JAVASCRIPT

JAVASCRIPT

Introduction

- JavaScript is a compact, object-based scripting language for developing client Internet applications.
- JavaScript was designed to add interactivity to HTML pages.
- JavaScript is a scripting language - a scripting language is a lightweight programming language.
- A JavaScript is lines of executable computer code.
- A JavaScript is usually embedded directly in HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation).
- Everyone can use JavaScript without purchasing a license.
- All major browsers, like Netscape and Internet Explorer, support JavaScript.
- Javascript was developed by Netscape as Live Script - changed to JavaScript when endorsed by Sun 1993, version 1.0 released with Netscape 2.0.
- JavaScript is a powerful scripting language that is also capable of performing extensive form data collecting and form processing functions.

Comparing Javascript with Java

- While comparing javascript with java first lets see the basic differences between them

Javascript	Java
1. Javascript is a small, lightweight programming language.	1. Java is a full-blown powerful, sophisticated programming language.
2. Developed by Netscape communications.	2. Developed by Sun Microsystems.
3. Scripting language interpreted at runtime.	3. True programming compiled to byte-code.
4. Object-based, has limited number of built-in objects.	4. Object-oriented, can create their own classes.
5. Not fully extensible.	5. Fully extensible.
6. Code integrated with, and embedded in HTML.	6. Applies distinct from HTML (accessed from HTML pages).

7. Variables type must not be declared.	7. Variables type must be declared.
8. Javascript source code can be viewed by everybody using "view source command".	8. Your Java source is hidden because it's only the compiled byte-code, which the browser uses, but this is not a guarantee of security.

- Now lets see the similarities between java and javascript:
    1. Both can be used for enhancing the capabilities of the web pages.
    2. Both can run on the client machine - i.e. the machine where you have your browser, not the server where the page came from.
    3. Both have some level of security built in to guard against malicious use. No computer system is ever 100% secure unless it's isolated in a locked room surrounded by armed guards, but the developers of Java and JavaScript have taken some care in their security.

JavaScript Uses

- The most frequent uses of javascript are:
    - For displaying the clock
    - Used for creating Drop-down menus.
    - Showing Alert messages.
    - Displaying Popup windows.
    - Validating HTML Form Data.

Advantages of Javascript

- Cross Browser supporting:
    - This means that the javascript codes are supported by various browsers like Internet Explorer, Netscape Navigator, Mozilla etc.
- Platform Independent:
    - Javascript codes can be executed in any platform like Linux, Microsoft Windows and many other operating systems.
- Lightweight for fast downloading:

- The javascript codes runs directly in the client machine. The code should be downloaded all the way from server to the client and this time duration is very minimum and the executing the codes of javascript is also very fast.

- Validating Data:
  - The data can be validated in the two different way:
    - Validating in server side or in server machine.
    - Validating in client side or in client machine.
  - In this two different types of validation of data the second one is much more faster, and this is done through javascript.
- Sophisticated User Interfaces:
  - By using javascript you can create a user interactive interfaces that can communicate with the user and the Browser.
- In-Built software:
  - To you don't need any extra tools to write JavaScript, any plain text or HTML editor will do, so there's no expensive development software to buy.
- Easy to Learn:
  - The javascript programmer should know the minimal syntax of javascript since it supports many syntax and data types as C and C++.
  - It's also an easy language to learn, and there's a thriving and supportive online community of JavaScript developers and information resources.
- Designed for programming User-Events:
  - Javascript supports Object/Event based programming. So the code written in javascript can easily be break down into sub-modules.

Disadvantages of Javascript:

- Launching an application on the client computer.
  - Javascript is not used to create stand-alone application; it is only used to add some functionality in any web page.

- Reading or writing files:

  - Javascript cannot read and write files into the client machines. It can only be used as a utility language to develop any web site.

- Retrieving the text contents of HTML pages or files from the server.
- Reading and Writing files to the server:
  - Javascript can read and write to any file in the server as well.
- Sending secret e-mails from Web site visitors to you:
  - Javascript cannot be used to send email to the visitors or user of the web site. This can be done only with the server side scripting.

- Cannot create Database Application:
  - By using javascript you cannot connect the web site to the database. For this you need to use server-side scripting.
- Browser Compatibility Issues:
  - Not all browsers support the javascript codes. The browser may support javascript as a whole, but may not support the codes or lines of codes written in javascript and may interpret differently.
- Javascript does not implement multiprocessing or multithreading.
- Use printers or other devices on the user's system or the client-side LAN.
- Javascript has limitations of writing in a client machine. It can only write the cookie in client machine that is also of a certain size i.e. 4K.

Adding Javascript codes

- You have to focus on three major steps while adding javascript codes in HTML document.
  1. Use a <script> tag to tell the browser that you are using javascript.

     Example:

     <script language = "Javascript">
  2. Write the javascript codes.

     Example:

     <script language = "Javascript">

     // javascript codes HERE

```
</script>
```

3. Test the scripts.

- While writing the javascript codes there can be many types of errors like:
  - Human Errors.
  - Browser compatibility issues.
  - Incorrect behavior based on the different operating systems.
- So when using javascript, be sure that you test your scripts out on a wide variety of systems and setups.
- You can add javascript codes between the <head> tag as well as in the <body> tag. And you can have multiple <script> tag in a single web page, but you must have the closing </script> tag of each opening tag.

First Example in Javascript

```
<Script language = "javascript">
        <!- -
                document.write ("Hello World");
        // - - >
</script>
```

- The first line <script language = "javascript"> tells the browser that the code written between the <script> and </script> tags are the javascript codes.
- Some browsers doesn't support scripts, and may display the scripts as the HTML content in the web-page.
- To prevent this you must write the javascript codes in between <! - - and // - ->.
- The two forward slashes at the end of the line are the javascript comments.
- You cannot put this slashes to the starting of the comment line like (// <!- -), because the older browser will display it.
- Now the above code will display Hello World in the browser.

Using External Javascript Files

- While using external javascript files in a web-page, the javascript files must have the three main features:
  - First, the file that you are importing must be a valid javascript file.
  - Second, the file must have the extension ".js".
  - Lastly, you must know the location of the file.
- Here is the example of using external javascript files in a web-page.

```
<HTML>
        <HEAD>
            <Script src = "myExample.js"></script>
            <TILTE></TITLE>
        </HEAD>
            <BODY>
            </BODY>
<HTML>
```

- Here myExample.js contains the javascript codes and whatever is written in the file will be displayed in the browser.
- In this condition the myExample.js file and the above HTML file must be in the same location.
- For implementing the external javascript file you must have at least two files: one for HTML codes and other for javascript file.

Points to Ponder in Javascript

- Optional semicolon (;) to end the statement.
- Some browser doesn't support javascript codes, and display the script as the content of the web-page.
- To prevent this we have to use the HTML comments.
- To use JavaScript functions the code needs to follow the line:
  <SCRIPT language = "JavaScript"> and closed with </SCRIPT>.
- JavaScript uses both // and /* */ to indicate comments.

- JavaScript supports data types: numeric, string, Boolean, and null.
- JavaScript contains three user interfaces: alert (message), prompt (message, [Default]), confirm(message).
- JavaScript does not have built-in support for arrays, but does allow for building of arrays.
- JavaScript supports operators: +, -, *, /, %, ++, --, =, +=, -=, *=, /=, %=, &&, ||, !, &, |, ^.
- JavaScript is case sensitive.
- It's a Object-based language.

Case Sensitivity

- Javascript is a case-sensitive language.
- This means the Upper and Lower case of any variables or methods may effect.
- The variable in Uppercase is different then the variable declared in Lowercase.
- For Example:

  myVar, myvar, MyVar and MYVAR are four different types of variables.
- The same rules occur in the name of function as well as in objects of javascript.

Optional Semicolons

- The semicolon (;) identifies the end of the statement in javascript.
- But it is not always necessary to put semicolon at the end of statement.
- If you are writing two different statements in two different lines without semicolon than it is valid in javascript.
- But if you are writing the two different statements in same single line then you have to keep semicolon after the end of the first statement.
- Examples:

  var a=5

  var b=2

  The above statement are valid in javascript since javascript has optional semicolon to end the statement.

var a=5; var b=5

This is also valid in javascript

var a=5;

var b=2;

These two statements are also valid in javascript.

Literals

- You use literals to represent values in JavaScript.
- These are fixed values, not variables that you literally provide in your script.
- Javascript supports the following types of literals:
  1. Array Literals
     - An array literal is a list of zero or more expressions, each of which represents an array element, enclosed in square brackets ([]).
     - When you create an array using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified.
     - The following example creates the coffees array with three elements and a length of three:
       coffees = ["French Roast", "Columbian", "Kona"]
     - If an array is created using a literal in a top-level script, JavaScript interprets the array each time it evaluates the expression containing the array literal.
     - In addition, a literal used in a function is created each time the function is called.
  2. Integers Literals
     - Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8).
     - A decimal integer literal consists of a sequence of digits without a leading 0 (zero).
     - A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal.
     - Hexadecimal integers can include digits (0-9) and the letters a-f and A-F.
     - Octal integers can include only the digits 0-7.
     - Some examples of integer literals are: 42, 0xFFF, and –345.

3. Object Literals
    - An object literal is a list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}).
    - You should not use an object literal at the beginning of a statement.
    - This will lead to an error or not behave as you expect, because the { will be interpreted as the beginning of a block.
4. String Literals
    - A string literal is zero or more characters enclosed in double (") or single (') quotation marks.
    - A string must be delimited by quotation marks of the same type; that is, either both single quotation marks or both double quotation marks.
    - The following are examples of string literals:

      "blah"

      'blah'
    - You should use string literals unless you specifically need to use a String object.

Reserved Words

- Javascript have some reserved words that cannot be used as variable, objects or a functions.
- These words are used for some reserved purpose in javascript.
- The lists of reserved words are listed below:

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	
else	enum	export	extends
false	final	finally	float
for	function	goto	if
implements	import	in	instanceof
int	interface	long	native

new	null	package	private
protected	public	return	short
static	super	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	

Data types

- A computer works with values. It manipulates values and produces some kind of result.
- Depending on the program, it may be limited as to what types of values it can work with.
- The types of values a program can work with are known as its data types.
- JavaScript supports many types of data.
- Data types can be broken into two broad categories:
    1. Primitive Data types
        - A primitive data type is a data type that stores a single value, normally a literal of some sort, such as numbers and strings.
    2. Composite Data types
        - A composite data type, for the purposes of JavaScript, is the same thing as an object.
        - It is a data type that can consist of multiple values grouped together in some way.
        - JavaScript treats objects as associative arrays.
        - An associative array is an array that can have its elements associated with names in addition to their numeric index position in the array.

Primitive Data types

- There are three primitive data type, javascript provides to us:
    - Numbers
        - Numbers are the easist of the data types to understand.
        - They represent numeric values.
        - The simplest type of number is an integer.

- It is a number with no fractional component.
- JavaScript can handle integer values between 2^{53} and -2^{53}. Some JavaScript functions, however, cannot handle extremely large or small numbers, so it is probably a good idea to try to keep your numbers in the range of 2^{31} and -2^{31}.
- Another commonly used type of number is a floating-point number.
- A floating-point number is a number that is assumed to have a decimal point.
- Being a computing language, JavaScript also has the ability to handle hexadecimal (four bit) and octal (three bit) numbers.

- Strings
  - A string is sequence of valid characters within a given character set.
  - It is normally used to represent text. A string literal is defined by enclosing it in matching single or double quotes.
  - If you create a string with nothing in it, it is called an empty string and is normally created by two quote marks with nothing between them.
  - Some characters that you may want in a string may not exist on the keyboard, or may be special characters that can't appear as themselves in a string.
  - In order to put these characters in a string, you need to use an escape sequence to represent the character. An escape sequence is a character or numeric value representing a character that is preceeded by a backslash (\ ) to indicate that it is a special character.
  - Some escaped characters are as follows:

Escape Sequence	Character
\0	(blackslash zero) The Null character (\u0000). If you've taken COBOL, you may recognize this as COBOL's low-value, used to populate fields with a clearly defined invalid value that tests less than anything else.
\b	Backspace.
\t	Tab. Tabs behave erratically on the Web and are best avoided, but sometimes you need them.
\n	New line (\u0000a). Inserts a line break at the point specified. It is a

	combination of the carriage return (\r) and the form feed (\f).
\"	Double quote.
\'	Single quote, or an apostrophe, such as in can\'t.
\\	The backslash, since by itself it is a command.
\x99	A two digit number specifying the hexadecimal value of a character in the Latin-1 character set (ASCII).
\u9999	A four digit hexadecimal number specifying a character in the Unicode character set. This is not supported before JavaScript 1.3.

- Boolean values
  - There are two boolean values, true and false.
  - These are normally the result of a logical comparison in your code that returns a true/false or yes/no result, such as:
    a = = b
  - This statment reads: does the value of variable a equal the value of variable b?
  - If you need to use boolean values in computations, JavaScript will automatically convert true to 1 and false to 0.
  - The conversion can also work the other way.
  - When testing for the result of a comparison, JavaScript will treat any non-zero value as true, and a zero value as false.
  - JavaScript will also test to false when you test for the existence of something that does not exist.

Composite Data types

- All composite data types can be treated as objects, but we normally categorize them by their purpose as a data type.
- For composite data types we will look at objects, including some special pre-defined objects that JavaScript provides, as well as functions and arrays.
  - Objects:
    - An object is a collection of named values, called the properties of that object.

- Functions associated with an object are referred to as the methods of that object.
- Properties and methods of objects are referred to with a dot.notation that starts with the name of the object and ends with the name of the property.
- For instance:

  image.src.
- Objects in JavaScript can be treated as associative arrays.
- This means that image.src and image['src'] are equivalent.
- JavaScript has many predefined objects, such as a Date object and a Math object.
- These are used much as function libraries are used in a language like C.

- Functions:
  - A function is a piece of code, predefined or written by the person creating the JavaScript, that is executed based on a call to it by name.
  - Any JavaScript that is not inside a function (or is not associated with some even attribute or hyperlink attribute) is executed the moment the Web browser reaches it when first parsing the document.
  - Functions allow execution to be delayed.
  - They also allow for the same piece of code to be re-used many times in the same document, since functions allow the section of code they contain to be referred to by name.
  - A function is a data type in JavaScript.
  - This is different from many other programming languages.
  - This means that they can be treated as containing values that can be changed.
- Arrays:
  - An Array is an ordered collection of data values.
  - In some programming languages, arrays have very specific limitations.
  - In JavaScript, an array is just an object that has an index to refer to its contents.
  - In other words, the fields in the array are numbered, and you can refer to the number position of the field.
  - The array index is included in square brackets immediately after the array name.

- In JavaScript, the array index starts with zero, so the first element in an array would be arrayName[0], and the third would be arrayName[2].
- JavaScript does not have multi-dimensional arrays, but you can nest them, which is to say, an array element can contain another array.
- You access them listing the array numbers starting for the outmost array and working inward.
- Therefore, the third element (position 2) of or inside the ninth element (position 8) would be arrayName[8][2].

Variable - Introduction

- Variable is a name that can be used to store values.
- It is a name assigned to a location in computer's memory to store data.
- These variables can take different values but one at a time and the value can be changed during the execution of program.
- Variable names may consist of uppercase character, lowercase character and underscore.
- Rules to giving the name of variable are as:
    1. First character should be a letter or underscore.
    2. The variable name cannot be a keyword.
    3. Uppercase and lowercase letters are significant for example code, Code, CODE are three different variables. But variables are in lowercase.
- Note: The Netscape supports the ($) sign as the first character in variable but the Internet Explorer does not supports.

Typing

- Javascript is a untyped language.
- This means you can use variables directly where you want to use it.
- But in javascript the variables are declared with the var keyword.
- This keyword declares all types of variables and you can use a same variable as string, as Integer, as Number and any type of Objects.

- For Example:

  var a=”Mr. ABC”;

  a=12345;

- The both statements are valid in javascript, but in C and C++ the statements are not valid.
- The both statements are valid due to javascript is untyped language.

Local and Global Variables

- Javascript supports both local as well as global variables.

  Local Variables:

  - The variables which are defined within a body of the function or block is local to that function or block only is called local variable.
  - They have no presence outside the function.
  - The values of such Local variables cannot be changed by the main code or other functions.
  - Example:

```
<script language=”javascript”>
        function testLocal()
        {
                var a =5;
                alert(“The local value of a is: ” + a);
        }
</script>
```

  Global Variables:

  - The variables that are declared outside the function and is used inside the function is called global variables.
  - The global variable has the same data type and same name throughout the program.
  - It is useful to declare the variable global when the variable has constant value throughout the program.
  - These are the variables that can be used throughout the scripts and the value of which can be changed.
  - Example:

```
<script language="javascript">
    Var a=10;
    function testGlobal_1()
    {
        alert("The local value of a is: " + a);
        //displays the value of a as 10
    }
    function testGlobal_2()
    {
        alert("The Global value of a is:" + a);
        //displays the value of a as 10
    }
</script>
```

- Here the value of a is global and is accessed within both functions testGlobal_1 and testGlobal_2.

What is variable scoping?

- Local variables exist only inside a particular function hence they have Local Scope.
- Global variables on the other hand are present throughout the script and their values can be accessed by any function. Thus, they have Global Scope.

Constants

- You can create a read-only, named constant with the const keyword.
- The syntax of a constant identifier is the same as for a variable identifier: it must start with a letter or underscore and can contain alphabetic, numeric, or underscore characters.
- Example:

  const prefix = '212';
- A constant cannot change value through assignment or be re-declared while the script is running.

- The scope rules for constants are the same as those for variables, except that the const keyword is always required, even for global constants.
- If the keyword is omitted, the identifier is assumed to represent a var.
- You cannot declare a constant at the same scope as a function or variable with the same name as the function or variable.
- For example:

```
//THIS WILL CAUSE AN ERROR
    function f{};
    const f = 5;
//THIS WILL CAUSE AN ERROR ALSO
    function f
    {
    const g=5;
    var g;
    }
```

Garbage Collection

- is a way of reclaiming the memory occupied by objects that are no longer in use.
- In C and C++, garbage collection is manual--the programmer explicitly decides when to free memory for reuse.
- In Java, on the other hand, garbage collection is handled automatically--the system can detect when objects are no longer in use and free them appropriately.
- JavaScript also supports automatic garbage collection.
- In Internet Explorer 3.0, garbage collection is implemented in a technically sound way and you don't have to understand any of its details.
- It is enough to know that when your objects are no longer in use, the memory they occupy will automatically be reclaimed by the system.
- Navigator 4.0 will also have a perfectly transparent garbage collection scheme like this.
- Unfortunately, garbage collection in earlier versions of Navigator is less than perfect.
- In Navigator 3.0, it is pretty good, but requires you to be aware of a couple of issues.

- In Navigator 2.0, garbage collection is seriously flawed, and you must take a number of steps to avoid crashing the browser.
- [Note: Study self if you want to more about Garbage Collection]

Expression

- An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value:
  - the value can be a number,
  - a string, or
  - a logical value.
- Conceptually, there are two types of expressions:
  - those that assign a value to a variable, and
  - those that simply have a value.
- For example, the expression x = 7 is an expression that assigns x the value seven.
- This expression itself evaluates to seven. Such expressions use assignment operators.
- On the other hand, the expression 3 + 4 simply evaluates to seven; it does not perform an assignment.
- The operators used in such expressions are referred to simply as operators.
- JavaScript has the following types of expressions:
  1. Arithmetic: evaluates to a number, for example 3.14159.
  2. String: evaluates to a character string, for example, "Fred" or "234".
  3. Logical: evaluates to true or false.
  4. Object: evaluates to an object.

Operators

- Javascript has different types of operator, which can be classified by several criteria, such as:
  1. Number of Operands
     - In this type of category, javascript supports 3 basic types of Operators:
       - Unary
         - Converts a single expression into a single more complex expression.
         - A unary operator requires a single operand, either before or after the operator.
         - For Example:

           i--;

```
++i;
```

- Binary
  - Combine two expressions into a single, more complex expression.
  - A binary operator requires two operands, one before the operator and one after the operator.
  - For Example:

```
var x = 13 + 14
```

- Ternary
  - Also known as conditional operator.
  - The conditional operator is the only JavaScript operator that takes three operands.
  - The operator can have one of two values based on a condition.
  - The syntax is:

```
condition ? val_1 : val_2
```

  - If condition is true, the operator has the value of val_1. Otherwise it has the value of val_2.
  - You can use the conditional operator anywhere you would use a standard operator.
  - For example:

```
status = (age >= 18) ? "adult" : "minor"
```

  - This statement assigns the value "adult" to the variable status if age is 18 or more.
  - Otherwise, it assigns the value "minor" to status.

2. Number of Operation.
   - Javascript supports many types of operators according to this classification.
   - They can be categorized as:
     - Comparison Operators
       - A comparison operator compares its operands and returns a logical value based on whether the comparison is true.
       - The operands can be numerical, string, logical, or object values.
       - Strings are compared based on standard lexicographical ordering, using Unicode values.
       - The following table describes the comparison operators.

Operator	Description	Example
Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type.	3 === var1
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	var1 !== "3" 3 !== '3'
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2 var2 <= 5

- Arithmetic Operators
  - Arithmetic operators take numerical values as their operands and return a single numerical value.
  - The standard arithmetic operators are addition(+), subtraction(-), multiplication(*), and division(/).
  - These operators work as they do in most other programming languages, except the / operator returns a floating-point division in JavaScript.
  - In addition, JavaScript provides the arithmetic operators listed in the following table.

Operator	Description	Example
%(Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++(Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ sets x to 4 and returns 3.
--(Decrement)	Unary operator. Subtracts one to its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x++ sets x to 2 and returns 3.
-(Unary negation)	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.

- Assignment Operators
  - An assignment operator assigns a value to its left operand based on the value of its right operand.
  - The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand.
  - That is, x = y assigns the value of y to x.
  - The other assignment operators are shorthand for standard operations, as shown in the following table.

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

- Bitwise Operators
  - Bitwise operators treat their operands as a set of 32 bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers.
  - For example, the decimal number nine has a binary representation of 1001.
  - Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.
  - The following table summarizes JavaScript's bitwise operators.

Operator	Usage	Description
Bitwise AND	a & b	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	a | b	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	a ^ b	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	~ a	Inverts the bits of its operand.
Left shift	a << b	Shifts a in binary representation b bits to left, shifting in zeros from the right.
Sign-propagating right shift	a >> b	Shifts a in binary representation b bits to right, discarding bits shifted off.
Zero-fill right	a >>> b	Shifts a in binary representation b bits to the right, discarding bits

shift		shifted off, and shifting in zeros from the left.

➢ Logical Operators

- Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value.
- However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.
- The logical operators are described in the following table.

Operator	Usage	Description
&&	expr1 && expr2	(Logical AND) Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	expr1 || expr2	(Logical OR) Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	!expr	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

➢ Special Operators

- Javascript provides the following Special Operators:
  1. Comma Operator:
     - The comma operator (,) simply evaluates both of its operands and returns the value of the second operand.
     - This operator is primarily used inside a for loop, to allow multiple variables to be updated each time through the loop.

- For example, if a is a 2-dimensional array with 10 elements on a side, the following code uses the comma operator to increment two variables at once.
- The code prints the values of the diagonal elements in the array:

```
for(i=0,j=9;i<=9;i++,j--)
document.write("a["+i+","+j+"]= " + a[i*10 +j])
```

- [Note: Two-dimensional arrays are not yet supported.]
- This example emulates a two-dimensional array using a one-dimensional array.

2. delete
   - The delete operator deletes an object, an object's property, or an element at a specified index in an array.
   - The syntax is:
   - You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.
   - If the delete operator succeeds, it sets the property or element to undefined.
   - The delete operator returns true if the operation is possible; it returns false if the operation is not possible.
   - Example:

```
x=42
var y= 43
delete x //returns true (can delete if declared implicitly)
delete y  // returns false (cannot delete if declared with var)
```

3. instanceof
   - The instanceof operator returns true if the specified object is of the specified object type.
   - The syntax is:

```
objectName instanceof objectType
```

   where objectName is the name of the object to compare to objectType, and objectType is an object type, such as Date or Array.

- Use instanceof when you need to confirm the type of an object at runtime.
- For example, the following code uses instanceof to determine whether theDay is a Date object.
- Because theDay is a Date object, the statements in the if statement execute.

```
TheDay = new Date(1995, 12, 17)
    if (TheDay instanceof Date)
    {
      // statements to execute
    }
```

4. new
   - You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.
   - On the server, you can also use it with DbPool, Lock, File, or SendMail.
   - The syntax for new operator is as follow:

```
objectName = new objectType ( param1...[,paramN] )
```

5. this
   - Use the this keyword to refer to the current object.
   - In general, this refers to the calling object in a method.
   - The syntax of this is as follows:

```
this[.propertyName]
```

6. typeof
   - The typeof operator is used in either of the following ways:
     1. typeof operand
     2. typeof (operand)
   - The typeof operator returns a string indicating the type of the unevaluated operand.

- Operand is the string, variable, keyword, or object for which the type is to be returned.
- The parentheses are optional.
- Suppose you define the following variables:

  var myFun = new Function("5+2")

  var shape="round"
- The typeof operator returns the following results for these variables:

  typeof myFun is function

  typeof shape is string

JavaScript Functions

- A function is really nothing more than a named block of code.
- It is a statement block that has been assigned a name.
- Functions are small groups of instructions that are carried out when they are called upon.
- It is a reusable code-block that will be executed by an event, or when the function is called.
- A function typically contains a set of commands for a specific purpose, which you want to run at a certain time.
- Each function in a script is given a unique name.
- The layout of a function always follows the same format, including these three things:
  - The word "function"
    - Function tells the browser "Do these instructions when this name is called upon.".
  - The function's name, followed by parentheses (which may or may not be empty)
    - Function's name must be unique and every function name is contains the parentheses, which may be empty if the function does not take any parameter.
  - Curly braces containing the function's code.
    - The curly brackets ({ })are used to contain the set of instructions.
- A script can contain any number of functions, according to the terms and conditions.

Defining Functions:

- The functions in javascript is defined by using the function keyword followed by function-name and the parentheses.
- <u>The Syntax of defining the functions:</u>

```
function function-Name(Argument lists.....)
{
        /*
                some lines of codes
                return;
        */
```

```
}
```

- The function here is a keyword and should be explictly written in-front of the function name.
- The function is followed by the function name, which should be unique and should follow all the naming conventions that any function name should follow.
- The parentheses should be attached with every function name and in-between the parantheses you should lists the parameters if the function contains any parameter.
- If not then you must include an empty parentheses, but you should include parantheses.
- Then every function contains some lines of codes and these codes are included within the curly braces.
- If the function returns any value then it is returned with the keyword return from the function.

Invoking Functions

- Functions do not run automatically. When the page loads, each function waits quietly until it is told to run.
- To run a function you must call it.
- This means sending an instruction to the function telling it to run.
- There are two common ways to call a function:
  - From an event handler and
  - From another function.
  - Calling Functions from Event-handlers
    - An event handler is a command, which calls a function when an event happens, such as the user clicking a button.
    - The command is written in the format onEvent, where Event is the name for a specific action.
    - Here are some common examples:
    - User clicks a button (onClick):

```
<input type="button" value="Click Here" onClick="doSomething();">
```

    - User places their cursor in a text field (onFocus):

```
<input type="text" onFocus="doSomething();">
```

- Calling Functions from another Functions
  - Functions can also call other functions. Simply enter the name of the function to be called, with its parentheses.
  - Example:

```
function doSomething()
{
        doSomethingElse(); //this line calls the another function
}
function doSomethingElse()
{
        //the function codes go here
}
```

The Function constructor

- JavaScript has a Function constructor which allows you create a variable and assign it a function as a value.
- Example:

```
var q = new Function('a', 'b', 'return a / b;');
```

- The Function constructor takes any number of arguments, each of which must be defined as a string, and the last of which must be the body of the code for the function.
- A function defined this way can still be invoked by the variable named, res = q(20, 5).
- But technically it has no name. Instead the function name is a variable that contains a reference to that function as a value.
- Such function are referred to as anonymous functions, since there is no way to refer to them by name.
- The benefit of declaring a function this way is that it is reparsed each time it is run, meaning, you could replace the strings that define the arguments and function body with variables, dynamically assign values to these variables, and thus rewrite the function each time it ran.
- Since the function is reparsed each time, it can really slow down processing if it is run repeatedly.

Function Properties

- As an object, a function has other properties as well.
- One is the length property, which specifies how many arguments the function is expecting.
- As with objects, you can also define properties for functions.
- This allows you to create values that persist across repeated calls to the function.
- To create a new property, you can just assign it a value.
- Since the function declaration is parsed before the code in the script is executed, we can put the initialization statement outside of the function.
- Example:

```
q.counter = 0;
function q (x)
{
        q.counter ++;
        //statements of the function
}
```

- Each time function q runs, it will increment the property q.counter by one.