var re = /(w+)\s(w+)/; var str = "John Smith";
var newstr = str.replace(re, "$2, $1");
console.log(newstr);
```

This prints "Smith, John".

Regular expression flags	
Flag	Description
g	Global search.
i	Case-insensitive search.
m	Multi-line search.
y	Perform a "sticky" search that matches starting at the current position in the target string.

To include a flag with the regular expression, use this **syntax**:

```
var re = /pattern/flags; or  
  
var re = new RegExp("pattern", "flags");
```

Note that the flags are an integral part of a regular expression. They cannot be added or removed later.

For example, `re = /\w+\s/g` creates a regular expression that looks for one or more characters followed by a space, and it looks for this combination throughout the string.

```
var re = /\w+\s/g;  
var str = "fee fi fo fum";  
var myArray = str.match(re);  
console.log(myArray);
```

This displays ["fee ", "fi ", "fo "]. In this example, you could replace the line:

```
var re = /\w+\s/g;
```

with:

```
var re = new RegExp("\\w+\\s", "g"); and get the same result.
```

The `m` flag is used to specify that a multiline input string should be treated as multiple lines. If the `m` flag is used, `^` and `$` match at the start or end of any line within the input string instead of the start or end of the entire string.

---

## **Errors & Exceptions Handling**

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

### **Syntax Errors**

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script  
  type="text/javascript">  
<!--  
  window.print(  
  //-->  
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

### **Runtime Errors**

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script
```

```

    type="text/javascript">
    <!--
    window.printme();
    //-->
</script>

```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

## Logical Errors

Logical errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

## The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax –

```

<script type="text/javascript">
    <!--
    try {
        // Code to run [break;]
    }

    catch ( e ) {
        // Code to run if an exception occurs
        [break;]
    }

    [ finally {
        // Code that is always executed regardless of
        // an exception occurring
    }
    ]
    //-->
</script>

```

The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

## Examples

Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without try...catch

```

<html>
<head>

    <script type="text/javascript">
        <!--

```

```

        function myFunc()
        {
            var a = 100;
            alert("Value of variable a is : " + a );
        }

        //-->
</script>
</head>

<body>
    <p>Click the following to see the result:</p>

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

</body>
</html>

```

---

## DOM event handling in detail with examples.

### Event Handler

An event handler executes a segment of a code based on certain events occurring within the application, such as onLoad, onClick. JavaScript event handlers can be divided into two parts:

- interactive event handlers
- non-interactive event handlers.

An interactive event handler is the one that depends on the user interactivity with the form or the document. For example, onMouseOver is an interactive event handler because it depends on the users action with the mouse.

Non-interactive event handler would be onLoad, because this event handler would automatically execute JavaScript code without the user's interactivity. Here are all the event handlers available in JavaScript:

#### Event Handler Used In

<u>onAbort</u>	image
<u>onBlur</u>	select, text, text area
<u>onChange</u>	select, text, <u>textarea</u>
<u>onClick</u>	button, checkbox, radio, link, reset, submit, area
<u>onError</u>	image
<u>onFocus</u>	select, text, <u>testarea</u>
<u>onLoad</u>	windows, image
<u>onMouseOver</u>	link, area
<u>onMouseOut</u>	link, area
<u>onSelect</u>	text, <u>textarea</u>
<u>onSubmit</u>	form
<u>onUnload</u>	window

#### onMouseOver:

Execute a JavaScript when moving the mouse pointer onto an image

#### BEFORE and AFTER BEFORE and AFTER



```

<html>
<title>example of event handler</title>

```

```

<head>
  <script language="javascript">
    function changeimage()
    {
      document.getElementById("image").src="images-2.jpeg";
    }
  </script>
</head>
<body>
  <h1>before and after</h1>
  
</body>
</html>

```

## onsubmit event handler:

Execute a JavaScript when a form is submitted:

```

<html>
<head>
<title>
  Online Banking System
</title>
<script type="text/javascript">
  function fun()
  {
    alert("Name:"+form1.username.value);
    alert("Password:"+form1.pwd.value);
    alert("Password:"+form1.mymenu.value);
    alert("Password:"+form1.accno.value);
  }
</script>
</head>
  <body bgcolor="pink">
    <h4>onsubmit event handler</h4>
    <form name="form1" onsubmit="fun()">
<table>
  <tr>
    <td><b>Name:</b></td>
    <td><input type="text" name="username"></td>
  </tr>
  <tr>
    <td><b>Password:</b></td>
    <td><input type="password" name="pwd"></td>
  </tr>

```

```

<tr>
<td><b>Sex:</b></td>
<td><input type="radio" selected="false" value="male">Male</td>
<td><input type="radio" value="female">Female</td>
</tr>
<tr>
<td><b>Account Number:</b></td>
<td>
<select name="mymenu">
<option value="saving account">Saving Account</option>
<option value="current account">Current Account</option>
<option value="zero account">Zero Account</option>
</select>
</td>
</tr>
<tr>
<td><b>Account Number:</b></td>
<td><input type="number" name="accno"></td>
</tr>
<tr>
<td><b>Facilities:</b></td>
<td><input type="checkbox">ATM card</td>
<td><input type="checkbox">Mobile banking</td>
<td><input type="checkbox">Internet banking</td>
</tr>
</table>
<br/>
<br/>
<center>
<input type="submit" value="submit">
<input type="reset" value="Clear">
</center>
</form>
</body>
</html>

```

### onclick event handler:

Execute a JavaScript when a button is clicked

### onsubmit event handler

**Name:**

**Password:**

**Sex:** ☒ Male ☐ Female

**Account Number:**

**Account Number:**

**Facilities:** ☒ ATM card ☐ Mobile banking ☒ Internet banking

This page says

Name:Raja selvan

```

<html>
<head>
<title>
Online Banking System
</title>
<script type="text/javascript">

```

```

function fun()
{
alert("Name:"+form1.username.value);
alert("Password:"+form1.pwd.value);
alert("Password:"+form1.mymenu.value);
alert("Password:"+form1.accno.value);
}
</script>
</head>
<body bgcolor="skyblue">
<h4>onsubmit event handler</h4>
<form name="form1">
<table>
<tr>
<td><b>Name:</b></td>
<td><input type="text" name="username"></td>
</tr>
<tr>
<td><b>Password:</b></td>
<td><input type="password" name="pwd"></td>
</tr>
<tr>
<td><b>Sex:</b></td>
<td><input type="radio" selected="false" value="male">Male</td>
<td><input type="radio" value="female">Female</td>
</tr>
<tr>
<td><b>Account Number:</b></td>
<td>
<select name="mymenu">
<option value="saving account">Saving Account</option>
<option value="current account">Current Account</option>
<option value="zero account">Zero Account</option>
</select>
</td>
</tr>
<tr>
<td><b>Account Number:</b></td>
<td><input type="number" name="accno"></td>
</tr>
<tr>
<td><b>Facilities:</b></td>
<td><input type="checkbox">ATM card</td>
<td><input type="checkbox">Mobile banking</td>
<td><input type="checkbox">Internet banking</td>
</tr>
</table>
<br/>
<br/>
<center>
<input type="submit" value="submit" onclick="fun()">
<input type="reset" value="Clear">
</center>
</form>
</body>
</html>

```



## DOM nodes and DOM trees can be used with example, traverse and modify DOM trees.

### The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page. The HTML DOM model is constructed as a tree of Objects

The HTML DOM is a standard object model and programming interface for HTML. It defines:

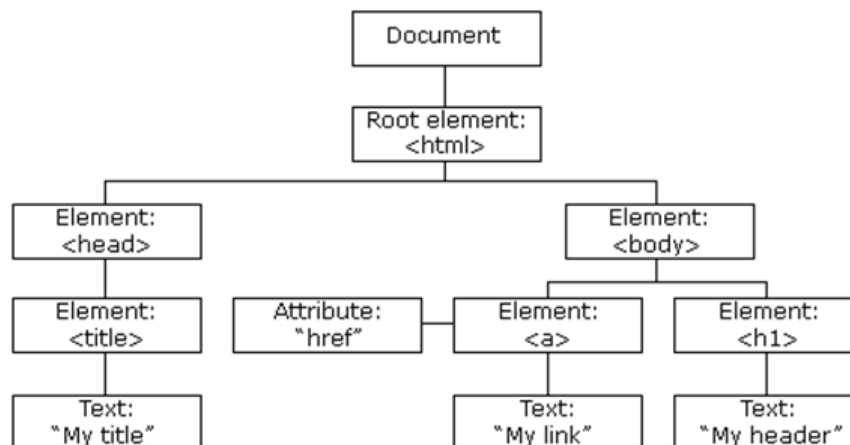
- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

### The HTML DOM Tree of Objects

#### Program:

```
<html>
<head><title>
My title
</title>
</head>
<body>
<a href=www.google.com>My link</a>
<h1>My header</h1>
</body>
</html>
```

A diagram of the above HTML document tree would look like this.



### Traverse DOM tree:

#### Ancestor

An ancestor refers to any element that is connected but further up the document tree - no matter how many levels higher.

#### Descendant

A descendant refers to any element that is connected but lower down the document tree - no matter how many levels lower.

#### Parent and Child

A parent is an element that is directly above and connected to an element in the document tree.

#### Sibling

A sibling is an element that shares the same parent with another element.

### Modifying element style-access,modify

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML *elements* in the page
- JavaScript can change all the HTML *attributes* in the page JavaScript can change all the CSS *styles* in the page

- JavaScript can remove existing HTML elements and attributes JavaScript can add new HTML elements and attributes JavaScript can react to all existing HTML events in the page JavaScript can create new HTML events in the page
- Any Changes made on the DOM using javascript properties and methods are reflected in document model instantly, this is based on the javascript interpreter of the browser.

This feature can be exploited even further to change the structure of the document itself. You can perform operations like **add** , **clone**, **remove**, **copy** etc on the elements on the document.

Users can alter the hierarchy of the DOM and the changes will be reflected in the document structure immediately.

The Javascript DOM properties and Methods used to alter the document hierarchy are explained in detail below.

Member	Description
appendChild(HTMLElement)	<i>To append the specified element as a child of the current element</i>
innerHTML	<i>To get or set the element's contents</i>
insertBefore(<newElem>, <childElem>)	<i>To insert the first element before the second element.</i>
removeChild(HTMLElement)	<i>To remove the specified child of the current element</i>
createElement(<tag>)	<i>To Replace a child of the current element</i>
createTextNode(<text>)	<i>To Replace a child of the current element</i>

### **createElement(),appendChild(),innerHTML-Insert element,text**

The createElement() method creates an Element Node with the specified name.

After the element is created, use the [element.appendChild\(\)](#) or [element.insertBefore\(\)](#) method to insert it to the document.

#### **Syntax**

document.createElement(*nodename*)

#### **Output:**

Click the button to create a P element with some text.

Try it

This is a paragraph.

#### **Example:**

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to create a P element with some text.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  var para = document.createElement("P");
  para.innerText = "This is a paragraph.";
  document.body.appendChild(para);
}
```

```

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

```

### **removeChild()**

before pressing Tryit button

- Coffee
- Tea
- Milk

Click the button to remove the first item from the list.

Try it

after pressing Tryit button twice

- Milk

Click the button to remove the first item from the list.

Try it

```

<!DOCTYPE html>
<html>
<body>
<ul id="myList">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<p>Click the button to remove the first item from the list.</p>
<button onclick="myFunction()">Try it</button>
<script>
  function myFunction()
  {
    var list = document.getElementById("myList");
    list.removeChild(list.childNodes[0]);
  }
</script>
</body>
</html>

```

---

### **DHTML JavaScript**

- DHTML stands for Dynamic HTML.
- Dynamic means that the content of the web page can be customized or changed according to user inputs i.e. a page that is interactive with the user. In earlier times, HTML was used to create a static page.
- It only defined the structure of the content that was displayed on the page. With the help of CSS, we can beautify the HTML page by changing various properties like text size, background color etc.
- The HTML and CSS could manage to navigate between static pages but couldn't do anything else.
- If 1000 users view a page that had their information for eg. Admit card then there was a problem because 1000 static pages for this application build to work.
- As the number of users increase, the problem also increases and at some point it becomes impossible to handle this problem.
- To overcome this problem, DHTML came into existence. DHTML included JavaScript along with HTML and CSS to make the page dynamic.
- This combo made the web pages dynamic and eliminated this problem of creating static page for each user. To integrate JavaScript into HTML, a Document Object Model(DOM) is made for the HTML document.

### **Moving elements (Positioning Elements):**

**Static Positioning** does not have top and left properties, so an image which is positioned as Static can't be moved.

- **Absolute positioning:** The image moves to the new location according to the values of top and left.
- **Relative positioning:** Here the image moves relative to its original position. From its original position, it applies the new top and left value and moves accordingly.

**Write a xHTML program to take input from the user and move the image to the corresponding location(position), using Javascript.**

```
<html>
<head><title> Move Image </title>

<style type="text/css">
    #kids { position: relative; top: 200px; left: 400px; }
</style>

<script type="text/javascript">

function moveIt()
{
    var x = document.getElementById("x").value;
    var y = document.getElementById("y").value;
    var kids = document.getElementById("kids").style;
    kids.top = x + "px";
    kids.left = y + "px";
}
</script>
</head>
<body>
    <form>
        X axis: <input type="text" id="x"><br />
        Y axis: <input type="text" id="y"><br />
        <input type="button" value=" Move " onclick="moveIt()" >
    </form>
<div id="kids">
    
</div>
</body>
</html>
```

**Output:**

X axis:   
Y axis:



---

**Design web page to create a clock with a timing event.(onLoad event)**

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

```

<script>
function startTime()
{
    var today = new Date();
    var h = today.getHours();
    var m = today.getMinutes();
    var s = today.getSeconds();
    m = checkTime(m);
    s = checkTime(s);
    document.getElementById('txt').innerHTML =
    h + ":" + m + ":" + s;
    var t = setTimeout(startTime, 500);
}
function checkTime(i)
{
    if (i < 10) {i = "0" + i}; // add zero in front of numbers < 10
    return i;
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>
</html>

```

#### **Output:**

10:07:22

---

### **Operators in javascript**

A simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

#### **concatenation operation**

+ operator performs concatenation operation when one of the operands is of string type. The following example shows how + operator performs operation on operands of different data types.

```
var a = 5, b = "Hello ", c = "World!", d = 10;
```

```
a + b; // "5Hello "
```

```
b + c; // "Hello World!"
```

```
a + d; // 15
```

#### **Arithmetic Operators**

JavaScript supports the following arithmetic operators Assume variable A holds 10 and variable B holds 20, then

Sr.No.	Operator & Description
1	<b>+</b> ( <b>Addition</b> ) Adds two operands <b>Ex:</b> A + B will give 30
2	<b>-</b> ( <b>Subtraction</b> ) Subtracts the second operand from the first <b>Ex:</b> A - B will give -10
3	<b>*</b> ( <b>Multiplication</b> ) Multiply both operands <b>Ex:</b> A * B will give 200
4	<b>/</b> ( <b>Division</b> ) Divide the numerator by the denominator <b>Ex:</b> B / A will give 2
5	<b>%</b> ( <b>Modulus</b> ) Outputs the remainder of an integer division <b>Ex:</b> B % A will give 0
6	<b>++</b> ( <b>Increment</b> ) Increases an integer value by one <b>Ex:</b> A++ will give 11
7	<b>--</b> ( <b>Decrement</b> ) Decreases an integer value by one <b>Ex:</b> A-- will give 9

**Note** – Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

### Comparison Operators

JavaScript supports the following comparison operators

Assume variable A holds 10 and variable B holds 20, then

Sr.No.	Operator & Description
1	<b>==</b> ( <b>Equal</b> )

	<p>Checks if the value of two operands are equal or not, if yes, then the condition becomes true.</p> <p><b>Ex:</b> (A == B) is not true.</p>
2	<p><b>!= (Not Equal)</b></p> <p>Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.</p> <p><b>Ex:</b> (A != B) is true.</p>
3	<p><b>&gt; (Greater than)</b></p> <p>Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.</p> <p><b>Ex:</b> (A &gt; B) is not true.</p>
4	<p><b>&lt; (Less than)</b></p> <p>Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.</p> <p><b>Ex:</b> (A &lt; B) is true.</p>
5	<p><b>&gt;= (Greater than or Equal to)</b></p> <p>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p><b>Ex:</b> (A &gt;= B) is not true.</p>
6	<p><b>&lt;= (Less than or Equal to)</b></p> <p>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p><b>Ex:</b> (A &lt;= B) is true.</p>

### Logical Operators

JavaScript supports the following logical operators.

Assume variable A holds 10 and variable B holds 20, then –

Sr.No.	Operator & Description
1	<p><b>&amp;&amp; (Logical AND)</b></p> <p>If both the operands are non-zero, then the condition becomes true.</p> <p><b>Ex:</b> (A &amp;&amp; B) is true.</p>
2	<p><b>   (Logical OR)</b></p>

	<p>If any of the two operands are non-zero, then the condition becomes true.</p> <p><b>Ex:</b> (A    B) is true.</p>
3	<p><b>! (Logical NOT)</b></p> <p>Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.</p> <p><b>Ex:</b> ! (A &amp;&amp; B) is false.</p>

## **Bitwise Operators**

JavaScript supports the following bitwise operators.

Assume variable A holds 2 and variable B holds 3, then

Sr.No.	Operator & Description
1	<p><b>&amp; (Bitwise AND)</b></p> <p>It performs a Boolean AND operation on each bit of its integer arguments.</p> <p><b>Ex:</b> (A &amp; B) is 2.</p>
2	<p><b>  (BitWise OR)</b></p> <p>It performs a Boolean OR operation on each bit of its integer arguments.</p> <p><b>Ex:</b> (A   B) is 3.</p>
3	<p><b>^ (Bitwise XOR)</b></p> <p>It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.</p> <p><b>Ex:</b> (A ^ B) is 1.</p>
4	<p><b>~ (Bitwise Not)</b></p> <p>It is a unary operator and operates by reversing all the bits in the operand.</p> <p><b>Ex:</b> (~B) is -4.</p>
5	<p><b>&lt;&lt; (Left Shift)</b></p> <p>It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.</p> <p><b>Ex:</b> (A &lt;&lt; 1) is 4.</p>



6	<b>&gt;&gt; (Right Shift)</b> Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. <b>Ex:</b> (A >> 1) is 1.
7	<b>&gt;&gt;&gt; (Right shift with Zero)</b> This operator is just like the >> operator, except that the bits shifted in on the left are always zero. <b>Ex:</b> (A >>> 1) is 1.

### Assignment Operators

JavaScript supports the following assignment operators

Sr.No.	Operator & Description
1	<b>= (Simple Assignment )</b> Assigns values from the right side operand to the left side operand <b>Ex:</b> C = A + B will assign the value of A + B into C
2	<b>+= (Add and Assignment)</b> It adds the right operand to the left operand and assigns the result to the left operand. <b>Ex:</b> C += A is equivalent to C = C + A
3	<b>-= (Subtract and Assignment)</b> It subtracts the right operand from the left operand and assigns the result to the left operand. <b>Ex:</b> C -= A is equivalent to C = C - A
4	<b>*= (Multiply and Assignment)</b> It multiplies the right operand with the left operand and assigns the result to the left operand. <b>Ex:</b> C *= A is equivalent to C = C * A
5	<b>/= (Divide and Assignment)</b> It divides the left operand with the right operand and assigns the result to the left operand. <b>Ex:</b> C /= A is equivalent to C = C / A
6	<b>%= (Modules and Assignment)</b>

	<p>It takes modulus using two operands and assigns the result to the left operand.</p> <p><b>Ex:</b> <math>C \% = A</math> is equivalent to <math>C = C \% A</math></p>
--	---

**Note** – Same logic applies to Bitwise operators so they will become like  $\ll$ ,  $\gg$ ,  $\gg$ ,  $\&$ ,  $|$  and  $\wedge$ .

### conditional/ternary operator

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No.	Operator and Description
1	<p><b>? : (Conditional)</b></p> <p>If Condition is true? Then value X : Otherwise value Y</p>

```
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
```

### typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

### **Program:**

```
<!DOCTYPE html>
```

```

<html>
<body>
<p>OPERATORS</p>
<p id="demo"> </p>
<script>
    var a=5;
    var b=7;
    var person = "PRATHEE";
    document.write("Welcome "+person);
    document.write("<br>arithmetic<br>");
    document.write(a+b);
    document.write("<br>logical<br>");
    document.write((a>b)&&(a==b));
    document.write("<br>comparison<br>");
    document.write(a!=b);
    document.write("<br>");
</script>
</body>
</html>

```

**Output:**

OPERATORS

Welcome PRATHEE  
arithmetic  
12  
logical  
false  
comparison  
true

## **Identifier, keywords, comments**

### **IDENTIFIERS**

- Name given in variables
- Begin with either letter or underscore or dollar sign
- Length- no limit
- Case sensitive

### **KEYWORDS(Reserved word)**

- Special meaning
- E.G.
- break ,continue, if, else, try, throw, catch, void, case, switch, for

### **COMMENTS**

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

### **Example**

```

<script language="javascript" type="text/javascript">
<!--

```

```

// this is a comment. It is similar to comments in C++

```

```

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming

```

```
*/  
//-->  
<script>
```

---

### **Input and Output in javascript:**

#### **HTML DOM write() Method:**

The write() method writes HTML expressions or JavaScript code to a document. The write() method is mostly used for testing: If it is used after an HTML document is fully loaded, it will delete all existing HTML.

##### **Syntax**

```
document.write(exp1, exp2, exp3, ...)
```

##### **Example:**

```
document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");
```

#### **popup in javascript:**

In Javascript, popup boxes are used to display the message or notification to the user. There are three types of pop up boxes in JavaScript namely **Alert Box**, **Confirm Box** and **Prompt Box**.

**Alert Box:** It is used when a warning message is needed to be produced. When the alert box is displayed to the user, the user needs to press ok and proceed.

##### **Syntax:**

```
alert("your Alert here")
```

**Prompt Box:** It is a type of pop up box which is used to get the user input for further use. After entering the required details user have to click ok to proceed next stage else by pressing the cancel button user returning the null value.

##### **Syntax:**

```
prompt("your Prompt here")
```

**Confirm Box** It is a type of pop up box which is used to get the authorization or permission from the user. The user has to press the ok or cancel button to proceed.

##### **Syntax:**

```
confirm("your query here")
```

#### **Example:**

```
<!DOCTYPE html>  
<html>  
<body>  
<h1>Demo: popup</h1>  
  <script>  
    var a,b;  
    if (confirm("Do you want to save changes?") == true)  
    {  
      alert("Data saved successfully!");  
    }  
    else  
    {  
      b=prompt("enter name", " ");  
      alert("Hi "+b);  
    }  
  </script>  
</body>  
</html>
```

##### **Output:**

<p>This page says</p> <p>Do you want to save changes?</p> <p><input type="button" value="OK"/> <input type="button" value="Cancel"/></p>	<p>If press ok</p> <p>This page says</p> <p>Data saved successfully!</p> <p><input type="button" value="OK"/></p>
<p>If press cancel</p>	
<p>This page says</p> <p>enter name</p> <p><input type="text" value="Pratheeba"/></p> <p><input type="button" value="OK"/> <input type="button" value="Cancel"/></p>	<p>This page says</p> <p>Hi Pratheeba</p> <p><input type="button" value="OK"/></p>

## Control structures in javascript

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

### Key Points

- It is Lightweight, interpreted programming language.
- It is designed for creating network-centric applications.
- It is complementary to and integrated with Java.
- It is complementary to and integrated with HTML
- It is an open and cross-platform

### JavaScript Statements

JavaScript statements are the commands to tell the browser to what action to perform. Statements are separated by semicolon (;).

Following table shows the various JavaScript Statements

Sr.No.	Statement	Description
1.	switch case	A block of statements in which execution of code depends upon different cases. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.
2.	If else	The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
3.	While	The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.
4.	do while	Block of statements that are executed at least once and continues to be executed while condition is true.
5.	for	Same as while but initialization, condition and increment/decrement is done in the same line.
6.	for in	This loop is used to loop through an object's properties.

7.	continue	The continue statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.
8.	break	The break statement is used to exit a loop early, breaking out of the enclosing curly braces.
9.	function	A function is a group of reusable code which can be called anywhere in your programme. The keyword function is used to declare a function.
10.	return	Return statement is used to return a value from a function.
11.	var	Used to declare a variable.
12.	try	A block of statements on which error handling is implemented.
13.	catch	A block of statements that are executed when an error occur.
14.	throw	Used to throw an error.

#### Program:( if...else if...else)

```

<html>
<body>
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
{
document.write("<b>Hello World!</b>");
}
</script>
</body>
</html>

```

#### Program:( switch)

```

<html>
<body>
<script type="text/javascript">
//You will receive a different greeting based on what day it is. Note that Sunday=0, Monday=1, Tuesday=2, etc.
var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
document.write("Finally Friday"); break;
case 6:
document.write("Super Saturday"); break;
case 0:
document.write("Sleepy Sunday"); break;

```

```
default:
document.write("I'm looking forward to this weekend!");
}
</script>
</body>
</html>
```

---

## **Functions in javascript**

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes.

Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters.

We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well.

### **Function Definition**

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

#### **Syntax**

```
<script type = "text/javascript">
  <!--
    function functionname(parameter-list) {
      statements
    }
  //-->
</script>
```

#### **Example**

Try the following example. It defines a function called sayHello that takes no parameters –

```
<script type = "text/javascript">
  <!--
    function sayHello() {
      alert("Hello there");
    }
  //-->
</script>
```

### **Calling a Function**

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Internal Script</title>
    <script type = "text/JavaScript">
```

```

function Hello()
{
    alert("Hello, World");
}
</script>
</head>
<body>
    <input type = "button" onclick = "Hello();" name = "ok" value = "Click Me" />
</body>
</html>

```

## Function Parameters

we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

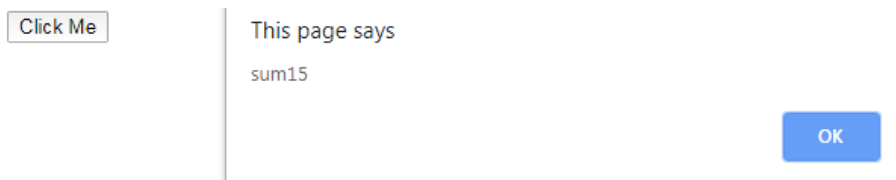
### Example

```

<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript Internal Script</title>
        <script type = "text/JavaScript">
            function Hello(a,b)
            {
                alert("sum"+(a+b));
            }
        </script>
    </head>
    <body>
        <input type = "button" onclick = "Hello(5,10);" name = "ok" value = "Click Me" />
    </body>
</html>

```

### Output:



## The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

### Example



```

<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Internal Script</title>
    <script type = "text/JavaScript">
      function Hello(a,b)
      {
        return a+b;
      }
    </script>
  </head>
  <body>
    <p>the sum is
    <script type = "text/JavaScript">
      document.write(Hello(6,7));
    </script>
    </p></body>
</html>

```

**Output:**

the sum is 13

**Global functions in javascript:**

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

JavaScript Global Properties

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

JavaScript Global Functions

Function	Description
decodeURI()	Decodes a URI
encodeURI()	Encodes a URI
eval()	Evaluates a string and executes it as if it was script code
isNaN()	Determines whether a value is an illegal number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer

**encodeURI()**

The **encodeURI()** function encodes a Uniform Resource Identifier (URI) by replacing each instance of certain characters by one, two, three, or four escape sequences representing the UTF-8 encoding of the character

*encodeURI(URI)*

Does not encode characters that have special meaning (reserved characters) for a URI. The following example shows all the parts that a URI "scheme" can possibly contain. Note how certain characters are used to signify special meaning:

```
http://username:password@www.example.com:80/path/to/file.php?foo=316&bar=this+has+spaces#anchor
```

Hence encodeURI **does not** encode characters that are necessary to formulate a complete URI. Also, encodeURI **does not** encode a few additional characters, known as "unreserved marks",

#### **Not Escaped:**

A-Z a-z 0-9 ; , / ? : @ & = + \$ - \_ . ! ~ \* ' ( ) #

#### **decodeURI()**

The **decodeURI()** function decodes a Uniform Resource Identifier (URI) previously created by encodeURI() or by a similar routine.

*decodeURI(encodedURI)*

Replaces each escape sequence in the encoded URI with the character that it represents, but does not decode escape sequences that could not have been introduced by encodeURI. The character “#” is not decoded from escape sequences.

#### **eval()**

The **eval()** function evaluates JavaScript code represented as a string.

*eval(string)*

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>global fun</title>
  </head>
  <body>
    <script type = "text/JavaScript">
      var set3 = "ABC abc 123";
      document.write(eval('2+4*3')+"<br>");
      document.write(encodeURI(set3)+"<br>");
      document.write(decodeURI(set3)+"<br>");
    </script>
  </body>
</html>
```

**Output:**

```
14
ABC%20abc%20123
ABC abc 123
```

---

### **Arrays in JavaScript**

In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements.

#### **Declaration of an Array**

There are basically two ways to declare an array.

```
var House = [ ];           // method 1
var House = new array();    // method 2
```

But generally method 1 is preferred over the method 2. Let us understand the reason for this.

### Initialization of an Array

```
// Initializing while declaring
var house = ["1BHK", "2BHK", "3BHK", "4BHK"];
// Creates an array having elements 10, 20, 30, 40, 50
var house = new Array(10, 20, 30, 40, 50);

//Creates an array of 5 undefined elements
var house1 = new Array(5);

//Creates an array with element 1BHK
var home = new Array("1BHK");
```

As shown in above example the **house** contains 5 elements i.e. (10, 20, 30, 40, 50) while **house1** contains 5 **undefined elements** instead of having a **single element 5**. Hence, while working with numbers this method is generally **not preferred** but it works **fine with Strings** and Boolean as shown in the example above home contains a single element 1BHK.

### An array with different elements

We can store Numbers, Strings and Boolean in a single array.

#### Example:

```
// Storing number, boolean, strings in an Array
var house = ["1BHK", 25000, "2BHK", 50000, "Rent", true];
```

### Accessing Array Elements

Array in JavaScript are indexed from 0 so we can access array elements as follows:

```
var house = ["1BHK", 25000, "2BHK", 50000, "Rent", true];
var rent = house[5];
alert("Is house for rent = " + rent);
```

#### Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

#### Output:

## JavaScript Arrays

Saab, Volvo, BMW

### Length property of an Array

Length property of an Array returns the length of an Array. Length of an Array is always one more than the highest index of an Array.

#### Example

```
var house = ["1BHK", 25000, "2BHK", 50000, "Rent", true];

//len contains the length of the array
var len = house.length;
for (var i = 0; i < len; i++)
    alert(house[i]);
```

## 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.

---

## Objects in javascript

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

## Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyValue;
```

## Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example** – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

## The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are `Object()`, `Array()`, and `Date()`. These constructors are built-in JavaScript functions.

```
var employee = new Object();  
var books = new Array("C++", "Perl", "Java");
```

## User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects –

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

## The Date Object

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date()** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

## Syntax

The following syntaxes to create a Date object using Date() constructor.

```
new Date( )  
new Date(milliseconds)  
new Date(datestring)  
new Date(year,month,date[,hour,minute,second,millisecond ])
```

Here is a description of the parameters –

- **No Argument** – With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds** – When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring** – When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
- **7 arguments** – To use the last form of the constructor shown above. Here is a description of each argument –
  - **year** – Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
  - **month** – Integer value representing the month, beginning with 0 for January to 11 for December.
  - **date** – Integer value representing the day of the month.
  - **hour** – Integer value representing the hour of the day (24-hour scale).
  - **minute** – Integer value representing the minute segment of a time reading.
  - **second** – Integer value representing the second segment of a time reading.
  - **millisecond** – Integer value representing the millisecond segment of a time reading.

## Date Methods

Here is a list of the methods used with **Date** and their description.

Sr.No.	Method & Description
1	Date() Returns today's date and time
2	getDate() Returns the day of the month for the specified date according to local time.
3	getDay() Returns the day of the week for the specified date according to local time.
4	getHours() Returns the hour in the specified date according to local time.
5	getMinutes() Returns the minutes in the specified date according to local time.
6	getMonth() Returns the month in the specified date according to local time.

## The Math Object

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using **Math** as an object without creating it.

Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

## Syntax

The syntax to call the properties and methods of **Math** are as follows

```
var pi_val = Math.PI;  
var sine_val = Math.sin(30);
```

## Math Methods

Here is a list of the methods associated with **Math** object and their description

Sr.No.	Method & Description
1	abs() Returns the absolute value of a number.
2	ceil()

	Returns the smallest integer greater than or equal to a number.
3	cos() Returns the cosine of a number.

## The Strings Object

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

### Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded

### String Methods

Here is a list of the methods available in String object along with their description.

Sr.No.	Method & Description
1	concat() Combines the text of two strings and returns a new string.
2	toLowerCase() Returns the calling string value converted to lower case.
3	toString() Returns a string representing the specified object.
4	toUpperCase() Returns the calling string value converted to uppercase.

## The Number Object

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

### Syntax

The syntax for creating a **number** object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

### Number Properties

Here is a list of each property and their description.

Sr.No.	Property & Description
1	MAX_VALUE

	The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	MIN_VALUE The smallest possible value a number in JavaScript can have 5E-324
3	NaN Equal to a value that is not a number.
4	NEGATIVE_INFINITY A value that is less than MIN_VALUE.

## Number Methods

The Number object contains only the default methods that are a part of every object's definition.

Sr.No.	Method & Description
1	toExponential() Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	toString() Returns the string representation of the number's value.

### Program:

```

<!DOCTYPE html>
<html>
<head>
<script>
function startTime()
{
    var today = new Date();
    var val = new Number(8);
    var pi=Math.PI;
    var h = today.getHours();
    var m = today.getMinutes();
    var s = today.getSeconds();
    m = checkTime(m);
    s = checkTime(s);
    document.getElementById('txt').innerHTML = h + ":" + m + ":" + s+"<br> pi value"+pi;
    var t = setTimeout(startTime, 500);
}
function checkTime(i)
{
    if (i < 10) {i = "0" + i}; // add zero in front of numbers < 10
    return i;
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>

```



</html>

---