❖ **Adding local and remote links:**

HTML links are hyperlinks.You can click on a link and jump to another document.When you move the mouse over a link, the mouse arrow will turn into a little hand.

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

❖ **Creating lists in HTML:**

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain −

- **<ul>** − An unordered list. This will list items using plain bullets.
- **<ol>** − An ordered list. This will use different schemes of numbers to list your items.
- **<dl>** − A definition list. This arranges your items in the same way as they are arranged in a dictionary.

1)**Unordered Lists**

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML **<ul>** tag. Each item in the list is marked with a bullet.

- Live Demo

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Unordered List</title>
  </head>

  <body>
    <ul>
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ul>
  </body>

```

```
</html>
```

2)**Ordered Lists:**

If you are required to put your items in a numbered list instead of bulleted, then HTML ordered list will be used. This list is created by using **<ol>** tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Ordered List</title>
  </head>

  <body>
    <ol>
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>

</html>
```

❖ HTML – Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells. The elements under <td> are regular and left aligned by default

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Tables</title>
```

```
  </head>

  <body>
    <table border = "1">
      <tr>
        <td>Row 1, Column 1</td>
        <td>Row 1, Column 2</td>
      </tr>

      <tr>
        <td>Row 2, Column 1</td>
        <td>Row 2, Column 2</td>
      </tr>
    </table>

  </body>
</html>
```

Cellpadding and Cellspacing Attributes

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells. The cellspacing attribute defines space between table cells, while cellpadding represents the distance between cell borders and the content within a cell.

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Table Cellpadding</title>
  </head>

  <body>
    <table border = "1" cellpadding = "5" cellspacing = "5">
      <tr>
        <th>Name</th>
        <th>Salary</th>
      </tr>
      <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
```

```
      </tr>
      <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
      </tr>
    </table>
  </body>

</html>
```

❖ **Dividing the window with frames:**

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages −

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back* button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

```
<!DOCTYPE html>
```

```
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "/html/top_frame.htm" />
    <frame name = "main" src = "/html/main_frame.htm" />
    <frame name = "bottom" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>

  </frameset>

</html>
```

The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag −

Sr.No	Attribute & Description
1	**cols** Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways − Absolute values in pixels. For example, to create three vertical frames, use *cols = "100, 500, 100"*. A percentage of the browser window. For example, to create three vertical frames, use *cols = "10%, 80%, 10%"*. Using a wildcard symbol. For example, to create three vertical frames, use *cols = "10%, *, 10%"*. In this case wildcard takes remainder of the window. As relative widths of the browser window. For example, to create three vertical frames, use *cols = "3*, 2*, 1*"*. This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.

2	**rows** This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use *rows = "10%, 90%"*. You can specify the height of each row in the same way as explained above for columns.
3	**border** This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.
4	**frameborder** This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder = "0" specifies no border.
5	**framespacing** This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing = "10" means there should be 10 pixels spacing between each frames.

The <frame> Tag Attributes

Sr.No	Attribute & Description
1	**src** This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.
2	**name** This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.

3	**frameborder** This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
4	**marginwidth** This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth = "10".
5	**marginheight** This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight = "10".
6	**noresize** By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize = "noresize".
7	**scrolling** This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling = "no" means it should not have scroll bars.
8	**longdesc** This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc = "framedescription.htm"

❖ JavaScript Overview

What is JavaScript ?

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

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

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Client-Side JavaScript

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

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

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

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

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

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

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

❖ Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

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

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100

Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type = "text/javascript">
  <!--
    var money;
    var name;
  //-->
</script>
```

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

❖ JavaScript Variable Scope

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

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

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

❖ JavaScript | Type Conversion

JavaScript is loosely typed language and most of the time operators automatically convert a value to the right type but there are also cases when we need to explicitly do type conversions.
While JavaScript provides numerous ways to convert data from one type to another but there are two most common data conversions :

- **Converting Values to String**
- **Converting Values to Number**

Implicit Conversion:
There are various operator and functions in JavaScript which automatically converts a value to the right type like alert() function in JavaScript accepts any value and convert it into a string. But various operator creates a problem like '+' operator.
Example:

- **Input:** "2" + "3"
- **Output:** "23"
- here + operator stands for string concatenation in this case.
- But **"3" - "1"** gives output **2** by using Implicit Conversion.

Converting Values to Strings:
String() or **toString()** function can be used in JavaScript to convert a value to a string.
Syntax of String() function:

- String(value)
- <script>
-
- // Number and date has been assigned
- // to variable v and d respectively

```
    var v = 123;
    var d = new Date('1995-12-17T03:24:00');

    // Conversion of number to string
    document.write(" String(v) = " + String(v) + "<br>");

    // Conversion of number to string
    document.write(" String(v + 11) = " + String(v + 11) + "<br>");
    document.write(" String( 10 + 10) = " + String(10 + 10) + "<br>");

    // Conversion of boolean value to string
    document.write(" String(false) = " + String(false) + "<br>");

    // Conversion of Date to string
    document.write(" String(d) = " + String(d) + "<br>");

</script>
```

Output:

```
String(v) = 123
String(v + 11) = 134
String( 10 + 10) = 20
String(false) = false
String(d) = Sun Dec 17 1995 03:24:00 GMT+0530 (India Standard Time)
```

JavaScript - Operators

What is an Operator?

Let us take 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.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

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

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

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

Example

The following code shows how to use arithmetic operators in JavaScript.

Live Demo

```
<html>
  <body>

    <script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);

        document.write("a / b = ");
        result = a / b;
        document.write(result);
        document.write(linebreak);

        document.write("a % b = ");
        result = a % b;
        document.write(result);
        document.write(linebreak);

        document.write("a + b + c = ");
        result = a + b + c;
        document.write(result);
```

```
         document.write(linebreak);

         a = ++a;
         document.write("++a = ");
         result = ++a;
         document.write(result);
         document.write(linebreak);

         b = --b;
         document.write("--b = ");
         result = --b;
         document.write(result);
         document.write(linebreak);
       //-->
    </script>

    Set the variables to different values and then try...
  </body>
</html>
```

Output

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...

Comparison Operators

JavaScript supports the following comparison operators −

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

Sr.No.	Operator & Description
1	**= = (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true.

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

Example

The following code shows how to use comparison operators in JavaScript.

Live Demo

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

```
      <!--
         var a = 10;
         var b = 20;
         var linebreak = "<br />";

         document.write("(a == b) => ");
         result = (a == b);
         document.write(result);
         document.write(linebreak);

         document.write("(a < b) => ");
         result = (a < b);
         document.write(result);
         document.write(linebreak);

         document.write("(a > b) => ");
         result = (a > b);
         document.write(result);
         document.write(linebreak);

         document.write("(a != b) => ");
         result = (a != b);
         document.write(result);
         document.write(linebreak);

         document.write("(a >= b) => ");
         result = (a >= b);
         document.write(result);
         document.write(linebreak);

         document.write("(a <= b) => ");
         result = (a <= b);
         document.write(result);
         document.write(linebreak);
      //-->
   </script>
   Set the variables to different values and different operators and then try...
  </body>
</html>
```

Output
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators and then try...

Logical Operators

JavaScript supports the following logical operators −

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

Sr.No.	Operator & Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

Live Demo

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = true;
```

```
         var b = false;
         var linebreak = "<br />";

         document.write("(a && b) => ");
         result = (a && b);
         document.write(result);
         document.write(linebreak);

         document.write("(a || b) => ");
         result = (a || b);
         document.write(result);
         document.write(linebreak);

         document.write("!(a && b) => ");
         result = (!(a && b));
         document.write(result);
         document.write(linebreak);
      //-->
   </script>
   <p>Set the variables to different values and different operators and then
try...</p>
  </body>
</html>
```

Output
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators and then try...

Bitwise Operators

JavaScript supports the following bitwise operators −

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

Sr.No.	Operator & Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments.

	Ex: (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** 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. **Ex:** (A ^ B) is 1.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)** 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. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.
7	**>>> (Right shift with Zero)** This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Example

Try the following code to implement Bitwise operator in JavaScript.

Live Demo

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11
        var linebreak = "<br />";

        document.write("(a & b) => ");
        result = (a & b);
        document.write(result);
        document.write(linebreak);

        document.write("(a | b) => ");
        result = (a | b);
        document.write(result);
        document.write(linebreak);

        document.write("(a ^ b) => ");
        result = (a ^ b);
        document.write(result);
        document.write(linebreak);

        document.write("(~b) => ");
        result = (~b);
        document.write(result);
        document.write(linebreak);

        document.write("(a << b) => ");
        result = (a << b);
        document.write(result);
        document.write(linebreak);

        document.write("(a >> b) => ");
        result = (a >> b);
        document.write(result);
```

```
         document.write(linebreak);
        //-->
      </script>
      <p>Set the variables to different values and different operators and then
try...</p>
    </body>
</html>
```

(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and then try...

Assignment Operators

JavaScript supports the following assignment operators −

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

	Ex: C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example

Try the following code to implement assignment operator in JavaScript.

Live Demo

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var linebreak = "<br />";

        document.write("Value of a => (a = b) => ");
        result = (a = b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a += b) => ");
        result = (a += b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a -= b) => ");
        result = (a -= b);
```

```
         document.write(result);
         document.write(linebreak);

         document.write("Value of a => (a *= b) => ");
         result = (a *= b);
         document.write(result);
         document.write(linebreak);

         document.write("Value of a => (a /= b) => ");
         result = (a /= b);
         document.write(result);
         document.write(linebreak);

         document.write("Value of a => (a %= b) => ");
         result = (a %= b);
         document.write(result);
         document.write(linebreak);
       //-->
     </script>
     <p>Set the variables to different values and different operators and then
try...</p>
   </body>
</html>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators and then try...
```

Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

Conditional 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	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

Live Demo

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
         var a = 10;
         var b = 20;
         var linebreak = "<br />";

         document.write ("((a > b) ? 100 : 200) => ");
         result = (a > b) ? 100 : 200;
         document.write(result);
         document.write(linebreak);

         document.write ("((a < b) ? 100 : 200) => ");
         result = (a < b) ? 100 : 200;
         document.write(result);
         document.write(linebreak);
      //-->
    </script>
    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators and then try...

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"

Example

The following code shows how to implement **typeof** operator.

Live Demo

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 10;
        var b = "String";
        var linebreak = "<br />";

        result = (typeof b == "string" ? "B is String" : "B is Numeric");
        document.write("Result => ");
        document.write(result);
```

```
         document.write(linebreak);

         result = (typeof a == "string" ? "A is String" : "A is Numeric");
         document.write("Result => ");
         document.write(result);
         document.write(linebreak);
       //-->
    </script>
    <p>Set the variables to different values and different operators and then
try...</p>
  </body>
</html>
```

Output
Result => B is String
Result => A is Numeric
Set the variables to different values and different operators and then try...

JavaScript - The Arrays Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** object −
var fruits = new Array("apple", "orange", "mango");

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every() Returns true if every element in this array satisfies the provided testing function.
3	filter() Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	forEach() Calls a function for each element in the array.
5	indexOf() Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	join() Joins all elements of an array into a string.
7	lastIndexOf() Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	map() Creates a new array with the results of calling a provided function on every element in this array.
9	pop() Removes the last element from an array and returns that element.
10	push() Adds one or more elements to the end of an array and returns the new length of the array.

11	reduce() Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	reduceRight() Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	reverse() Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	shift() Removes the first element from an array and returns that element.
15	slice() Extracts a section of an array and returns a new array.
16	some() Returns true if at least one element in this array satisfies the provided testing function.
17	toSource() Represents the source code of an object
18	sort() Sorts the elements of an array
19	splice() Adds and/or removes elements from an array.
20	toString() Returns a string representing the array and its elements.
21	unshift() Adds one or more elements to the front of an array and returns the new length of the array.

JavaScript - Dialog 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.

Live Demo

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "Warn();" />
    </form>
  </body>
</html>
```

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

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.

❖ Custom Functions

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

❖ Working with Frame

Working with Frames

Some browsers (including the latest Netscape and Microsoft browsers) support *frames*, which enable you to divide the browser window into multiple panes. Each frame can contain a separate URL or the output of a script.

Using JavaScript Objects for Frames

When a window contains multiple frames, each frame is represented in JavaScript by a frame object. This object is equivalent to a window object, but it is used for dealing with that frame. The frame object's name is the same as the NAME attribute you give it in the <frame> tag.

Remember the window and self keywords, which refer to the current window? When you are using frames, these keywords refer to the current frame instead. Another keyword, parent, enables you to refer to the main window.

```
<frameset ROWS="*,*" COLS="*,*">

<frame NAME="topleft" SRC="topleft.htm">

<frame NAME="topright" SRC="topright.htm">

<frame NAME="bottomleft" SRC="botleft.htm">

<frame NAME="bottomright" SRC="botright.htm">

</frameset>
```

❖ JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

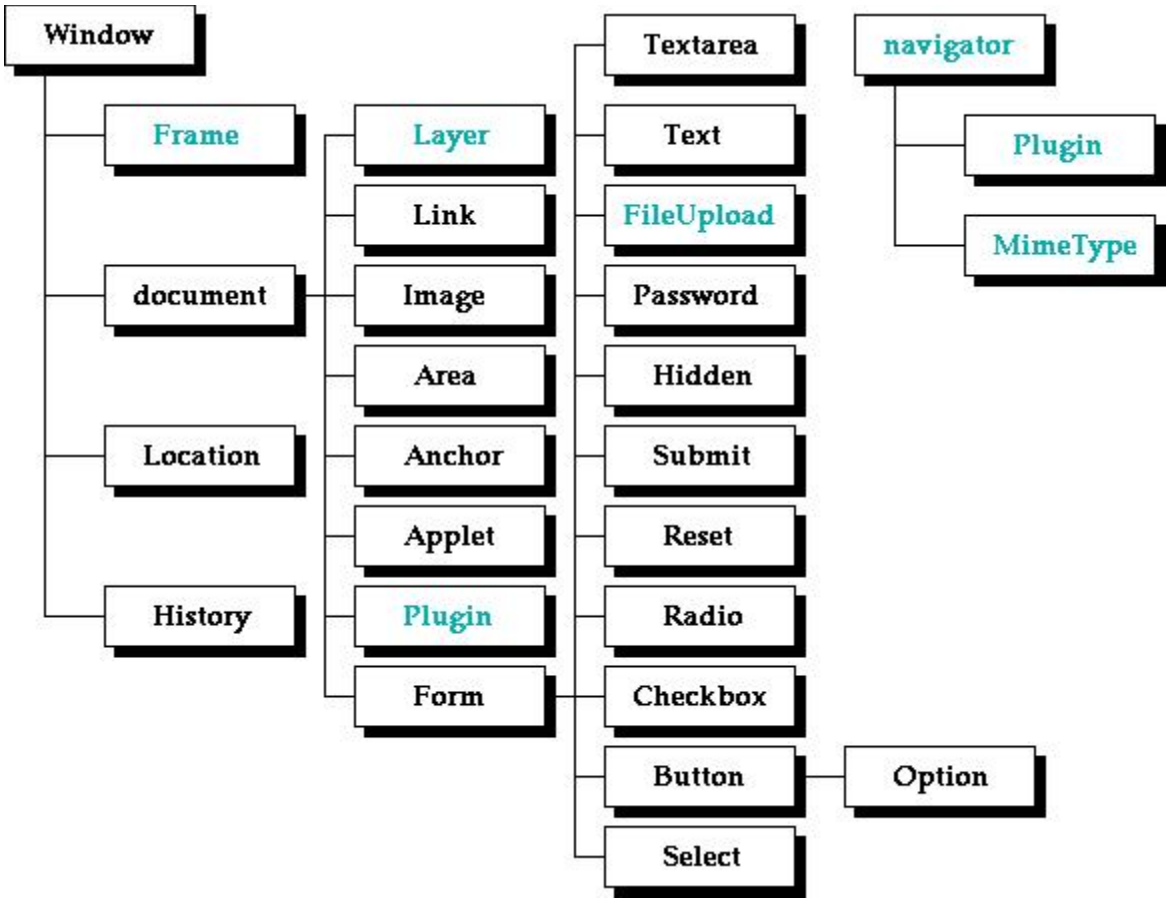
The replace() method returns a modified string where the pattern is replaced.

```
<!DOCTYPE html>
```

```
<html>

<body><h2>JavaScript Regular Expressions</h2>

<p>Search a string for "w3Schools", and display the position of the match:</p>

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

<script>

var str = "Visit W3Schools!";

var n = str.search(/w3Schools/i);

document.getElementById("demo").innerHTML = n;

</script>

</body>

</html>
```

❖ Hierarchy of JavaScript objects

When loading a document in Navigator, it creates a number of JavaScript objects with property values based on the HTML document and other information. They exist in a hierarchy that reflects the structure of the HTML page. This hierarchy is known as instance hierarchy, because it concerns specific instances of objects rather than object classes. The next figure illustrates this hierarchical structure:



❖

What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Live Demo

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function sayHello() {
```

```
            alert("Hello World")
         }
      //-->
   </script>
</head>

<body>
   <p>Click the following button and see result</p>
   <form>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
   </form>
</body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

Live Demo

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
```

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

Event handlers

Imagine an interface where the only way to find out whether a key on the keyboard is being pressed is to read the current state of that key. To be able to react to keypresses, you would have to constantly read the key's state so that you'd catch it before it's released again. It would be dangerous to perform other time-intensive computations since you might miss a keypress.

```
<p>Click this document to activate the handler.</p>
<script>
  window.addEventListener("click", () => {
    console.log("You knocked?");
  });
</script>
```