JavaScript Introduction

**What is JavaScript?**

* JavaScript is an **object-based scripting language** that is **lightweight** (is one that is designed to have very small [memory footprint](https://en.wikipedia.org/wiki/Memory_footprint), is easy to implement (important when [porting](https://en.wikipedia.org/wiki/Porting) a language), and/or has [minimalist](https://en.wikipedia.org/wiki/Minimalism_(computing)) syntax and features ) and **cross-platform** (able to be used on different types of computers or with different software packages).
* JavaScript is the programming language of HTML and the Web.
* JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

**Where JavaScript is used?**

JavaScript is used to create interactive websites. It is mainly used for:

* Client-side validation.
* Dynamic drop-down menus.
* Displaying Date and Time.
* Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box).
* Displaying Clock etc.

**Advantages of JavaScript**

* **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.

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

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

JavaScript Example

<html>

<body>

<h2>Welcome to Neo Rays Software Solutions Pvt. Ltd. </h2>

<script>

document.write("Hello this is JavaScript");

</script>

</body>

</html>

## JavaScript Example using alert

<html>

<body>

<h2>Welcome to Neo Rays Software Solutions Pvt. Ltd. </h2>

<script>

alert("Hello this is JavaScript"); //to display output in pop up

</script>

</body>

</html>

## JavaScript Language

1. Character Set
2. Keywords
3. Identifiers
4. Variable
5. Data Types
6. Operators
7. **Character Set:**

* To display an HTML page correctly, the browser must know what character set (character encoding) to use.
* It is list of characters which developer can use for writing JavaScript code. Character set allows the following 3 types.

a) Numbers - 0 to 9.

b) Alphabets – A to Z, A to z

c) Special Characters - #, $, etc

* It uses **UTF-8** (8-bit Unicode Transformation Format by default) for Internationalization (support all the langauge)

1. **Keywords :**

In JavaScript you cannot use these reserved words as variables, labels, or function nam

****

1. **Identifiers :**

JavaScript Identifiers are names; names that you give things in JavaScript. These JavaScript “things” include

* variable
* functions
* objects
* properties
* methods
* events

Like much of JavaScript, there are rules to be followed.

* Identifiers can only contain letters, numbers, underscore (\_) and the dollar sign ($)
* Identifiers cannot start with a number
* Identifiers are case-sensitive
* Identifiers can be any length
* Identifiers cannot be the same as JavaScript reserved words
* Don’t use global properties and methods as identifiers (more later)
* Don’t use words similar to reserved words

When naming an identifier with two words in it, it’s a best practice to use camel case. With this convention, the first letter of each word, excluding the first word, is uppercase

Example:

first**N**ame

my**C**ommon**V**ariable**N**ame

Here are some examples of valid identifier naming conventions:

* firstname
* totalPrice
* cust\_1
* click\_calculate
* $
* $total

1. **Variables :**

* Variable is a container used for temporary storage of data. The variables that are used in the program have to be declared before its use.
* All JavaScript **variables** must be **identified** with **unique names**.
* These unique names are called **identifiers**.
* Identifiers can be short names (like x and y), or more descriptive names (age, sum, totalVolume).
* There are two types of variables in JavaScript: local variable and global variable.

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter
* Names can also begin with $ and \_
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

## Correct JavaScript variables:

var x = 10;

var \_value="NeoRays";

## Incorrect JavaScript variables

var 123=30;

var \*aa=320;

Example of JavaScript variable

**<script>**

**var x = 10;**

**var y = 20;**

**var z=x+y;**

**document.write(z);**

**</script>**

* **JavaScript local variable:**

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

<script>

function abc()

{

var x=10; //local variable

}

</script>

* **JavaScript global variable:**

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

<script>

var data=200; //global variable

function a(){

document.writeln(data);

}

function b(){

document.writeln(data);

}

a();//calling JavaScript function

b();

</script>

1. **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:

var a=40;//holding number

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:

|  |  |
| --- | --- |
| **Data Type Description** | |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

**Syntax :**

var length = 16; // Number

var lastName = "Johnson"; // String

var cars = ["Saab", "Volvo", "BMW"]; , var cars = new Array("Saab", "Volvo", "BMW" ); // Array

var x = {firstName:"John", lastName:"Doe"}; // Object

1. **Operators:**

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

## 1.JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | 10+20 = 30 |
| - | Subtraction | 20-10 = 10 |
| \* | Multiplication | 10\*20 = 200 |
| / | Division | 20/10 = 2 |
| % | Modulus (Remainder) | 20%10 = 0 |
| ++ | Increment | var a=10; a++; Now a = 11 |
| -- | Decrement | var a=10; a--; Now a = 9 |

## 2.JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Is equal to | 10==20 = false |
| === | Identical (equal and of same type) | 10==20 = false |
| != | Not equal to | 10!=20 = true |
| !== | Not Identical | 20!==20 = false |
| > | Greater than | 20>10 = true |
| >= | Greater than or equal to | 20>=10 = true |
| < | Less than | 20<10 = false |
| <= | Less than or equal to | 20<=10 = false |

## 3.JavaScript Bitwise Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Bitwise AND | (10==20 & 20==33) = false |
| | | Bitwise OR | (10==20 | 20==33) = false |
| ^ | Bitwise XOR | (10==20 ^ 20==33) = false |
| ~ | Bitwise NOT | (~10) = -10 |
| << | Bitwise Left Shift | (10<<2) = 40 |
| >> | Bitwise Right Shift | (10>>2) = 2 |
| >>> | Bitwise Right Shift with Zero | (10>>>2) = 2 |

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

## 4.JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (10==20 && 20==33) = false |
| || | Logical OR | (10==20 || 20==33) = false |
| ! | Logical Not | !(10==20) = true |

## 5.JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assign | 10+10 = 20 |
| += | Add and assign | var a=10; a+=20; Now a = 30 |
| -= | Subtract and assign | var a=20; a+=10; Now a = 10 |
| \*= | Multiply and assign | var a=10; a\*=20; Now a = 200 |
| /= | Divide and assign | var a=10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a=10; a%=2; Now a = 0 |

## 6.JavaScript Special Operators

The following operators are known as JavaScript special operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | Comma Operator allows multiple expressions to be evaluated as single statement. |
| delete | Delete Operator deletes a property from the object. |
| In | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| New | creates an instance (object) |
| typeof | checks the type of object. |
| Void | it discards the expression's return value. |
| Yield | checks what is returned in a generator by the generator's iterator. |

## JavaScript Display Possibilities

**JavaScript can "display" data in different ways:**

* Writing into an alert box, using **window.alert( )**.
* Writing into the HTML output using **document.write( )**.
* Writing into an HTML element, using **innerHTML**.
* Writing into the browser console, using **console.log( )**.

**Using windows.alert( )**

Display the output in pop up menu.

**Using document.write( )**

Display the output in window and for testing purposes it is convenient to use.

**Using innerHTML( )**

To access an HTML element, JavaScript can use the **document.getElementById(id)** method and to change the HTML element into innerHTML we will use **innerHTML( )**

## Using console.log( )

In your browser, you can use the **console.log()** method to display data. Activate the browser console with F12, and select "Console" in the menu.

**JavaScript Structure:**

* Javascript code located inside HTML, script code can write inside script tag ( <script> </script> )

</DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<div>

<p>regular body</p>

</div>

<script> alert(“hello”); </script> // We can specify <script> inside body and inside head also.

</body>

</html>

**Javascript is Interpreted Language, not Compiled.**

An **interpreted language** is a [programming language](https://en.wikipedia.org/wiki/Programming_language) for which most of its implementations execute instructions directly, without previously [compiling](https://en.wikipedia.org/wiki/Compiler) a [program](https://en.wikipedia.org/wiki/Computer_program) into [machine-language](https://en.wikipedia.org/wiki/Machine_language) instructions. The [interpreter](https://en.wikipedia.org/wiki/Interpreter_(computing)) executes the program directly, translating each statement into a sequence of one or more [subroutines](https://en.wikipedia.org/wiki/Subroutines) already compiled into machine code.

## Advantages and Disadvantages of Interpreted Languages

**Advantages**

* easy to learn and use
* minimum programming knowledge or experience
* allows complex tasks to be performed in relatively few steps
* allows simple creation and editing in a variety of text editors
* allows the addition of dynamic and interactive activities to web pages
* edit and running of code is fast.

**Disadvantages**

* usually run quite slowly
* Limited access to low level and speed optimization code.
* Limited commands to run detailed operations on graphics.

## Advantages and Disadvantages of Compiled Languages

**Advantages**

* fast execution
* optimized for the target hardware

**Disadvantages**

* require a compiler
* editing and deploying the code is a lot slower than interpreters.

**Important properties of Javascript**

**Javascript is Case sensitive:**

alert(“hello”); // wrong

alert(“hello”); //correct.

**Javascript Forgive some omissions:**

alert(“hello”)**; //** correct.

alert(“hello”) //correct, if alert will not end with semicolon(**;**)

**Javascript is Whitespace Insensitive:**

**Javascript allowes the whitespaces.**

alert(“hello world”); //correct.

alert( “hello world”); // correct

alert

( “hello world” // correct

);

alert(“hello world”); //correct

**Javascript Language:**

1.Variables

2.Conditional code

3.Operators.

4.Loops

5.Functions

6.Comments

7. Data Types.

**1.Variable**

* Variable is a container used for temporary storage of data. The variables that are used in the program have to be declared before its use.
* All JavaScript **variables** must be **identified** with **unique names**.
* These unique names are called **identifiers**.
* Identifiers can be short names (like x and y), or more descriptive names (age, sum, totalVolume).
* There are two types of variables in JavaScript: local variable and global variable.

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter
* Names can also begin with $ and \_
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

**Creating variables:**

Var year; // if variable not initialized or varible not assigned with any value, considered as **undefined.**

Var year = 2011; // avoid undefined, we should intialize value (ex.2011) to variable,

**Variable names are case sensitive:**

10

var x= 10; x

var X=20; X

20

**Multiple Variables:**

var year;

var month; // variables can have multiple lines.

var day;

**OR**

var year, month, day; // we can also write multiple variable names in a single line

**Multiple variables with Value.**

var year= 2015;

var month= 10;

var day=21;

**OR**

var year= 2015, month= 10, day= 21; // we can also write multiple variables with value in a single line.

**Variable Data Types:**

Variables can be initialized with Interger, floating point number, Text string, Boolen value ( true, false), array, date and object.

Examples:

var myVariable; // undefined

var myVariable = 12; // initialized with **integer**

var myVariable= “Hello”; // initialized with string with **double quote** ( “ “ ).

var myVariable= ‘Hello’; // initialized with string with **single quote** ( ‘ ‘ ).

var myVariable= true; // initialized with **boolean value true.**

var myVariable=false; // initialized with **boolean value false.**

var myVariable = 12.2; // initialized with **floating point number.**

**2. Conditional Code**

* **if**

if ( condition) {

// code goes here ( code block)

//…..

}

**Note:**

**( )** - Parentheses.

**[ ]** - Brackets.

**{ }** - Braces.

* **if else**

if ( condition) {

// code goes here ( code block)

//…..

} else {

// otherwise, different code

if ( condition) {

// nested if

}

}

**3. Operators.**

* **Arithmetic operators ( +, -, \*, / , %, ++, --)**

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Examples:

var score;

score= 20 + 10; // calculates and display 30

score= 20 - 10; // calculates and display 10

score= 20 \* 10; // calculates and display 200

score= 20 / 10; // calculates and display 10

var a=10, var b;

b= a++; // first a value assign to b then increment by 1 so, now b value is 10.

b= ++a; // first a value increment by 1 then assign to so, now b value is 11.

b= a--; // first a value assign to b then decrement by 1 so, now b value is 10.

b= --a; // first a value increment by 1 then assign to so, now b value is 9.

* **Assignment operators ( =, +=, -=, \*=, /=, %=)**

Examples:

var score= 10;

score= score + 10; // now score = 10

score += 10; // now score = 10, adds and assign

var score= 20;

score -= 10; // now score 10, subtract and assign

score \*=10; // now score= 200 , multiplies and assign

score /=10; // now score 10, divides and assign

score %=10; // now score = 0, takes remainder and assign

* **Relational (Comparison) operators ( ==, ===, !=, !==, >, >=, <, <= )**

The JavaScript comparison operator compares the two operands.

**Note:**

**= assignment**

**== equality**

**=== strict equality**

Example:

var a= 5;

var b= “5”;

document.write (a == b); // returns false, it compares only value.

document.write (a === b); // returns true, it compare the data type also.

* **Logical AND/OR**

if(a === b && c === d) // logical AND operator

Example :

if(10 === 20 && 22 === 33) // returns boolean value false

if(a === b || c === d) // logical OR operator

Example :

if(10 === 20 || 22 === 33) // returns boolean value false

* **Ternary operator**

Condition ? true : false

Example :

var a= 10, b= 20;

if ( a>b) ? a : b // checks the condition if condition true returns a value else b value.

**4.Working with Loops**

The **JavaScript loops** are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop

## 1) JavaScript for loop

The **JavaScript for loop** iterates the elements for the fixed number of times. It should be used if number of iteration is known. The syntax of for loop is given below.

for (initialization; condition; increment)

{

    code to be executed

}

Example :

for (i=1; i**<**=5; i++)

{

document.write(i ) ;  // Displays 12345

}

## 2) JavaScript while loop

The **JavaScript while loop** iterates the elements for the infinite number of times. It should be used if number of iteration is not known. The syntax of while loop is given below.

while (condition)

{

    code to be executed

}

Example:

var i=11;

while (i**<**=15)

{

document.write(“ “ + i );  // displays 11 12 13 14 15

i++;

}

## 3) JavaScript do while loop

The **JavaScript do while loop** iterates the elements for the infinite number of times like while loop. But, code is **executed at least** **once** whether condition is true or false. The syntax of do while loop is given below.

do{

    code to be executed

}while (condition);

Example :

var i=21;

do{

document.write(“ “ + i );  // displays 21 22 23 24 25

i++;

}while (i**<**=25);

# 5.JavaScript Functions

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

#### Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

## JavaScript Function Syntax

The syntax of declaring function is given below.

function functionName([arg1, arg2, ...argN]){

 //code to be executed

}

**Note : JavaScript Functions can have 0 or more arguments.**

Example 1:

**<script>**

function msg(){

alert("hello! this is message");  // displays hello! this is message in pop up menu.

}

**</script>**

Example 2: Function with Arguments

**<script>**

function msg(num){

alert(num\*num\*num);  // displays 64

}

msg(4);

**</script>**

Example 2: Function with Return value

**<script>**

function getInfo(){

return "hello";   // displays hello

}

document.write(getInfo());

**</script>**

# 6.JavaScript Comment

1. [JavaScript comments](http://www.javatpoint.com/javascript-comment)
2. [Advantage of javaScript comments](http://www.javatpoint.com/javascript-comment)
3. [Single-line and Multi-line comments](http://www.javatpoint.com/javascript-comment)

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

#### Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

## Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

## JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement. Let’s see the example of single-line comment i.e. added before the statement.

**<script>**

// It is single line comment

document.write("hello javascript");

**</script>**

Example:

**<script>**

var a=10;

var b=20;

var c= a+b;//It adds values of a and b variable

document.write(c); //prints sum of 10 and 20

**</script>**

## JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by /\* your code here  \*/

It can be used before, after and middle of the statement.

**<script>**

/\* It is multi line comment.

It will not be displayed \*/

document.write("example of javascript multiline comment");

**</script>**

# 7.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 ( Number, String, Boolean, Floating point number)
2. Non-primitive (reference) data type ( Array, Object)

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.

example:

var a=40; //holding number

var b="Rahul"; //holding string

# Javascript Array:

**JavaScript array** is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword or without using new keyword)
3. By using an Array constructor (using new keyword)

## 1) JavaScript array literal

The syntax of creating array using array literal is given below:

var arrayname= [value1,value2.....valueN];

Example:

**<script>**

var emp=["Sonoo","Vimal","Ratan"];

for (i=0;i**<emp.length**;i++){

document.write(emp[i]);  // displays Sonoo Vimal Ratan

}

**</script>**

## 2) JavaScript Array directly (new keyword or without using new keyword)

The syntax of creating array directly is given below:

var arrayname= new Array();

var arrayname= Array(); // without using new keyword.

Here, **new keyword** is used to create instance of array.

Example:

**<script>**

var i;

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

for (i=0;i**<emp.length**;i++){

document.write(“ “+emp[i]);  // displays Arun Varun John

}

**</script>**

## 3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

**<script>**

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i**<emp.length**;i++){

document.write(“ “ +emp[i]);  //displays Jai Vijay Smith

}

**</script>**

# JavaScript Array Methods

## Converting Arrays to Strings

* The JavaScript method **toString()** converts an array to a string of (comma separated) array values.
* The **join()** method also joins all array elements into a string. It behaves just like toString(), but in addition you can specify the separator:

## Popping and Pushing

When we work with arrays, it is easy to remove elements and add new elements. Popping items **out** of an array, or pushing items **into** an array.

## Popping

The **pop()** method removes the last element from an array. The pop() method returns the value that was "popped out":

## Pushing

The **push()** method adds a new element to an array (at the end). The push() method returns the new array length:

## Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last. The **shift()** method removes the first element of an array, and "shifts" all other elements one place up.

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

The shift() method returns the string that was "shifted out".The unshift() method returns the new array length.

## Changing Elements

Array elements are accessed using their **index number**. The length property provides an easy way to append a new element to an array:

## Sorting an Array

The **sort()** method sorts an array alphabetically.

* Reversing an Array

The **reverse()** method reverses the elements in an array.

## Joining Arrays

The **concat()** method creates a new array by concatenating two arrays:

# JavaScript Strings

JavaScript strings are used for storing and manipulating text. A string can be any text inside quotes. You can use single or double quotes:

var str = “Hi Hello”; // string can write inside double quote.

var str = “Hi Hello”; // string can write inside single quote.

var str = ‘Hi Hello”; // string can’t write one double and othe single quote.

Quotes inside Quotes:

var str = ‘Don’t mix quotes’ // wrong, inside single quote we should not write one more single quote.

var str = “Don’t mix quotes” // correct, within double quote we can write single quote.

var str = “He said “That’s fine” and left” // wrong

var str = “He said \“That’s fine\” and left” // correct.

**String Properties:**

By using string Property **length**, we can find the length of a given string.

Var str = “Hi Hello”;

Document.write(str.length); // display 8

**String Methods:**

* **toUpperCase() and toLowerCase()** : These two methods used to convert lower case letters to upper case and upper case letters to lower case respectively.

Var str = “hi hello”;

Document.write(str.toUpperCase()); // display HI HELLO

Var str = “HI HELLO”;

Document.write(str.toLowerCase()); // display hi hello

* **indexOf() :** Methos used to get index position of given string and it returns -1 if the string is not found.

**Example:**

Var str = “hi hello”;

Document.write(str.indexOf(“hello”)); // display 3

**//If string not fond returns -1.**

If( str.indexOf(“add”) == -1 )) {

Document.write(“That world does not occur”);

}

**slice(start, end), substring(start,end) and Substr( start,length)**

**a) substring(start,end)**

1. If start equals stop, it returns an empty string.
2. If stop is omitted, it extracts characters to the end of the string.
3. If start > stop, then substring will swap those 2 arguments.
4. If either argument is greater than the string's length, either argument will use the string's length.
5. If either argument is less than 0 or is NaN, it is treated as if it were 0.

**b) substring(start,end)**

1. If start equals stop, it returns an empty string, exactly like substring().
2. If stop is omitted, slice extracts chars to the end of the string, exactly like substring().
3. If start > stop, slice() will NOT swap the 2 arguments.
4. If either argument is greater than the string's length, either argument will use the string's length, exactly like substring().

5.

* + If start is negative, slice() will set string length + start index position.
  + If stop is negative, slice() will set string length + start index position.

**c) Substr( start,length)**

1. If start and length both zero, it returns an empty string, exactly like substring() and slice()
2. If length is negative and zero, it returns an empty string,
3. If start is negative, substr() strarts counting from reverse and display upto length mentioned.
4. If strart is greater than string length, , it returns an empty string.

Example:

<html>

<body>

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

<script>

var str = "priya";

**Slice(start, end) methods example**

1. document.getElementById("demo").innerHTML = str.slice(1,3);

**//display “ri**”**, slice method starts consider start index position to end indexposition-1.**

2.document.getElementById("demo").innerHTML = str.slice(3,1);

**// display empty string, slice will not count from higherindex position to lower index position.**

3.document.getElementById("demo").innerHTML = str.slice(-1,-3);

**// display empty string,if starts index position is negative it will take (string length + start index position) as start index position**

4.document.getElementById("demo").innerHTML = str.slice(-3,-1);

**//display "iy",if end index position is negative it will take (string length + end index position) as end index position and start consider from lower to higher index position.**

5.document.getElementById("demo").innerHTML = str.slice(0);

**//display priya, slice method considers zero index position and display rest of the string.**

6.document.getElementById("demo").innerHTML = str.slice(6);

**//display empty string, if index postion is greater than string length.**

**Substring(start, end) methods example**

1.document.getElementById("demo").innerHTML = str.substring(3,1);

**//display ri, substring method starts consider start index position to end index position-1.**

2.document.getElementById("demo").innerHTML = str.substring(1,3);

**//display ri,if start index position is high substring will swap both index postion and make it low to high position.**

3.document.getElementById("demo").innerHTML = str.substring(-1,-3);

**//display empty string,substring will consider negative values as "zero" index position.**

4.document.getElementById("demo").innerHTML = str.substring(-1,3);

**//display empty "pri" string,substring will consider negative values as "zero" index position so it consider substring(0,3).**

5.document.getElementById("demo").innerHTML = str.substring(0);

**//display ” priya”, substring method considers zero index position and display rest of the string.**

6.document.getElementById("demo").innerHTML = str.substring(6);

**//display empty string, if index postion is greater than string length.**

**Substr(start, length) methods example**

1.document.getElementById("demo").innerHTML = str.substr(3,1);

**//display y, substr method starts consider start index position to length mentioned.**

2.document.getElementById("demo").innerHTML = str.substr(-1,-3);

//**length is negative and zero, it returns an empty string,**

6.document.getElementById("demo").innerHTML = str.substr(6);

**//display empty string, if index postion is greater than string length.**

</script>

</body>

</html>

# JavaScript Date Object

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

## Constructor

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds. You can use 4 variant of Date constructor to create date object.

Date()

Date(milliseconds)

Date(dateString)

Date(year, month, day, hours, minutes, seconds, milliseconds)

Example:

var today= new Date(); // Current date and time.

var d= new Date(2000,0,1); // year,month,day.

var m= new Date(2000,0,1,0,0,0); // year,month,day,hours,minutes,seconds.

**Methods of DATE Object:**

var today= new Date()

today.getMonth(); // returns 0-11

today.getFullYear(); // yyyy (not zero based)

today.getYear(); // deprecated

today.getDate(); // 1-31 day of month

today.getDay(); // 0-6 day of week. 0 == sunday

today.getHours(); // 0-23 hours

today.getTime(); // milliseconds since 1/1/1970

**JavaScript Objects**

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

## Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

## 1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

**<script>**

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);  // 102 Shyam Kumar 40000

**</script>**

## 2) By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

**<script>**

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);  // 101 Ravi 50000

**</script>**

## 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

**<script>**

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e= new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);  // 103 Vimal Jaiswal 30000

**</script>**

## The typeof Operator

The **typeof** operator to find the data type of a JavaScript variable.

typeof "John"                   // Returns string   
typeof 3.14                    // Returns number  
typeof NaN                      // Returns number  
typeof false                   // Returns boolean  
typeof [1,2,3,4]               // Returns object  
typeof {name:'John', age:34}   // Returns object  
typeof new Date()              // Returns object  
typeof function () {}          // Returns function  
typeof myCar                   // Returns undefined (if myCar is not declared)  
typeof null                    // Returns object

**Note:**

* The data type of NaN is number
* The data type of an array is object
* The data type of a date is object
* The data type of null is object
* The data type of an undefined variable is undefined

## The Data Type of typeof

The typeof operator is not a variable. It is an operator. Operators ( + - \* / ) do not have any data type.

But, the typeof operator always **returns a string** containing the type of the operand.

**DOM ( Document Object Model )**

## What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

* Core DOM - standard model for all document types
* XML DOM - standard model for XML documents
* HTML DOM - standard model for HTML documents

## What is the HTML DOM?

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

* The HTML elements as **objects**
* The **properties** of all HTML elements
* The **methods** to access all HTML elements
* The **events** for all HTML elements

In other words:**The HTML DOM is a standard for how to get, change, add, or delete HTML elements.** With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

## The HTML DOM ( Document Object Model )

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

**Document** is :

* HTML Page
* Source code

**Object** is :

* It can be contents of inside the Unordered list
* It can be contents of inside the body tag
* It can be contents of inside the HTML tag As shown in figure boxes

**Model** is :

* Agreed upon set of terms:
* When a web page is loaded, the browser creates a Document Object Model of the page.
* With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
* The **HTML DOM** model is constructed as a tree of **Objects**.
* With the object model, JavaScript gets all the power it needs to create dynamic HTML:
* JavaScript can change all the HTML elements in the page
* JavaScript can change all the HTML attributes in the page
* JavaScript can change all the CSS styles in the page
* JavaScript can remove existing HTML elements and attributes
* JavaScript can add new HTML elements and attributes
* JavaScript can react to all existing HTML events in the page
* JavaScript can create new HTML events in the page