

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.

JAVASCRIPT – SYNTAX

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

You can place the `<script>` tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the `<head>` tags. The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script.

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

The script tag takes two important attributes:

- **Language:** This attribute specifies what scripting language you are using. Typically, its value will be JavaScript.
- **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

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

Code:

```
<html>
```

```
<head>
```

```
<title> Javascript code-1
```

```
</title>
```

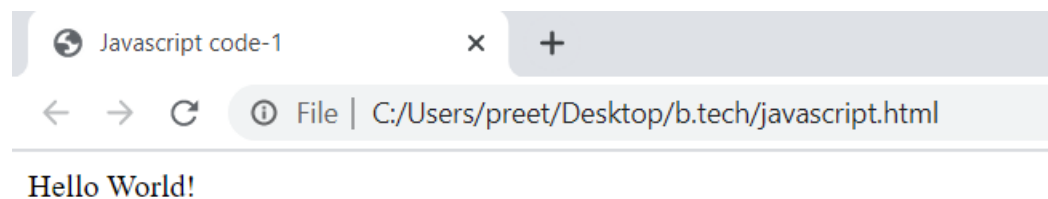
```
</head>
```

```
<body>
```

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

```
document.write ("Hello World!")  
  
</script>  
  
</body>  
  
</html>
```

Output:



Note:

- **JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.** You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.
- **JavaScript is a case-sensitive language.** This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- **Non-JavaScript Browsers** If you have to do something important using JavaScript, then you can display a warning message to the user using `<noscript>` tags. You can add a noscript block immediately after the script block as follows:

Code:

```
<html>

<head>

<title> javascript code-2 </title>

</head>

<body>

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

document.write ("Hello javascrip)

</script>

<noscript>

Sorry...JavaScript is needed to go ahead.

</noscript>

</body>

</html>
```

Now, if then the user's browser does not support JavaScript or JavaScript is not enabled message from </noscript> will be displayed on the screen.

JAVASCRIPT – PLACEMENT

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

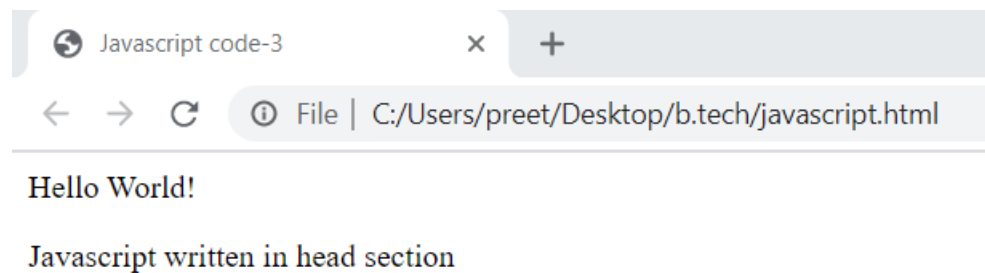
- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

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

Code:

```
<html>
<head>
<title> Javascript code-3
</title>
<script language="javascript" type="text/javascript">
  document.write ("Hello World!")
</script>
</head>
<body>
<p>Javascript written in head section</p>
</body>
</html>
```

Output:



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

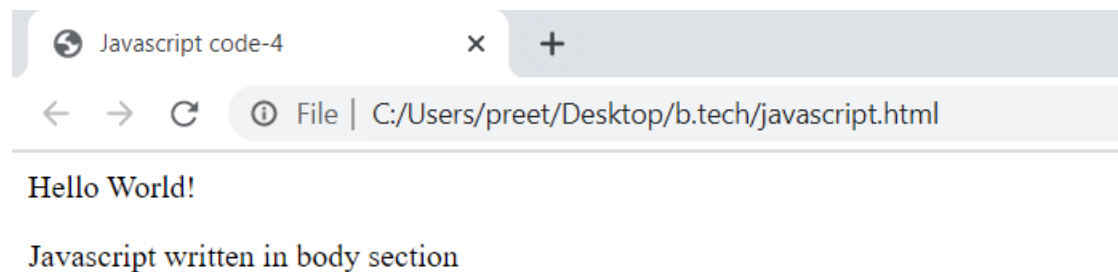
Code:

```
<html>
```

```
<head>
<title> Javascript code-4
</title>

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

Output:



JavaScript in <body> and <head> Sections

Code:

```
<html>
<head>
<title> Javascript code-4
```

```
</title>
<script language="javascript" type="text/javascript">
  document.write ("head section")
</script>
</head>
<body>
<hr></hr>
<script language="javascript" type="text/javascript">
  document.write ("body section")
</script>
</body>
</html>
```

JavaScript in External File- save external file with .js extension

Html Code:

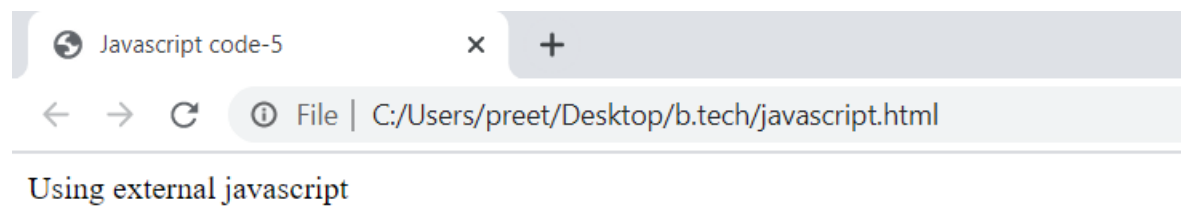
```
<html>
<head>
<title> Javascript code-5
</title>
<script language="javascript" type="text/javascript"
src="C:\Users\preet\Desktop\b.tech\javascript.js">
</script>
</head>
<body>
</body>
```

</html>

JavaScript code:

```
document.write ("Using external javascript")
```

Output:



JAVASCRIPT – datatype

JavaScript allows you to work with three primitive data types:

- Numbers, e.g., 123, 120.50 etc.
- Strings of text, e.g. "This text string" etc.
- Boolean, e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object.

Note: Java does not make a distinction between integer values and floating-point values.

JavaScript Variables

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

```
var money; var name;
```

```
</script>
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript"> var money, 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. For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

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

```
var name = "Ali"; var money; money = 2000.50;
```

```
</script>
```

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

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.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter

with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>

<head>

<title> Javascript code-6

</title>

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


var A="global";          // Declare a global variable
function checkscope( )
{
var A="local";          // Declare a local variable
  document.write(A);
}
</script>

</head>

<body>

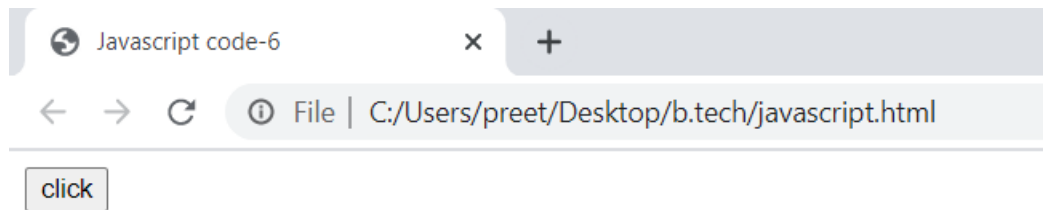
<input type="button" value="click" onclick="checkscope()"/>

</body>

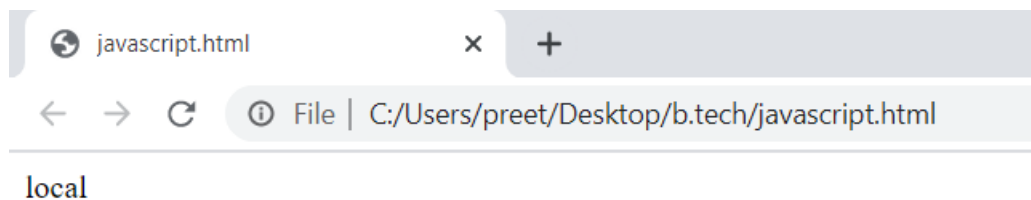
</html>

</script>
```

Output:



After clicking the button



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 Reserved Words A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

JAVASCRIPT – OPERATORS

JavaScript supports the following types of operators.

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

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

Arithmetic operators Code:

```
<html>

<head>

<title> Arithmetic Operators in javascript</title>

</head>

<body>


    <script type = "text/javascript">


        var a = 50;
        var b = 10;
        var c = "java";
        var linebreak = "<br />";
        var result ;


        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 = ");
```

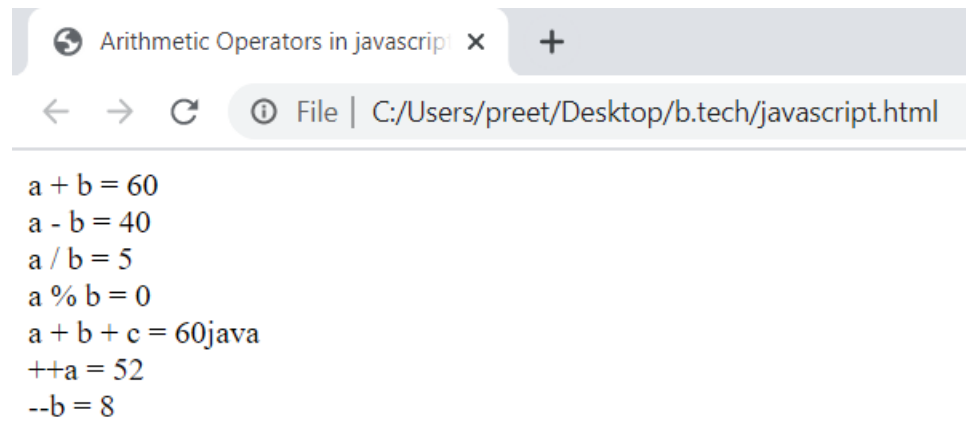
```
result = --b;  
document.write(result);  
document.write(linebreak);
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:



Comparison Operators

- == (Equal)
- != (Not Equal)
- >(Greater than)
- < (Less than)
- >= (Greater than or Equal to)
- <= (Less than or Equal to)

Comparison Operators code:

```
<html>

<head>

<title> Comparision Operators in javascript</title>

</head>

<body>

<script type = "text/javascript">

    var a = 10;

    var b = 50;

    var linebreak = "<br />";

    var result

    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);
```

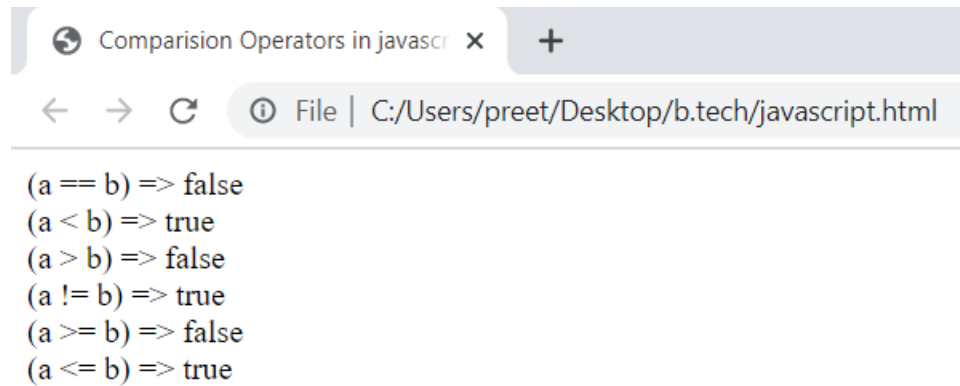
```
document.write("(a <= b) => ");  
result = (a <= b);  
document.write(result);  
document.write(linebreak);
```

```
</script>
```


</body>

</html>

Output:



The screenshot shows a web browser window with a single tab titled "Comparision Operators in javascr". The address bar displays the file path "C:/Users/preet/Desktop/b.tech/javascript.html". The main content area of the browser shows the following JavaScript expressions and their results:

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
```

Logical Operators

- **&& (Logical AND):** If both the operands are non-zero, then the condition becomes true.
- **|| (Logical OR):** If any of the two operands are non-zero, then the condition becomes true.
- **! (Logical NOT):** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Code:

<html>

<head>

<title> Logical Operators in javascript</title>

</head>

<body>

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

```
var a = 10;
```

```
var b = 50;
```

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

```
var result;
```

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

```
result = (a<20 && b<60);
```

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

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

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

```
result = (a<20 || b<40);
```

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

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

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

```
result = (!(a<20 && b<60));
```

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

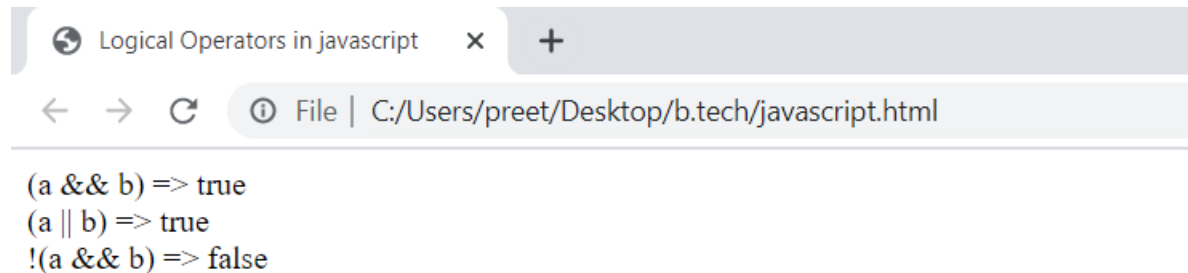
```
document.write(linebreak);
```

```
</script>
```

</body>

</html>

Output:

A screenshot of a web browser window. The title bar shows 'Logical Operators in javascript' with a close button. The address bar shows the file path 'C:/Users/preet/Desktop/b.tech/javascript.html'. The main content area displays three lines of JavaScript code and their results: '(a && b) => true', '(a || b) => true', and '!(a && b) => false'.

```
(a && b) => true
(a || b) => true
!(a && b) => false
```

Bitwise Operators

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

Code:

```
<html>

<head>

<title> Bitwise Operators in javascript</title>

</head>

  <body>


    <script type = "text/javascript">


      var a = 4;          // Bit presentation 100
      var b = 2;          // Bit presentation 10
      var linebreak = "<br />";
      var result;


      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);
```

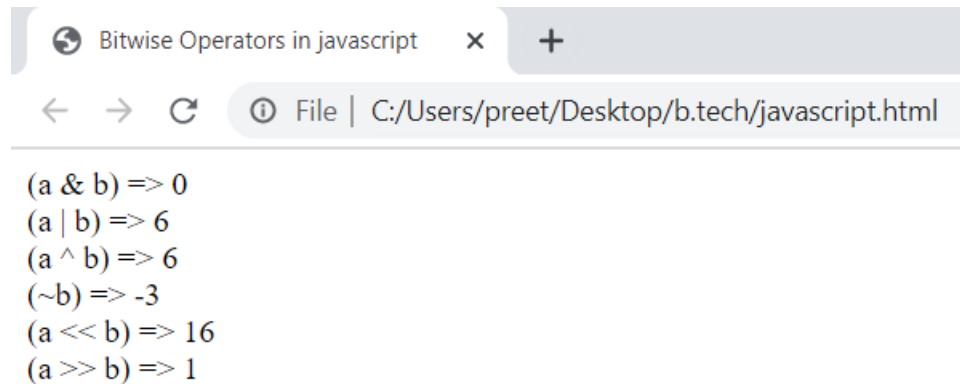
```
document.write("(a >> b) => ");  
result = (a >> b);  
document.write(result);  
document.write(linebreak);
```

```
</script>
```

```
</body>
```

</html>

Output:



The screenshot shows a web browser window with a single tab titled "Bitwise Operators in javascript". The address bar displays the file path "C:/Users/preet/Desktop/b.tech/javascript.html". The main content area of the browser shows the following output:

```
(a & b) => 0
(a | b) => 6
(a ^ b) => 6
(~b) => -3
(a << b) => 16
(a >> b) => 1
```

Assignment Operators

- **= (Simple Assignment)**: Assigns values from the right side operand to the left side operand
- **+= (Add and Assignment)**: It adds the right operand to the left operand and assigns the result to the left operand.
- **-= (Subtract and Assignment)**: It subtracts the right operand from the left operand and assigns the result to the left operand.
- ***= (Multiply and Assignment)**: It multiplies the right operand with the left operand and assigns the result to the left operand.
- **/= (Divide and Assignment)**: It divides the left operand with the right operand and assigns the result to the left operand.
- **%= (Modules and Assignment)**: It takes modulus using two operands and assigns the result to the left operand.

Miscellaneous 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.
- **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.

Code:

```
<html>
<head>
<title> Miscellaneous Operators in javascript</title>
</head>
<body>

    var a = 10;
    var b = 50;
    var linebreak = "<br />";
    var c = "String";
    var result;

    document.write ("((a > b) ? true : false) => ");
    result = (a > b) ? true : false;
```

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

```
document.write ("((a < b) ? true : false) => ");  
result = (a < b) ? true : false;  
document.write(result);  
document.write(linebreak);
```

```
result = (typeof c == "string" ? "C is String" : "C is Numeric");  
document.write("Result => ");  
document.write(result);  
document.write(linebreak);
```

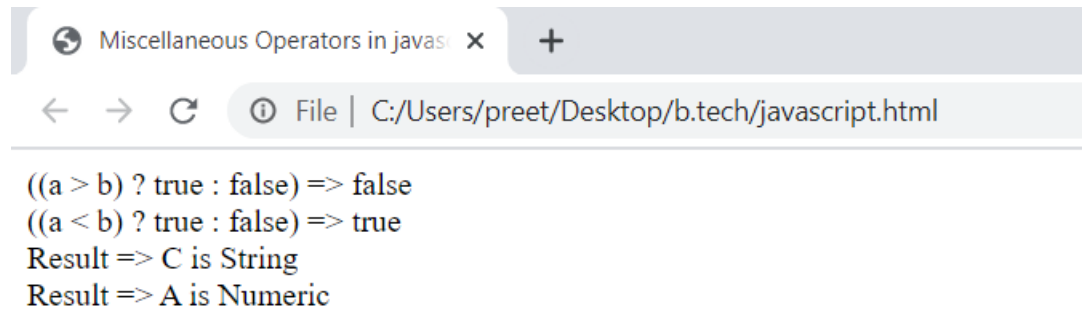
```
result = (typeof a == "string" ? "A is String" : "A is Numeric");  
document.write("Result => ");  
document.write(result);  
document.write(linebreak);
```

```
</script>
```

```
</body>
```

```
</html>
```


Output:



The screenshot shows a web browser window with a single tab titled "Miscellaneous Operators in javas...". The address bar displays the file path "C:/Users/preet/Desktop/b.tech/javascript.html". The browser content area shows the following text:

```
((a > b) ? true : false) => false
((a < b) ? true : false) => true
Result => C is String
Result => A is Numeric
```

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 multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.