CHAPTER 1 :

INTRODUCTION :

**History of JavaScript :**

* JavaScript was created in 10 days in May 1995 by [Brendan Eich](http://en.wikipedia.org/wiki/Brendan_Eich), when working at [Netscape](http://en.wikipedia.org/wiki/Netscape) and now of [Mozilla](http://www.mozilla.com).
* JavaScript was not always known as JavaScript: the original name was Mocha, a name chosen by [Marc Andreessen](http://en.wikipedia.org/wiki/Marc_Andreessen), founder of Netscape.
* In September of 1995 the name was changed to LiveScript, then in December of the same year, upon receiving a trademark license from Sun, the name JavaScript was adopted.

**What is JavaScript?**

* JavaScript is an object-based scripting language that is lightweight and cross-platform.
* JavaScript is the programming language of HTML and the Web.
* It is designed for creating network-centric applications.
* JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.
* JavaScript does not designed as general purpose programming language but it designed to manipulate web pages.
* [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) is a major Web technology that provides interactivity and special effects to WebPages.
* JavaScript can be used inside another applications i.e. Web Browser.
* Operating system runs the Web Browser, and the Web Browser is having the implementation of JavaScript engine.

**Why Study JavaScript?**

JavaScript is one of the **3 languages** all web developers **must** learn:

* 1. **HTML** to define the content of web pages
* 2. **CSS** to specify the layout of web pages
* 3. **JavaScript** to program the behavior of web pages

**Why JavaScript is used in Web?**

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).
* Dynamic styling,
* Games
* Responses when buttons are pressed or data entered in forms
* animation
* Displaying Clock etc.

**Other use of JavaScript :**

* Applications(Acrobat, Photoshop)
* Server-Side(Node.js, Google Apps Script)

**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.

**Disadvantages of JavaScript :**

* JavaScript can be disabled at client side (But it will not affect because we can enable it while writing JavaScript).
* Different browser can display web pages differently because of not having the proper implementation of CSS.

## 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.
* Can’t access database directly.
* Can’t access Hardware (USB etc).

## What are the tools to write JavaScript?

* Notepad
* Notepad++
* Dreamweaver
* Xcode
* Visual studios
* Aptana etc

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.

CHAPTER 2 :

JAVASCRIPT SYNTAX :

## JavaScript Language :

## Character Set

## Keywords

## Identifiers

## Variable

## Data Types

## Literals

## Operators

## Control Statement

## 1.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 followin3 types.

## a) Numbers - 0 to 9.

## b) Alphabets – A to Z, A to z

## c) Special Characters - #, $, etc

## It uses [**UTF-16**](https://tools.ietf.org/html/rfc2781) (16-bit Unicode Transformation Format).

## 2.Keywords :

JavaScript **keywords** are used to identify actions to be performed.

The **var** keyword tells the browser to create a new variable:

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

****

3. Identifiers :

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, an underscore (\_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

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.

Here are some examples of valid identifier naming conventions:

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

4.variable

is simply a name of storage location. There are two types of variables in JavaScript : local

variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1.Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.

2.After first letter we can use digits (0 to 9), for example value1.

3.JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

var x = 10;

var \_value="sonoo";

Incorrect JavaScript variables

var  123=30;

var \*aa=320;

**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>**

Or,

**<script>**

If(10**<13**){

var y=20;//JavaScript 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;//gloabal variable

function a(){

document.writeln(data);

}

function b(){

document.writeln(data);

}

a();//calling JavaScript function

b();

**</script>**

**5. Data types :**

In JavaScript, there are three primary data types, two composite data types, and two special data

types.

[**Primary Data Types**](javascript:void(0)) **:**

The primary (primitive) data types are:

1. String
2. Number
3. Boolean

[**Composite Data Types**](javascript:void(0)) **:**

The composite (reference) data types are:

1. Object
2. Array

[**Special Data Types**](javascript:void(0)) **:**

The special data types are:

1. Null
2. Undefined

Example :

var length = 16; // Number

var lastName = "Johnson"; // String

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

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

## 6. Literals :

The most important rules for writing fixed values are:

**Numbers** are written with or without decimals:

10.50  
  
1001

**Strings** are text, written within double or single quotes:

"JohnDoe"  
  
'John Doe'

**7. Operators:**

JavaScript operators are symbols that are used to perform operations on operands.

For example: var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

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

**Arithmetic Operators:**

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

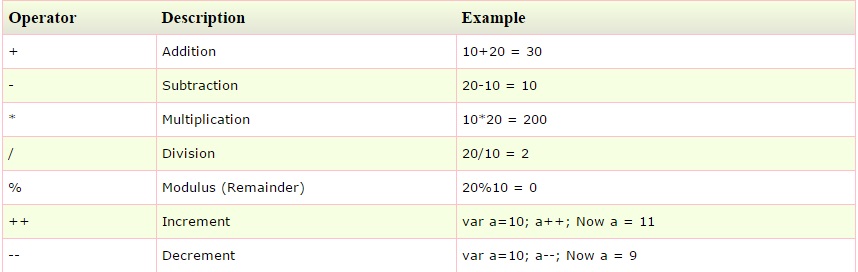
operators are known as JavaScript arithmetic operators.

Arithmetic operators perform arithmetic on numbers (literals or variables).

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**

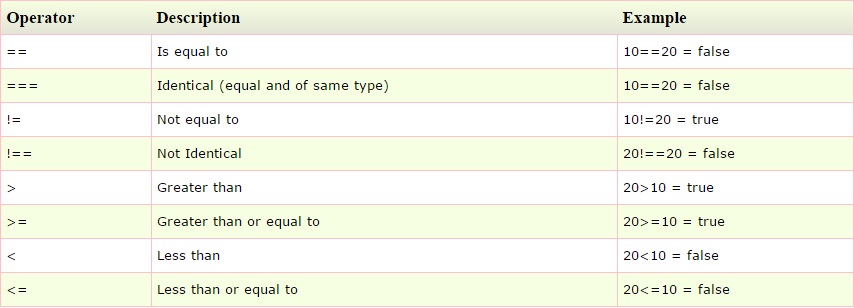




**JavaScript Comparison Operator:**

The JavaScript comparison operator compares the two operands. The comparison operators are

as follows:



**Bitwise Operators:**

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

follows:

|  |  |
| --- | --- |
| **Sr.No** | **Operator and Description** |
| 1 | **& (Bitwise AND)**  It performs a Boolean AND operation on each bit of its integer arguments.  **Ex:** (A & B) is 2. |
| 2 | **| (BitWise OR)**  It performs a Boolean OR operation on each bit of its integer arguments.  **Ex:** (A | B) is 3. |
| 3 | **^ (Bitwise XOR)**  It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.  **Ex:** (A ^ B) is 1. |
| 4 | **~ (Bitwise Not)**  It is a unary operator and operates by reversing all the bits in the operand.  **Ex:** (~B) is -4. |
| 5 | **<< (Left Shift)**It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.  **Ex:** (A << 1) is 4. |
| 6 | **>> (Right Shift)**  Binary Right Shift Operator. The left operand’s value is moved right by the number of bits specified by the right operand.  **Ex:** (A >> 1) is 1. |
| 7 | **>>> (Right shift with Zero)**  This operator is just like the >> operator, except that the bits shifted in on the left are always zero.  **Ex:** (A >>> 1) is 1. |

**Logical Operator:**

JavaScript supports the following logical operators −Assume variable A holds 10 and variable B holds

20,then –

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

**Assignment Operator:**

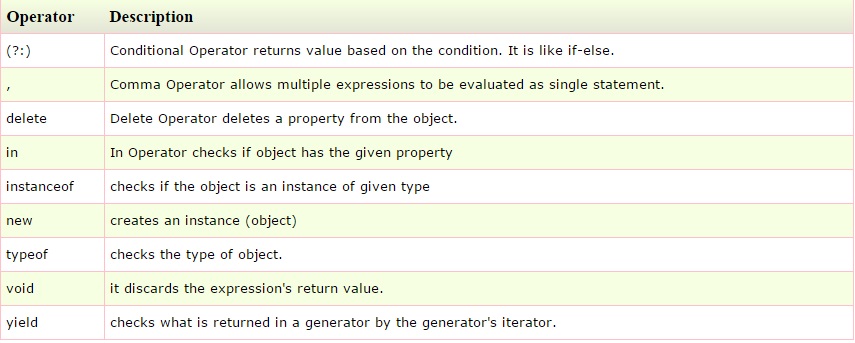
JavaScript supports the following assignment operators –

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

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

**Special Operators:**

The following operators are known as JavaScript special operators.

****

**8) Control Statement:**

1. if / else
2. switch / case
3. for
4. while and do / while
5. break
6. continue

# If...Else Statements :

# Conditional statements are used to perform different actions based on different conditions.

Conditional Statements :

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

* Use **if** to specify a block of code to be executed, if a specified condition is true
* Use **else** to specify a block of code to be executed, if the same condition is false
* Use **else if** to specify a new condition to test, if the first condition is false
* Use **switch** to specify many alternative blocks of code to be executed

## The if Statement :

## Use the **if** statement to specify a block of JavaScript code to be executed If a condition is true.

### Syntax :

### if (condition) {

### block of code to be executed if the condition is true

### }

|  |  |
| --- | --- |
|  |  |

### Example

Make a "Good day" greeting if the hour is less than 18:00:

if (hour < 18) {

greeting = "Good day";

}

## The else Statement :

Use the **else** statement to specify a block of code to be executed if the condition is false.

if (condition) {

block of code to be executed if the condition is true

}

Else

{

block of code to be executed if the condition is false

}

### Example :

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

if (hour < 18) {

greeting = "Good day";} else {

greeting = "Good evening";

}

## The else if Statement :

Use the **else if** statement to specify a new condition if the first condition is false.

### Syntax :

### if (condition1) {

### block of code to be executed if condition1 is true

### } else if (condition2) {

### block of code to be executed if the condition1 is false and condition2 is true

### } else {

### block of code to be executed if the condition1 is false and condition2 is false

### }

### Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

if (time < 10) {

greeting = "Good morning";

} else if (time < 20) {

greeting = "Good day";

} else {

greeting = "Good evening";

}

## Switch Statement :

## Use the switch statement to select one of many blocks of code to be executed.

### Syntax :

### switch(expression) {

### case n*:*

### code block

### break;

### case n:

### code block

### break;

### default:

### default code block

### }

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.

### Example :

### The getDay() method returns the weekday as a number between 0 and 6. (Sunday=0, Monday=1, Tuesday=2 ..)

Use the weekday number to calculate weekday name:

switch (new Date().getDay()) {

case 0:

        day = "Sunday"

        break;

    case 1 : day = "Monday";

         break;

    case 2: day = "Tuesday";

         break;

    case 3: day = "Wednesday";

break;

    case 4 :day = "Thursday";

break;  
    case 5: day = "Friday";

       break;

case 6:  
        day = "Saturday";

        break;  
}

## The break Keyword :

When the JavaScript code interpreter reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

## The default Keyword :

## The **default** keyword specifies the code to run if there is no case match:

### Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

switch (new Date().getDay()) {

case 6:

        text = "Today is Saturday";

  break;

    case 0:text = "Today is Sunday";

break;

    default:

        text = "Looking forward to the Weekend";

}

## JavaScript Loops :

Loops are handy, if you want to run the same code over and over again, each time with a

different value.

Often this is the case when working with arrays:

### Instead of writing:

text += cars[0] + "<br>";

text += cars[1] + "<br>";

text += cars[2] + "<br>";

text += cars[3] + "<br>";

text += cars[4] + "<br>";

text += cars[5] + "<br>";

### You can write:

for (i = 0; i < cars.length; i++){

   text += cars[i] + "<br>";

}

## Different Kinds of Loops :

JavaScript supports different kinds of loops:

* **for** - loops through a block of code a number of times
* **for/in** - loops through the properties of an object
* **while** - loops through a block of code while a specified condition is true
* **do/while** - also loops through a block of code while a specified condition is true

The For Loop :

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following syntax:

for (statement 1; statement 2; statement 3) {

code block to be executed

}

**Statement 1** is executed before the loop (the code block) starts.

**Statement 2** defines the condition for running the loop (the code block).

**Statement 3** is executed each time after the loop (the code block) has been executed.

### Example

for (i = 0; i < 5; i++) {

text += "The number is " + i + "<br>";

}

## The For/In Loop :

The JavaScript for/in statement loops through the properties of an object:

### Example

var person = {fname:"John", lname:"Doe", age:25};

var text = "";

var  x;

for (x in person) {

    text += person[x];

}

The While Loop :

The while loop loops through a block of code as long as a specified

condition is true.

Syntax

while (condition) {

    code block to be executed

}

### Example

### In the following example, the code in the loop will run, over and over again, as long as a variable

### (i) is less than 10:

### Example

### while (i < 10) {

### text += "The number is " + i;

### i++; }

## The Do/While Loop :

The do/while loop is a variant of the while loop. This loop will execute the code block once,

before checking if the condition is true, then it will repeat the loop as long as the condition is

true.

### Syntax

do {

    code block to be executed

}  
while (condition);

### Example:

### The example below uses a do/while loop. The loop will always be executed at least once, even if

the condition is false, because the code block is executed before the condition is tested:

### Example

do {

    text += "The number is " + i;

    i++;

}  
while (i < 10);

CHAPTER 3

TYPES AND OBJECTS :

* Arrays
* Numbers
* String
* Dates
* Objects

# 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)
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];

As you can see, values are contained inside [ ] and separated by , (comma).

Let’s see the simple example of creating and using array in JavaScript.

**<script>**

var emp=["Sonoo","Vimal","Ratan"];

for (i=0;i**<emp.length**;i++){

document.write(emp[i] + "**<br/>**");

}

**</script>**

## 2) JavaScript Array directly (new keyword) :

The syntax of creating array directly is given below:

var arrayname=new Array();

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

Let’s see the example of creating array directly.

**<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] + "**<br>**");

}

**</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 explicitely.

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] + "<br>");

}

</script>

# JavaScript Number Object :

The **JavaScript number** object enables you to represent a numeric value*.* It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, you can create number object in JavaScript. For example:

var n=new Number(value);

If value can't be converted to number, it returns NaN(Not a Number) that can be checked

by isNaN() method.

You can direct assign a number to a variable also. For example:

var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object

## JavaScript Number Constants :

Let's see the list of JavaScript number constants with description.

|  |  |
| --- | --- |
| **Constant** | **Description** |
| MIN\_VALUE | returns the largest minimum value. |
| MAX\_VALUE | returns the largest maximum value. |
| POSITIVE\_INFINITY | returns positive infinity, overflow value. |
| NEGATIVE\_INFINITY | returns negative infinity, overflow value. |
| NaN | represents "Not a Number" value. |

## JavaScript Number Methods

Let's see the list of JavaScript number methods with description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| toExponential(x) | displays exponential value. |
| toFixed(x) | limits the number of digits after decimal value. |
| toPrecision(x) | formats the number with given number of digits. |
| toString() | converts number into string. |
| valueOf() | coverts other type of value into number. |

# JavaScript String :

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

## 1) By string literal :

## The string literal is created using double quotes. The syntax of creating string using string literal is given below:

## var stringname="string value";

## Let’s see the simple example of creating string literal.

## <script>

## var str="This is string literal";

## document.write(str);

## </script>

## 2) By string object (using new keyword) :

The syntax of creating string object using new keyword is given below:

var stringname=new String("string literal");

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

Let’s see the example of creating string in JavaScript by new keyword.

**<script>**

var stringname=new String("hello javascript string");

document.write(stringname);

**</script>**

## JavaScript String Methods :

## Let's see the list of JavaScript string methods with examples.

* charAt(index)
* concat(str)
* indexOf(str)
* lastIndexOf(str)
* toLowerCase()
* toUpperCase()
* slice(beginIndex, endIndex)
* trim()

## 1) JavaScript String charAt(index) Method :

The JavaScript String charAt() method returns the character at the given index.

<script>

var str="javascript";

document.write(str.charAt(2));

</script>

## 2) JavaScript String concat(str) Method :

## The JavaScript String concat(str) method concatenates or joins two strings.

## <script>

## var s1="javascript ";

## var s2="concat example";

## var s3=s1.concat(s2);

## document.write(s3);

## </script>

## 3) JavaScript String indexOf(str) Method :

## The JavaScript String indexOf(str) method returns the index position of the given

## string.

## <script>

## var s1="javascript from javatpoint indexof";

## var n=s1.indexOf("from");

## document.write(n);

## </script>

4) JavaScript String lastIndexOf(str) Method :

## The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

## <script>

## var s1="javascript from javatpoint indexof";

## var n=s1.lastIndexOf("java");

## document.write(n);

## </script>

## 5) JavaScript String toLowerCase() Method :

## The JavaScript String toLowerCase() method returns the given string in lowercase letters.

## <script>

## var s1="JavaScript toLowerCase Example";

## var s2=s1.toLowerCase();

## document.write(s2);

## </script>

## 6) JavaScript String toUpperCase() Method :

## The JavaScript String toUpperCase() method returns the given string in uppercase letters.

## <script>

## var s1="JavaScript toUpperCase Example";

## var s2=s1.toUpperCase();

## document.write(s2);

## </script>

## 7) JavaScript String slice(beginIndex, endIndex) Method :

## The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given

## beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

## <script>

## var s1="abcdefgh";

## var s2=s1.slice(2,5);

## document.write(s2);

## </script>

## 8) JavaScript String trim() Method :

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

**<script>**

var s1="     javascript trim    ";

var s2=s1.trim();

document.write(s2);

**</script>**

# 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.

# You can use different Date constructors to create date object. It provides methods to get

# and set day, month, year, hour, minute and seconds.

## Constructor :

You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

## JavaScript Date Methods :

The important methods of date object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| getFullYear() | returns the year in 4 digit e.g. 2015. It is a new method and suggested than getYear()  which is now deprecated. |
| getMonth() | returns the month in 2 digit from 1 to 31. |
| getDate() | returns the date in 1 or 2 digit from 1 to 31. |
| getDay() | returns the day of week in 1 digit from 0 to 6. |
| getHours() | returns all the elements having the given name value. |
| getMinutes() | returns all the elements having the given class name. |
| getSeconds() | returns all the elements having the given class name. |
| getMilliseconds() | returns all the elements having the given tag name. |

### JavaScript Date Example :

Let's see the simple example to print date object. It prints date and time both.

Current Date and Time: **<span** id="txt"**></span>**

**<script>**

var today=new Date();

document.getElementById('txt').innerHTML=today;

**</script>**

### JavaScript Current Time Example :

Let's see the simple example to print current time of system.

Current Time: **<span** id="txt"**></span>**

**<script>**

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

**</script>**

### JavaScript Digital Clock Example :

Let's see the simple example to display digital clock using JavaScript date object.

There are two ways to set interval in JavaScript: by setTimeout() or setInterval() method.

Current Time: **<span** id="txt"**></span>**

**<script>**

window.onload=function(){getTime();}

function getTime(){

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

// add a zero in front of numbers**<10**

m=checkTime(m);

s=checkTime(s);

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

setTimeout(function(){getTime()},1000);

}

//setInterval("getTime()",1000);//another way

function checkTime(i){

if (i**<10**){

  i="0" + i;

 }

return i;

}

**</script>**

# 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).

Let’s see the simple example of creating object in JavaScript.

**<script>**

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

document.write(emp.id+" "+emp.name+" "+emp.salary);

**</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);

**</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);

**</script>**

## Defining method in JavaScript Object :

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

**<script>**

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);

document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);

**</script>**

**Chapter 4**

**DOCUMENT OBJECT MODEL :**

# JavaScript HTML DOM :

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**:

### The HTML DOM 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

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

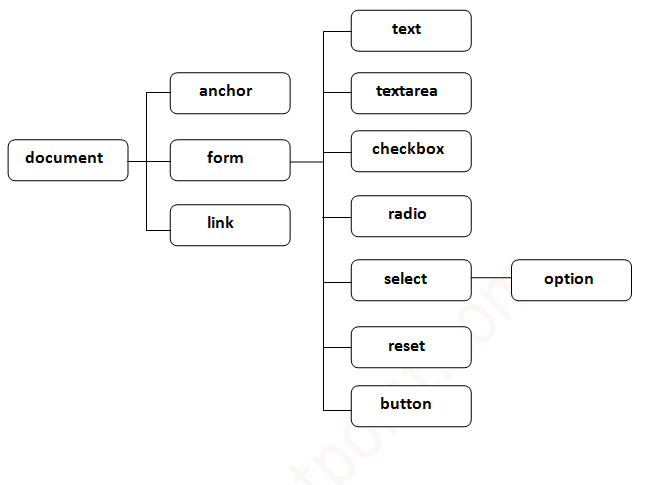
What is the HTML DOM?

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

## Properties of document object :

Let's see the properties of document object that can be accessed and modified by the document object.



## Methods of document object :

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| write("string") | writes the given string on the doucment. |
| writeln("string") | writes the given string on the doucment with newline character at the end. |
| getElementById() | returns the element having the given id value. |
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name. |

### Accessing the field value by document object :

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

**form1** is the name of the form.

**name** is the attribute name of the input text.

**value** is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

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

function printvalue(){

var name=document.form1.name.value;

alert("Welcome: "+name);

}

**</script>**

**<form** name="form1"**>**

Enter Name:**<input** type="text" name="name"**/>**

**<input** type="button" onclick="printvalue()" value="print name"**/>**

**</form>**

# Javascript - document.getElementById() method :

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

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

function getcube(){

var number=document.getElementById("number").value;

alert(number\*number\*number);

}

**</script>**

**<form>**

Enter No:**<input** type="text" id="number" name="number"**/><br/>**

**<input** type="button" value="cube" onclick="getcube()"**/>**

**</form>**

# Javascript - document.getElementsByName() method :

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

document.getElementsByName("name")   Here, name is required.

### Example of document.getElementsByName() method :

### In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

### <script type="text/javascript">

### function totalelements()

{

var allgenders=document.getElementsByName("gender");

alert("Total Genders:"+allgenders.length);

}

**</script>**

**<form>**

Male:**<input** type="radio" name="gender" value="male"**>**

Female:**<input** type="radio" name="gender" value="female"**>**

**<input** type="button" onclick="totalelements()" value="Total Genders"**>**

**</form>**

# Javascript -document.getElementsByTagName() method :

The document.getElementsByTagName() method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

1. document.getElementsByTagName("name")   Here, name is required.

### Example of document.getElementsByTagName() method :

### In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

<script type="text/javascript">

function countpara(){

var totalpara=document.getElementsByTagName("p");

alert("total p tags are: "+totalpara.length);

}

</script>

<p>This is a pragraph</p>

<p>Here we are going to count total number of paragraphs by getElementByTagName() m

ethod.</p>

<p>Let's see the simple example</p>

<button onclick="countpara()">count paragraph</button>

### Another example of document.getElementsByTagName() method :

In this example, we going to count total number of h2 and h3 tags used in the document.

<script type="text/javascript">

function counth2(){  var totalh2=document.getElementsByTagName("h2");  alert("total h2 tags a

re: "+totalh2.length);  }

function counth3(){

var totalh3=document.getElementsByTagName("h3");

alert("total h3 tags are: "+totalh3.length);

}

</script>

<h2>This is h2 tag</h2>

<h2>This is h2 tag</h2>

<h3>This is h3 tag</h3>

<h3>This is h3 tag</h3>

<h3>This is h3 tag</h3>

<button onclick="counth2()">count h2</button>

<button onclick="counth3()">count h3</button>

# Javascript – innerHTML :

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

### Example of innerHTML property :

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

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

function showcommentform() {  var data="Name:<input type='text' name='name'><br>c

omment:<textarea rows='5' cols='80'></textarea><br><input type='submit' value='com

ment'>";

document.getElementById('mylocation').innerHTML=data;

}

**</script>**

**<form** name="myForm"**>**

**<input** type="button" value="comment" onclick="showcommentform()"**>**

**<div** id="mylocation"**></div>**

**</form>**

# Javascript – innerText :

# The innerText property can be used to write the dynamic text on the html document. Here,

# text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

### Example of innerText property :

In this example, we are going to display the password strength when releases the key after press.

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

function validate() {

var msg;

if(document.myForm.userPass.value.length**>**5){

msg="good";

}

else{

msg="poor";

}

document.getElementById('mylocation').innerText=msg;

 }

**</script>**

**<form** name="myForm"**>**

**<input** type="password" value="" name="userPass" onkeyup="validate()"**>**

Strength:**<span** id="mylocation"**>**no strength**</span>**

**</form>**

**CHAPTER 5**

**JAVASCRIPT EVENTS :**

## What is an Event ?

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

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

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](http://www.tutorialspoint.com/html/html_events_ref.htm). Here we will see a few examples to understand a relation between Event and JavaScript −

## onclick Event Type :

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

### Example

Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello" />

</form>

</body>

</html>

## onsubmit Event type :

**onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

### Example

The following example shows how to use onsubmit. Here we are calling a**validate()** function before submitting a form data to the webserver. If**validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function validation() {

all validation goes here

.........

return either true or false

}

//-->

</script>

</head>

<body>

<form method="POST" action="t.cgi" onsubmit="return validate()">

<input type="submit" value="Submit" />

</form>

</body>

</html>

## onmouseover and onmouseout :

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

<html>

<head>

<script type="text/javascript">

<!--

function over() {

document.write ("Mouse Over");

}

function out() {

document.write ("Mouse Out");

}

//-->

</script>

</head>

<body>

<p>Bring your mouse inside the division to see the result:</p>

<div onmouseover="over()" onmouseout="out()">

<h2> This is inside the division </h2>

</div>

</body>

</html>

## HTML 5 Standard Events :

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

**CHAPTER 6 :**

**JAVASCRIPT EVENTS AND EVENT LISTENERS :**

# JavaScript: Events and Listeners :

# Although you can start functions when your page loads, many times you'll want to start functions when a user clicks a link, enters a form, scrolls, moves his or her mouse over an object, or does something else. These actions are called events. You can set specific functions to run when the user performs an event. These functions "listen" for an event and then initiate the function.

## Common Event Listeners :

The following are common events:

onload //when the page loads

onclick //when a user clicks something

onmouseover //when a user mouses over something

onfocus //when a user puts the cursor in a form field

onblur //When a user leaves a form field

Note that each event listener begins with on and is entirely lowercase.

## Adding Event Listeners :

You can add an event listener directly in the HTML code in the following way:

**<**button onclick**=**"alert('Hello, world');"**><**/button>

However, mixing the JavaScript code with the HTML directly is generally a poor practice for the same reason that you don't integrate style commands into HTML but rather reference the styles in a separate CSS file.

A more common way to integrate the event listeners is by identifying the element and adding the event listener as a method. The general format for doing this is as follows:

myelement.onclick **=** **function**() {

*//run your event handler code...*

};

In this example, myelement would be a variable that refers to a specific style on your page.

Note that the word function is written without a name because its code is specified in the code block that immediately follows.

Also note that the statement ends with a semicolon ; after the closing curly brace }.

## Associating Events with IDs :

If you have a specific ID in your web page that uniquely identifies a section, you can associate an event with that ID. Here's an example:

//set a variable to refer to a specific ID

**var** specialSection **=** document.getElementById("specialSection");

//initiate this function when the user clicks the ID

specialSection.onclick **=** **function**() {

alert("Hello World")

};

## Adding the addEventListener Method :

You can also add event listeners using a method called addEventListener. However, this method isn't supported in Internet Explorer 8, so if you use this method, you need to add some conditional functions to check for browser functionality before running the function.

Some JavaScript libraries, like jQuery, automatically include checks for cross-browser functionality.

The format for adding events using this method is as follows:

document.addEventListener('click', myfunction, **false**);

In the above format, you start by adding the method to the Document object. In the parentheses, list the event listener but without the on. (In this example, the event listener is onclick, which is shortened to click.) You then add the function to run. (The function is declared elsewhere.) Thefalse refers to an advanced, rarely used parameter for event handling.

## Referencing Event Handlers :

If your event handler refers to a specific element on the page, and you call your script before the element loads, the script won't work because the element isn't available at the time the script runs.

You could insert the script before the closing body tag to ensure the elements load first, but let's say you need your script to run earlier — before the element loads. You can use the prepareEventHandlersfunction to load the function when the page loads.

First, add the prepareEventHandlers function:

*//declare the function*

**function** prepareEventHandlers() {

*//get a specific page ID and assign it as a variable*

**var** specialSection **=** document.getElementById("specialSection");

*//initiate this function when the ID is clicked*

specialSection.onclick **=** **function**() {

alert("Hello World");

}

}

Now call the prepareEventHandlers function when the window loads:

window.onload **=** **function**() {

prepareEventHandlers();

}

Using this technique, the event will be loaded and ready even if it is referenced before the element it's listening for. Note that you can call the window.onload function only once per page.

## onblur and onfocus Events :

onblur and onfocus events refer to the way users interact with forms. When a user clicks a form field, an onfocus event occurs. When a user clicks out of the field, an onblur event occurs.

Here's an example. Let's say that your HTML form has an ID called namefield:

**<**input type**=**"text" value**=**"your name" name**=**"name"

id**=**"namefield" tabindex**=**"10" **/>**

First, get the namefield element, and then add an event to it.

//get the ID and assign it as a variable

**var** nameField **=** document.getElementbyId("name");

//when the ID is clicked, run this function

nameField.onfocus **=** **function**() {

//if the field has the value "your name", leave it blank

**if** ( nameField.value **==** "your name" ) {

nameField.value **=** "";

}

};

You can also add an onblur event to take place when the user leaves the field.

//get the ID and assign it as a variable

**var** nameField **=** document.getElementbyId("name");

//when the user leaves the ID, run this function

nameField.onblur **=** **function**() {

//if the field's value is "your name", don't write anything

**if** ( nameField.value **==** "your name" ) {

emailField.value **=** "";

}

};

## Timers :

You can add events that have a time delay. For example, if you want a pop-up message to appear after the user has been on the page for 10 seconds, you can do this through the setTimeOut method.

For example, let's say you have a simple message:

**function** welcomeVisitor() {

alert("Welcome to the site");

}

However, you don't want this message to appear until the user has been on the page for a while. You delay the action through the setTimeOut function:

setTimeOut(welcomeVisitor,8000);

The setTimeOut function has two parameters. The first allows you to call a function (which we declared earlier), and the second refers to the timer. The function will not be called for 8000 milliseconds.

There are other timer functions, such as setInterval, that initiate the function at the interval you specify