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

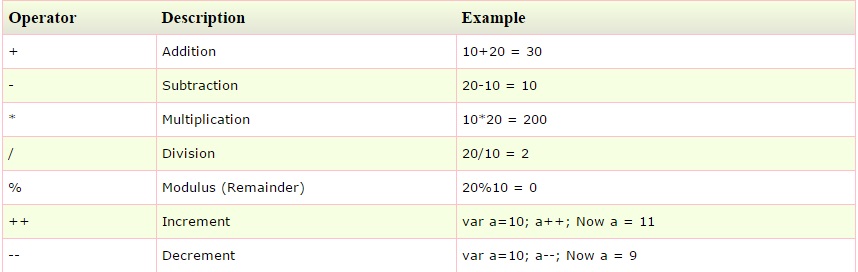
operators are known as JavaScript arithmetic operators.

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**

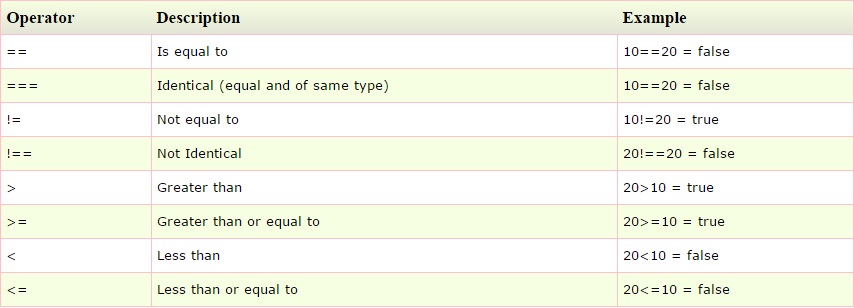




**JavaScript Comparison Operator:**

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:**

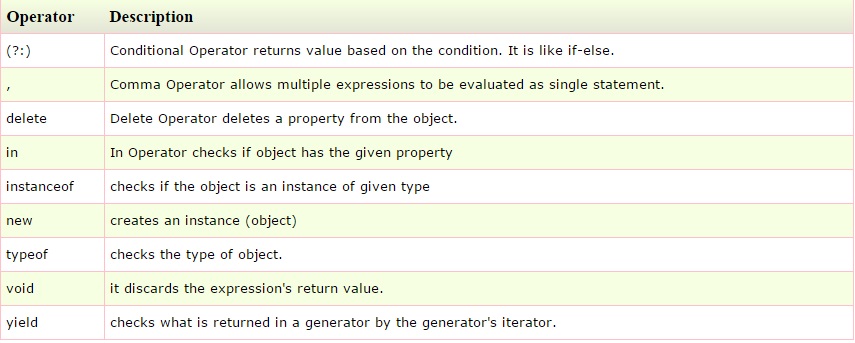
JavaScript supports the following assignment operators –

|  |  |
| --- | --- |
| **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

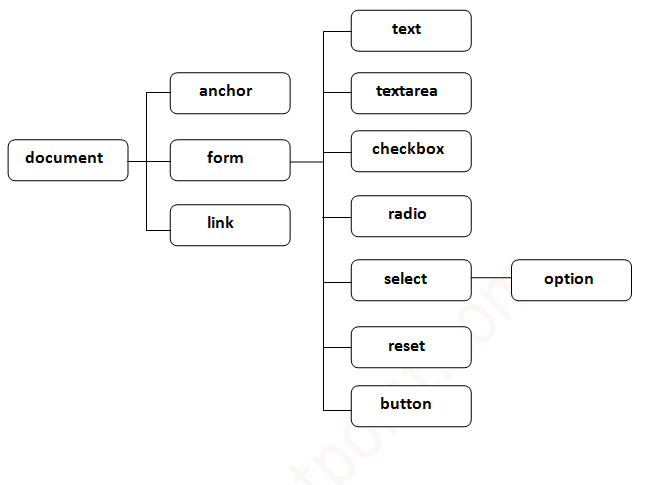
What is the HTML DOM?

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.

**CHAPTER 7**

# JavaScript – Debugging :

Every now and then, developers commit mistakes while coding. A mistake in a program or a script is referred to as a bug.

The process of finding and fixing bugs is called debugging and is a normal part of the development process. This section covers tools and techniques that can help you with debugging tasks..

## Error Messages in IE :

The most basic way to track down errors is by turning on error information in your browser. By default, Internet Explorer shows an error icon in the status bar when an error occurs on the page.

Double-clicking this icon takes you to a dialog box showing information about the specific error that occurred.

Since this icon is easy to overlook, Internet Explorer gives you the option to automatically show the Error dialog box whenever an error occurs.

To enable this option, select Tools → Internet Options → Advanced tab. and then finally check the "Display a Notification About Every Script Error" box option as shown below −

**Error Messages in Firefox or Mozilla :**

Other browsers like Firefox, Netscape, and Mozilla send error messages to a special window called the JavaScript Console or Error Consol. To view the console, select Tools → Error Consol or Web Development.

Unfortunately, since these browsers give no visual indication when an error occurs, you must keep the Console open and watch for errors as your script executes.

## Error Notifications :

Error notifications that show up on Console or through Internet Explorer dialog boxes are the result of both syntax and runtime errors. These error notification include the line number at which the error occurred.

If you are using Firefox, then you can click on the error available in the error console to go to the exact line in the script having error.

## How to debug a Script :

There are various ways to debug your JavaScript −

### Use a JavaScript Validator :

One way to check your JavaScript code for strange bugs is to run it through a program that checks it to make sure it is valid and that it follows the official syntax rules of the language. These programs are called **validating parsers** or just **validators** for short, and often come with commercial HTML and JavaScript editors.

The most convenient validator for JavaScript is Douglas Crockford's JavaScript Lint, which is available for free at [Douglas Crockford's JavaScript Lint](http://www.jslint.com/).

Simply visit that web page, paste your JavaScript (Only JavaScript) code into the text area provided, and click the jslint button. This program will parse through your JavaScript code, ensuring that all the variable and function definitions follow the correct syntax. It will also check JavaScript statements, such as if and **while,** to ensure they too follow the correct format

### Add Debugging Code to Your Programs

You can use the **alert()** or **document.write()** methods in your program to debug your code. For example, you might write something as follows −

var debugging = true;

var whichImage = "widget";

if( debugging )

alert( "Calls swapImage() with argument: " + whichImage );

var swapStatus = swapImage( whichImage );

if( debugging )

alert( "Exits swapImage() with swapStatus=" + swapStatus );

By examining the content and order of the alert() as they appear, you can examine the health of your program very easily.

### Use a JavaScript Debugger :

### A debugger is an application that places all aspects of script execution under the control of the

### programmer. Debuggers provide fine-grained control over the state of the script through an

### interface that allows you to examine and set values as well as control the flow of execution.

Once a script has been loaded into a debugger, it can be run one line at a time or instructed tohalt at certain breakpoints. Once execution is halted, the programmer can examine the state of the script and its variables in order to determine if something is amiss. You can also watch variables for changes in their values.

The latest version of the Mozilla JavaScript Debugger (code-named Venkman) for both Mozilla and Netscape browsers can be downloaded at <http://www.hacksrus.com/~ginda/venkman>

**CHAPTER 8 :**

**BUILDING SMARTER FORMS :**

# JavaScript Form Validation :

# It is important to validate the form submitted by the user because it can have inappropriate

# values. So validation is must.

# The JavaScript provides you the facility the validate the form on the client side so processing

# will be fast than server-side validation. So, most of the web developers prefer JavaScript form

# validation.

# Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

## JavaScript form validation example :

## In this example, we are going to validate the name and password. The name can’t be empty and

## password can’t be less than 6 characters long.

## Here, we are validating the form on form submit. The user will not be forwarded to the next page

## until given values are correct.

## <script>

## function validateform(){

## var name=document.myform.name.value;

## var password=document.myform.password.value;

## if (name==null || name==""){

## alert("Name can't be blank");

## return false;

## }else if(password.length<6){

## alert("Password must be at least 6 characters long.");

## return false;

## }

## }

## </script>

## <body>

## <form name="myform" method="post" action="abc.jsp" onsubmit="return valida

## eform()>

## Name: <input type="text" name="name"><br/>

## Password: <input type="password" name="password"><br/>

## <input type="submit" value="register">

</form>

## JavaScript Retype Password Validation :

## <script type="text/javascript">

## function matchpass(){

## var firstpassword=document.f1.password.value;

## var secondpassword=document.f1.password2.value;

## if(firstpassword==secondpassword){

## return true;

## }

## else{

## alert("password must be same!");

## return false;

## }

## }

## </script>

## <form name="f1" action="register.jsp" onsubmit="return matchpass()">

## Password:<input type="password" name="password" /><br/>

## Re-enter Password:<input type="password" name="password2"/><br/>

## <input type="submit">

</form>

## JavaScript Number Validation :

## Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

## <script>

## function validate(){

## var num=document.myform.num.value;

## if (isNaN(num)){

## document.getElementById("numloc").innerHTML="Enter Numeric value only";

## return false;

## }else{

## return true;

## }

## }

## </script>

## <form name="myform" onsubmit="return validate()" >

## Number: <input type="text" name="num"><span id="numloc"></span><br/>

## <input type="submit" value="submit">

## </form>

## JavaScript validation with image :

## Let’s see an interactive JavaScript form validation example that displays correct and incorrect

## image if input is correct or incorrect.

## <script>

## function validate(){

## var name=document.f1.name.value;

## var password=document.f1.password.value;

## var status=false;

## if(name.length<1){

## document.getElementById("nameloc").innerHTML=  "<img src='unchecked.gif'/> Please enter your name";

## status=false;  }else{  document.getElementById("nameloc").innerHTML="<img sr

## c='checked.gif'/>";

## status=true;

## }

## if(password.length<6){  document.getElementById("passwordloc").innerHTML="

## <img src='unchecked.gif'/> Password must be at least 6 char long";  status=false

## ;  }else{  document.getElementById("passwordloc").innerHTML="<img src='chec

## ked.gif'/>";  status=true;  }  return status;  }

</script>

<form name="f1" action="#" onsubmit="return validate()">

<table>

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

<span id="nameloc"></span></td></tr>

<tr><td>Enter Password:</td><td><input type="text" name="password"/>

<span id="passwordloc"></span></td></tr>

<tr><td colspan="2"><input type="submit" value="register"/></td></tr>

</table>

</form>

## JavaScript email validation :

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

* email id must contain the @ and . character
* There must be at least one character before and after the @.
* There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

**<script>**

function validateemail()

{

var x=document.myform.email.value;

var atposition=x.indexOf("@");

var dotposition=x.lastIndexOf(".");

if (atposition**<1** || dotposition**<atposition**+2 || dotposition+2**>**=x.length){

 alert("Please enter a valid email address \n atpostion:"+atposition+"\n dotposition:"+dotp

sition);

return false;

  }

}

**</script>**

**<body>**

**<form** name="myform"  method="post" action="#" onsubmit="return validateemail();"**>**

Email: **<input** type="text" name="email"**><br/>**

**<input** type="submit" value="register"**>**

**</form>**

**CHAPTER 9**

**UI ENHANCEMENT :**

# What is CSS?

**C**ascading **S**tyle **S**heets, fondly referred to as CSS, is a simple design language intended to

simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the

text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out,

what background images or colors are used, layout designs,variations in display for different

devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an

HTML document. Most commonly, CSS is combined with the markup languages HTML or

XHTML.

## Advantages of CSS :

* **CSS saves time** − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
* **Pages load faster** − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
* **Easy maintenance** − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
* **Superior styles to HTML** − CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* **Multiple Device Compatibility** − Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
* **Global web standards** − Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.
* **Offline Browsing** − CSS can store web applications locally with the help of an offline catche.Using of this, we can view offline websites.The cache also ensures faster loading and better overall performance of the website.
* **Platform Independence** − The Script offer consistent platform independence and can support latest browsers as well.

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts −

* **Selector** − A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
* **Property** - A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be*color*, *border* etc.
* **Value** - Values are assigned to properties. For example, *color* property can have value either *red* or *#F1F1F1* etc.

You can put CSS Style Rule Syntax as follows –

selector { property: value }

**Example:**

You can define a table border as follows – table{ border :1px solid #C00; }

Here table is a selector and border is a property and given value *1px solid #C00*is the value of that property.

# JavaScript HTML DOM - Changing CSS :

## Changing HTML Style :

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property*=*new style*

The following example changes the style of a <p> element:

**Example :**

<html>

<body>

<p id="p2">Hello World!</p>

<script>

document.getElementById("p2").style.color = "blue";

</script>

<p>The paragraph above was changed by a script.</p>

</body>

</html>

# HTML DOM className Property :

The className property sets or returns the class name of an element (the value of an element's

class attribute).

Syntax :

Return the className property:

HTMLElementObject.className

Set the className property:

HTMLElementObject.className=class

Example :

<!DOCTYPE html>

<html>

<head>

<style>

.mystyle {

width: 300px;

height: 100px;

background-color: coral;

text-align: center;

font-size: 25px;

color: white;

margin-bottom: 10px;

}

</style>

</head>

<body>

<p>Click the button to set a class for div.</p>

<div id="myDIV">

I am a DIV element

</div>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction() {

document.getElementById("myDIV").className = "mystyle";

}

</script

</body>

</html>

# Inline Styles :

# In React, inline styles are not specified as a string. Instead they are specified with an object whose key is the camelCased version of the style name, and whose value is the style's value, usually a string ([more on that later](https://facebook.github.io/react/tips/style-props-value-px.html)):

var divStyle = {

color: 'white',

backgroundImage: 'url(' + imgUrl + ')',

WebkitTransition: 'all', // note the capital 'W' here

msTransition: 'all' // 'ms' is the only lowercase vendor prefix

};

ReactDOM.render(<div style={divStyle}>Hello World!</div>, mountNode);

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes

from JS (e.g. node.style.backgroundImage).

**CHAPTER 10**

**JAVASCRIPT BEST PRACTICES :**

# JavaScript Best Practices :

Avoid global variables,  avoid new,  avoid  ==,  avoid eval()

## Avoid Global Variables :

Minimize the use of global variables.

This includes all data types, objects, and functions.

Global variables and functions can be overwritten by other scripts.

Use local variables instead, and learn how to use [closures](http://www.w3schools.com/js/js_function_closures.asp).

## Always Declare Local Variables :

All variables used in a function should be declared as **local** variables.

Local variables **must** be declared with the **var**keyword, otherwise they will become global

variables.

## Declarations on Top :

It is a good coding practice to put all declarations at the top of each script or function.

This will:

* Give cleaner code
* Provide a single place to look for local variables
* Make it easier to avoid unwanted (implied) global variables
* Reduce the possibility of unwanted re-declarations

// Declare at the beginning

var firstName, lastName, price, discount, fullPrice;

// Use later

firstName = "John";

lastName = "Doe";

price = 19.90;

discount = 0.10;

fullPrice = price \* 100 / discount;

This also goes for loop variables:

// Declare at the beginning

var i;

// Use later

for (i = 0; i < 5; i++) {

}

## Initialize Variables :

It is a good coding practice to initialize variables when you declare them.

This will:

* Give cleaner code
* Provide a single place to initialize variables
* Avoid undefined values

// Declare and initiate at the beginning

var firstName = "",

   lastName = "",

price = 0,

    discount = 0,

    fullPrice = 0,

myArray = [],

  myObject = {};

## Never Declare Number, String, or Boolean Objects :

Always treat numbers, strings, or booleans as primitive values. Not as objects.

Declaring these types as objects, slows down execution speed, and produces

nasty side effects:

### Example

### var x = "John";

### var y = new String("John");

### (x === y) // is false because x is a string and y is an object.

Or even worse:

### Example

### var x = new String("John");              var y = new String("John");

### (x == y) // is false because you cannot compare objects

### Don't Use new Object() :

* Use {} instead of new Object()
* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new function()

### Example

var x1 = {};           // new object

var x2 = "";           // new primitive string

var x3 = 0;            // new primitive number

var x4 = false;        // new primitive Boolean

var x5 = [];           // new array object

var x6 = /()/;         // new regexp object

var x7 = function(){}; // new function object

**Beware of Automatic Type Conversions :**

Beware that numbers can accidentally be converted to strings or NaN (Not a Number).

JavaScript is loosely typed. A variable can contain different data types, and a variable can

change its data type:

### Example

var x = "Hello";     // typeof x is a string

x = 5;               // changes typeof x to a number

When doing mathematical operations, JavaScript can convert numbers to strings:

### Example

var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number

var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string

var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string

var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number

var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number

var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number

var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number

Subtracting a string from a string, does not generate an error but returns NaN (Not a Number):

### Example

"Hello" - "Dolly"    // returns NaN

**Use === Comparison :**

The == comparison operator always converts (to matching types) before comparison.

The === operator forces comparison of values and type:

### Example

0 == "";        // true

1 == "1";       // true

1 == true;      // true

0 === "";       // false

1 === "1";      // false

1 === true;     // false

## Use Parameter Defaults :

If a function is called with a missing argument, the value of the missing argument is set to **undefined**.

Undefined values can break your code. It is a good habit to assign default values to arguments.

### Example

function myFunction(x, y) {

 if (y === undefined)

{

        y = 0;

   }

}

## End Your Switches with Defaults

Always end your switch statements with a default. Even if you think there is no need for it.

**Avoid Using eval()**

The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.

**MINFICATION :**

**What is Javascript minification?**

Minification in Javascript is the process of removing all characters that are not necessary from

The Javascript source code. That is why it is called “minification” – because all of the data that is

Not necessary to the functioning of the Javascript is removed from the source code, and therefore

The Javascript is “minimized”. Even though these characters are removed from the Javascript

source code, the functionality of the Javascript code does not change at all. So, your Javascript

code will behave exactly the same even after it goes through the minification process. Code that

has gone through the minification process is also known as “minified” code.

**What data is removed in Javascript minification?**

The data that is removed in Javascript minification is extra whitespace, comments, new line

characters, etc. Basically anything that can be removed without compromising the functionality

of the Javascript is removed.

**How are Javascript files minified?**

There are a lot of programs that will minify your Javascript code for free – all you have to do is

simply google “Javascript minifier”, and these sites will simply ask you to submit your

Javascript code and will return the minified result.

**What are the benefits and advantages of Javascript minification?**

The main purpose of Javascript minification is to speed up the downloading or transfer of the

Javascript code from the server hosting the website’s Javascript. The reason that minification

makes downloads go faster is because it reduces the amount of data (in the minified Javascript

file) that needs to be downloaded. Less data means that the user’s browser spends less time

processing that data, which is why time is saved. So, we can say that minification is performed

on Javascript source code because it is essentially a performance enhancement – and it allows

websites that use minified Javascript to load faster.

To summarize, some of the benefits of Javascript minification are:

* If you are a website owner, your users will have quicker download times for your

webpages because there is now less data in your Javascript files that needs to be

downloaded.

* Also, if you are a website owner, your site will have less bandwidth consumption. This

means that your servers will consume less energy and you will save money. If your site

just has a few visitors everyday, the bandwidth savings will be meaningless. But, think of

a large site like Facebook, which gets hundreds of millions of users each day, and how

Javascript minification could save a lot of bandwidth for them.

**Minification is NOT like normal data compression :**

Normally when you think of data compression, one of the first things that may come to mind is

“zipping” a file. When you zip a file, you are compressing it using a special algorithm, and when

someone else has to unzip that file, they have to decompress the file using a special program

(which you may have used without knowing it since it comes with most operating systems built

in). However, with Javascript minification, no special program is needed by the browser in order

to understand how to read the minified Javascript – because minified Javascript is still readable

by the browser as is. A human can even read it – although with great difficulty since there won’t

be any whitespace and newline characters. But, the point is that no intermediate program is

needed in order to convert the minified Javascript into a form that is readable by the browser, and

for that reason it is very different from normal data compression.

**To minify JavaScript,** try the [Closure Compiler](https://developers.google.com/closure/compiler), [JSMin](http://www.crockford.com/javascript/jsmin.html) or the [YUI Compressor](http://developer.yahoo.com/yui/compressor/). You can create

a build process that uses these tools to minify and rename the development files and save them to

a production directory.

**JAVASCRIPT CODE CHECKER :**

## JSLint :

## JSLint is the oldest of the four. Douglas Crockford created it in 2002 to enforce what, in his

## experience, are the good parts of JavaScript. If you agree with the good parts, JSLint can be a

## good tool—you install it and it’s ready to go.

## The downsides are that JSLint is not configurable or extensible. You can’t disable many features

## at all, and some of them lack documentation.

## The official website is not very helpful, for example it lacks any information on how to integrate

## it with your editor.

## Pros

* Comes configured and ready to go (if you agree with the rules it enforces)

### Cons :

* JSLint doesn’t have a configuration file, which can be problematic if you need to change the settings
* Limited number of configuration options, many rules cannot be disabled
* You can’t add custom rules
* Undocumented features
* Difficult to know which rule is causing which error

## JSHint :

## JSHint was created as a more configurable version of JSLint (of which it is a fork). You can

## configure every rule, and put them into a configuration file, which makes JSHint easy to use in

## bigger projects. JSHint also has good documentation for each of the rules, so you know exactly

## what they do. Integrating it into editors is also simple.

## A small downside to JSHint is that it comes with a relaxed default configuration. This means you

## need to do some setup to make it useful. When comparing it with ESLint, it’s also more difficult

## to know which rules you need to change in order to enable or disable certain error messages.

## Pros :

* Most settings can be configured
* Supports a configuration file, making it easier to use in larger projects
* Has support for many libraries out of the box, like jQuery, QUnit, NodeJS, Mocha, etc.
* Basic ES6 support

Cons :

* Difficult to know which rule is causing an error
* Has two types of option: enforcing and relaxing (which can be used to make JSHint stricter, or to suppress its warnings). This can make configuration slightly confusing
* No custom rule support

**CHAPTER 11 :**

# JAVASCRIPT LIBRARIES :

JavaScript libraries - jQuery, Prototype, MooTools.

## JavaScript Frameworks (Libraries)

## Advanced JavaScript programming (especially the complex handling of browser differences),

## can often be very difficult and time-consuming to work with.

## To deal with these difficulties, a lot of ****JavaScript (helper) libraries**** have been developed.

## These JavaScript libraries are often called **JavaScript frameworks**.

In this tutorial, we will take a look at some of the most popular JavaScript

frameworks:

* jQuery
* Prototype
* MooTools

All of these frameworks have functions for common JavaScript tasks like animations, DOM

manipulation, and Ajax handling.

In this tutorial we will teach you how start using them, to make JavaScript programming easier,

safer, and much more exciting.

## jQuery :

## **jQuery** is the most popular JavaScript framework on the Internet today.

## It uses CSS selectors to access and manipulate HTML elements (DOMObjects) on a web page.

## jQuery also provides a companion UI (user interface) framework and numerous other plug-ins.

Many of the largest companies on the Web use jQuery:

* Google
* Microsoft
* IBM
* Netflix

You will find an excellent [jQuery Tutorial](http://www.w3schools.com/jquery/default.asp) here at W3Schools.

**Prototype :**

**Prototype** is a JavaScript library that provides a simple API to perform common web tasks.

**API** is short for Application Programming Interface. It is a library of properties and methods for

manipulating the HTML DOM.

Prototype enhances JavaScript by providing classes and inheritance.

**MooTools :**

**MooTools** is also a framework that offers an API to make common JavaScript programming

easier.

MooTools also includes some lightweight effects and animation functions.

## Other Frameworks :

Here are some other frameworks not covered in this short overview:

**YUI** - The Yahoo! User Interface Framework is a large library that covers a lot of functions,

from simple JavaScript utilities to complete internet widgets.

**Ext JS** - Customizable widgets for building rich Internet applications.

**Dojo** - A toolkit designed around packages for DOM manipulation, events, widgets, and more.

**script.aculo.us** - Open-source JavaScript framework for visual effects and interface behaviors.

**UIZE** - Widgets, AJAX, DOM, templates, and more.

## CDN - Content Delivery Networks :

You always want your web pages to be as fast as possible. You want to keep the size of your

pages as small as possible, and you want the browser to cache as much as possible.

If many different web sites use the same JavaScript framework, it makes sense to host the

framework library in a common location for every web page to share.

A CDN (Content Delivery Network) solves this. A CDN is a network of servers containing

shared code libraries.

Google provides a free CDN for a number of JavaScript libraries, including:

* jQuery
* Prototype
* MooTools
* Dojo
* Yahoo! YUI

To use a JavaScript framework library in your web pages, just include the library in a <script>

tag:

### Including jQuery

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>

## Using Frameworks :

## Before you decide to use a JavaScript framework for your web pages, it might be a good idea to

## test the framework first.

## JavaScript frameworks are very easy to test. You don't have to install them on your computer,

## and there are no setup programs.

## Normally you just have to reference a library file from your web page.

## In the next chapter of this tutorial we will walk you through a test process for jQuery.

**Chapter 12:**

**JAVASCRIPT AND HTML5 :**

# HTML5 :

# HTML5 is the next major revision of the HTML standard superseding HTML 4.01,

# XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

# HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

# The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

# Browser Support :

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support

many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5

functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all

have excellent support for HTML5.

## New Features :

## HTML5 introduces a number of new elements and attributes that helps in building a modern

website. Following are great features introduced in HTML5.

* **New Semantic Elements** − These are like <header>, <footer>, and <section>.
* **Forms 2.0** − Improvements to HTML web forms where new attributes have been introduced for <input> tag.
* **Persistent Local Storage** − To achieve without resorting to third-party plugins.
* **WebSocket** − A a next-generation bidirectional communication technology for web applications.
* **Server-Sent Events** − HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
* **Canvas** − This supports a two-dimensional drawing surface that you can program with JavaScript.
* **Audio & Video** − You can embed audio or video on your web pages without resorting to third-party plugins.
* **Geolocation** − Now visitors can choose to share their physical location with your web application.
* **Microdata** − This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
* **Drag and drop** − Drag and drop the items from one location to another location on a the same webpage.

# JavaScript Use Strict :

**"use strict";**

Defines that JavaScript code should be executed in "strict mode".

**The "use strict" Directive :**

The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5).

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

The purpose of "use strict" is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

## Declaring Strict Mode :

Strict mode is declared by adding "use strict"; to the beginning of a JavaScript or a JavaScript function.

Declared at the beginning of a JavaScript file, it has global scope (all code will execute in strict mode):

Example :

<!DOCTYPE html>

<html>

<body>

<h2>Global "use strict" declaration.</h2>

<p>Activate debugging in your browser (F12) to see the error report.</p>

<script>

"use strict";

myFunction();

function myFunction() {

y = 3.14; // This will cause an error (y is not defined)

}

</script>

</body>

</html>

**CHAPTER 13**

**ADVANCED JAVASCRIPT FEATURES :**

## RegExp Object :

## A regular expression is an object that describes a pattern of characters.

## Regular expressions are used to perform pattern-matching and

## "search-and-replace" functions on text.

## Syntax

/pattern/modifiers;

Example

var patt = /w3schools/i

Example explained:

* **/w3schools/i**  is a regular expression.
* **w3schools**  is a pattern (to be used in a search).
* **i**  is a modifier (modifies the search to be case-insensitive).

For a tutorial about Regular Expressions, read our [JavaScript RegExp Tutorial](http://www.w3schools.com/js/js_regexp.asp).

**Modifiers :**

**Modifiers are used to perform case-insensitive and global searches:**

|  |  |
| --- | --- |
| Modifier | Description |
| [i](http://www.w3schools.com/jsref/jsref_regexp_i.asp) | Perform case-insensitive matching |
| [g](http://www.w3schools.com/jsref/jsref_regexp_g.asp) | Perform a global match (find all matches rather than stopping after the first match) |
| [m](http://www.w3schools.com/jsref/jsref_regexp_m.asp) | Perform multiline matching |

## Brackets :

**Brackets are used to find a range of characters:**

|  |  |
| --- | --- |
| Expression | Description |
| [[abc]](http://www.w3schools.com/jsref/jsref_regexp_charset.asp) | Find any character between the brackets |
| [[^abc]](http://www.w3schools.com/jsref/jsref_regexp_charset_not.asp) | Find any character NOT between the brackets |
| [[0-9]](http://www.w3schools.com/jsref/jsref_regexp_0-9.asp) | Find any digit between the brackets |
| [[^0-9]](http://www.w3schools.com/jsref/jsref_regexp_not_0-9.asp) | Find any digit NOT between the brackets |
| [(x|y)](http://www.w3schools.com/jsref/jsref_regexp_xy.asp) | Find any of the alternatives specified |

## Metacharacters :

Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| Metacharacter | Description |
| [.](http://www.w3schools.com/jsref/jsref_regexp_dot.asp) | Find a single character, except newline or line terminator |
| [\w](http://www.w3schools.com/jsref/jsref_regexp_wordchar.asp) | Find a word character |
| [\W](http://www.w3schools.com/jsref/jsref_regexp_wordchar_non.asp) | Find a non-word character |
| [\d](http://www.w3schools.com/jsref/jsref_regexp_digit.asp) | Find a digit |
| [\D](http://www.w3schools.com/jsref/jsref_regexp_digit_non.asp) | Find a non-digit character |
| [\s](http://www.w3schools.com/jsref/jsref_regexp_whitespace.asp) | Find a whitespace character |
| [\S](http://www.w3schools.com/jsref/jsref_regexp_whitespace_non.asp) | Find a non-whitespace character |
| [\b](http://www.w3schools.com/jsref/jsref_regexp_begin.asp) | Find a match at the beginning/end of a word |
| [\B](http://www.w3schools.com/jsref/jsref_regexp_begin_not.asp) | Find a match not at the beginning/end of a word |
| [\0](http://www.w3schools.com/jsref/jsref_regexp_nul.asp) | Find a NUL character |
| [\n](http://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| [\f](http://www.w3schools.com/jsref/jsref_regexp_formfeed.asp) | Find a form feed character |
| [\r](http://www.w3schools.com/jsref/jsref_regexp_carriagereturn.asp) | Find a carriage return character |
| [\t](http://www.w3schools.com/jsref/jsref_regexp_tab.asp) | Find a tab character |
| [\v](http://www.w3schools.com/jsref/jsref_regexp_vtab.asp) | Find a vertical tab character |
| [\xxx](http://www.w3schools.com/jsref/jsref_regexp_octal.asp) | Find the character specified by an octal number xxx |
| [\xdd](http://www.w3schools.com/jsref/jsref_regexp_hex.asp) | Find the character specified by a hexadecimal number dd |
| [\uxxxx](http://www.w3schools.com/jsref/jsref_regexp_unicode_hex.asp) | Find the Unicode character specified by a hexadecimal number xxxx |

## Quantifiers :

|  |  |
| --- | --- |
| Quantifier | Description |
| [n+](http://www.w3schools.com/jsref/jsref_regexp_onemore.asp) | Matches any string that contains at least one n |
| [n\*](http://www.w3schools.com/jsref/jsref_regexp_zeromore.asp) | Matches any string that contains zero or more occurrences of n |
| [n?](http://www.w3schools.com/jsref/jsref_regexp_zeroone.asp) | Matches any string that contains zero or one occurrences of n |
| [n{X}](http://www.w3schools.com/jsref/jsref_regexp_nx.asp) | Matches any string that contains a sequence of *X* *n*'s |
| [n{X,Y}](http://www.w3schools.com/jsref/jsref_regexp_nxy.asp) | Matches any string that contains a sequence of X to Y *n*'s |
| [n{X,}](http://www.w3schools.com/jsref/jsref_regexp_nxcomma.asp) | Matches any string that contains a sequence of at least X *n*'s |
| [n$](http://www.w3schools.com/jsref/jsref_regexp_ndollar.asp) | Matches any string with n at the end of it |
| [^n](http://www.w3schools.com/jsref/jsref_regexp_ncaret.asp) | Matches any string with n at the beginning of it |
| [?=n](http://www.w3schools.com/jsref/jsref_regexp_nfollow.asp) | Matches any string that is followed by a specific string n |
| [?!n](http://www.w3schools.com/jsref/jsref_regexp_nfollow_not.asp) | Matches any string that is not followed by a specific string n |

## RegExp Object Properties :

|  |  |
| --- | --- |
| Property | Description |
| [constructor](http://www.w3schools.com/jsref/jsref_regexp_constructor.asp) | Returns the function that created the RegExp object's prototype |
| [global](http://www.w3schools.com/jsref/jsref_regexp_global.asp) | Checks whether the "g" modifier is set |
| [ignoreCase](http://www.w3schools.com/jsref/jsref_regexp_ignorecase.asp) | Checks whether the "i" modifier is set |
| [lastIndex](http://www.w3schools.com/jsref/jsref_regexp_lastindex.asp) | Specifies the index at which to start the next match |
| [multiline](http://www.w3schools.com/jsref/jsref_regexp_multiline.asp) | Checks whether the "m" modifier is set |
| [source](http://www.w3schools.com/jsref/jsref_regexp_source.asp) | Returns the text of the RegExp pattern |

## RegExp Object Methods :

|  |  |
| --- | --- |
| Method | Description |
| [compile()](http://www.w3schools.com/jsref/jsref_regexp_compile.asp) | Deprecated in version 1.5. Compiles a regular expression |
| [exec()](http://www.w3schools.com/jsref/jsref_regexp_exec.asp) | Tests for a match in a string. Returns the first match |
| [test()](http://www.w3schools.com/jsref/jsref_regexp_test.asp) | Tests for a match in a string. Returns true or false |
| [toString()](http://www.w3schools.com/jsref/jsref_regexp_tostring.asp) | Returns the string value of the regular expression |

# AJAX  :

AJAX is about updating parts of a web page, without reloading the whole page.

What You Should Already Know :

Before you continue you should have a basic understanding of the following:

* HTML
* JavaScript

What is AJAX?

AJAX = Asynchronous JavaScript and XML.

AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small

amounts of data with the server behind the scenes. This means that it is

possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the

content should change.

Examples of applications using AJAX: Google Maps, Gmail, YouTube, and

Facebook.

How AJAX Works :



## JAVASCRIPT PROTOTYPES :

## All JavaScript objects inherit the properties and methods from their prototype.

## Objects created using an object literal, or with new Object(), inherit from a prototype called

## Object.prototype.

## Objects created with new Date() inherit the Date.prototype.

## The Object.prototype is on the top of the prototype chain.

## All JavaScript objects (Date, Array, RegExp, Function, ....) inherit from the Object.prototype.

## Creating a Prototype

The standard way to create an object prototype is to use an object constructor function:

### Example

function person(first,last,age,eyecolor){

 this.firstName=first;

  this.lastName=last;

 this.age=age;

 this.eyeColor=eyecolor;

}

With a constructor function, you can use the **new** keyword to create new objects from the same

prototype:

### Example

var myFather= new person("John", "Doe", 50, "blue");

var myMother = new person("Sally", "Rally", 48, "green");

## Adding Properties and Methods to Objects :

## Sometimes you want to add new properties (or methods) to an existing object.

## Sometimes you want to add new properties (or methods) to all existing objects of a given type.

## Sometimes you want to add new properties (or methods) to an object prototype.

Adding a Property to an Object :

Adding a new property to an existing object is easy:

### Example

myFather.nationality = "English";

The property will be added to myFather. Not to myMother. Not to any other person objects.

Adding a Method to an Object :

Adding a new method to an existing object is also easy:

### Example

myFather.name= function () {

return this.firstName+ "" +this.lastName;

};

The method will be added to myFather , Not to myMother.

**Adding Properties to a Prototype** :

You cannot add a new property to a prototype the same way as you add a new property to an

existing object, because the prototype is not an existing object.

### Example

person.nationality = "English";

To add a new property to a constructor, you must add it to the constructor

function:

### Example

function person(first,last,age,eyecolor) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eyecolor;

this.nationality = "English"

}

Adding Methods to a Prototype :

Your constructor function can also define methods:

### Example

function person(first, last, age, eyecolor) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eyecolor;

this.name = function() {return this.firstName + " " + this.lastName;};

Using the **prototype** Property :

The JavaScript prototype property allows you to add new properties to an existing prototype:

### Example

function person(first, last, age, eyecolor) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eyecolor;

}

person.prototype.nationality = "English";

The JavaScript prototype property also allows you to add new methods to an existing prototype:

### Example

function person(first, last, age, eyecolor) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eyecolor;

}

person.prototype.name = function() {

return this.firstName + " " + this.lastName;

};