Chapter 3
Overview of JavaScript

3.1 Overview of Scripting Language

- There is a definite need to allow users, browsing through a web site, to actually interact with the web site. The web site should accept the users input and should structure the web page content accordingly.
- Users, who browse through a web site today, prefer to choose to view what interest them. Thus, based on a user's wish the content of a web page needs to be dynamic.
- This requires a web site development environment that will allow the creation of interactive web pages. These web pages will accept input from a user, based on the input received return customized web pages, both in the content and presentation, to the user.
- In the absence of any user input the web site must be intelligent enough to return a default web page containing predetermined information and text formatting.
- This calls for a web site development environment with coding techniques capable of accepting a client's requests and processing these requests. Result of the processing being passed back to the client via standard HTML pages.
- Capturing user request is traditionally done via a Form. Form input is validated and then sends to the web server for further processing.
- A script is defined as set of instructions. For Web pages they are instructions either to the Web server (server-side scripting) or to the Web Browser (client-side scripting).
- Scripting languages, are written within HTML, to add some functionality to a Web page.
- There are two types of scripting languages:
  - Client side scripting languages affecting the data that the end user sees in a browser window.
  - Server-side scripting languages that manipulate the data, usually in a database, on the server.
- JavaScript, ASP, JSP, PHP, Perl and Python are examples of scripting languages.

3.1.1 Client side Scripting

- Client-side scripting is set of computer programs on the web that are executed by the user's web browser.
- Web developers write client-side scripts in languages such as JavaScript (Client-side JavaScript) and VBScript.
- Client-side scripts are written within an HTML document. The script may also be written in a separate file, which can be referenced by the one or more document(s).
- Client-side scripts contain instructions for the browser. When the user interacts with the document in a certain way, e.g., when a button is clicked or mouse is moved etc., these instructions can be followed without any more communication with the server.

3.1.2 Server side Scripting

- Server-side scripts are written in languages such as Perl, PHP, ASP.net, JSP etc. These languages are executed by the web server when the user requests a document.
- Server-side scripting is a web server technology where a user's request is satisfied by running a script on the web server to generate dynamic HTML pages.
- Server-side scripting is generally used for interactive web sites that interface to databases or other data stores.

3.1.3 Client side Scripting Vs Server side Scripting

- The difference between a client side script and a server side script is on which computer the script is executed. Server side scripts are executed on the server and the results are sent to the client. Client side scripts are executed on the client computer by the Web browser.
- Server executes server-side scripts and sends the page to the browser but it does not execute client-side scripts. The browser receives the page sent by the server and executes the client-side scripts.
- Client-side scripts have more access to the information and functions residing on the user's browser, whereas Server-side scripts have more access to the information and functions which are existing on the server.
- Server side scripts can be used to connect to the databases on the web server. Client side scripting cannot be used for the database connection.
- Server side scripting can access the file system located at the web server. Client side scripting can't access the file system locating at the web server.
- The settings of the Web server can be accessed with the help of Server side scripting. The files and settings at the local user's computer side can be accessed with Client side scripting. Due to security restrictions, client-side scripts may not be allowed to access the user's computer beyond the browser application.
- The user cannot block server side scripting. While the client side scripting can be blocked if the user need on browser.
- Since the server side scripting is processed on the remote computer the response server-side script is slower as compared to a client-side script it is processed. Whereas the response from a client-side script is faster as compared to a server-side script because these scripts are processed on the local computer.
- Some examples of Server side scripting languages are PHP, JSP, ASP.Net, Perl, Ruby, and many more. The examples of Client side scripting languages are Javascript, VB script, etc.
- Server-side scripts need their language's interpreter installed on the server, and generate the same output regardless of any browser, operating system, or other system details. Client-side scripts do not require any additional software other than web browser.

3.2 Introduction to JavaScript

- JavaScript is client-side scripting language of Web created by Netscape.
- JavaScript is used in billions of Web pages to add functionality, validate forms, communicate with the server, and much more.
- JavaScript is programming code that can be inserted into HTML pages.
- JavaScript inserted into HTML pages, can be executed by all modern web browsers.
- Scripts in HTML must be inserted between <script> and </script> tags.
- Scripts can be put in the <body> and in the <head> section of an HTML page.

Advantages of JavaScript

- JavaScript offers several advantages to a Web developer such as a short development cycle, ease of learning, small size scripts and so on.
- **An Interpreted Language:** JavaScript is an interpreted language. It does not require any compilation steps thus, providing an easy development process. The script is completely interpreted just as HTML tags by browser.
- **Embedded within HTML:** JavaScript does not require any separate editor for writing its program, editing or compiling. Any text editor like Notepad, Notepad++, Sublime, VS Code etc. can be used. The JavaScript code when embedded with appropriate HTML tags should be saved with .html filename extension. This file can then be read and interpreted by any browser that is JavaScript enabled.
- **Minimal Syntax - Easy to Learn:** The syntax follows very simple rule so by learning just a few commands a complete applications can be built using JavaScript.
- **Quick Development: J**avaScript does not require any long compilations time. The scripts can be developed in a short period of time. The different GUI elements, such as alerts, prompts, confirm boxes, also improve the quick development.
- **Designed for simple, small programs:** It is suitable to implement small and simple programs. These programs can be easily written and executed speedily. Also, they can be easily integrated in a web page.
- **Procedural Capabilities:** Like all other programming language JavaScript also support facilities such as condition checking, looping and branching.
- **Designed for Programming User Events:** JavaScript supports Object/Event based programming. It identify events like button click, form load etc. JavaScript code can be attached to this event, which will execute when such event occurs. They are termed as event handlers.
- **Easy Debugging and Testing:** As JavaScript is an interpreted language, the script is tested line by line, and the errors are listed with an appropriate error message. Error message includes the line number also for every error that is encountered. It is thus easy to find errors, make changes, and test it again.
- **Platform Independence / Architecture Neutral:** It is completely independent of the hardware on which it runs. Thus, it can work on any machine that has JavaScript enabled browser installed in it. This machine can reside anywhere on the network.

3.2 Structure of JavaScript

- JavaScript is embedded into an HTML file.
- A browser reads HTML files and interprets HTML tags.

- So, the browser needs to be informed that certain specific section within HTML code is JavaScript.
- The browser will then use its built-in JavaScript engine to interpret this code.
- The browser is given this information using the HTML tags <SCRIPT> …</SCRIPT>.
- The <SCRIPT> tag marks the beginning of scripting code and /SCRIPT> marks its end.
- Following explains how <script> tag can be used to embed JavaScript into HTML file.

  Syntax:

```
<script language = ”Javascript” >
    // JavaScript codes HERE
</script>
```

- Language indicates the scripting language used for writing the small piece of scripting code.
- If not specified now, JavaScript is the default scripting language.
- JavaScript codes can be added in the <head> tag as well as in the <body> tag. There can be multiple <script> tag in a single web page, but there must be the closing </script> tag for each opening tag.

  Example

```
<script language = “javascript”>
    document.write (“Hello World”);
</script>
```

- The first line <script language = “JavaScript”> tells the browser that the code written between the <script> and </script> tags is the JavaScript code.

Using External JavaScript Files

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

```
<HTML>
<HEAD>
    <script src = “myScript.js” >
    </script>
    <TILTE>title of the page</TITLE>
</HEAD>
<BODY>
      Content of page
</BODY>
</HTML>
```

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

3.3 Data types in JavaScript

Data types can be classified into two major categories:

1. Primitive Data types

- A primitive data type is a data type that stores a single value, such as number and strings.

2. Composite Data types

- A composite data type, is the same thing as an object.
- It is a data type that can consist of multiple values grouped together in some way.
- JavaScript treats objects as associative arrays.

3.3.1 Primitive Data types

There are four primitive data type, JavaScript provides to us:

a) Numbers

- They represent numeric values.
- The simplest type of number is an integer. It is a number without fractional component.
- Another type of number is a floating point number. JavaScript also has the ability to handle hexadecimal (four bit) and octal (three bit) numbers.

b) Strings

- A string is sequence of valid characters enclosed in a single or double quotes.
- An empty string is created by two quote marks with nothing between them.
- In order to put some characters in a string, which may not exist on the keyboard, or some special characters that can't appear as themselves in a string, you need to use an escape sequence to represent the character.
- An escape sequence is a character or numeric value representing a character that is preceded by a backslash to indicate that it is a special character.

c) Boolean values

- There are two boolean values, true and false.
- These are normally the result of a logical comparison in your code that returns a true/false or yes/no result, such as: a = = b
- If you need to use boolean values in computations, JavaScript will automatically convert true to 1 and false to 0.
- When testing for the result of a comparison, JavaScript will treat any nonzero value as true, and a zero value as false.
- While testing, if something does not exist, then JavaScript will also evaluate false.

d) Null

- Consists of a single value, null, which identifies a null, empty or nonexistent reference.

3.3.2 Composite Data types

- All composite data types can be treated as objects, but we normally categorize them by their purpose as a data type.
- For composite data types we will look at objects, including some special predefined objects that JavaScript provides, as well as functions and arrays.

a) Objects

- An object is a collection of named values, called the properties of that object.
- Functions associated with an object are referred to as the methods of that object.
- Properties and methods of objects are referred to with a dot (.) notation that starts with the name of the object and ends with the name of the property.
- JavaScript has many predefined objects, such as a Date object and a Math object.

b) Functions

- A function is a predefined small piece of code which is executed based on a call to it by name.
- The function code may be reused many times in the same document.
- A function is a data type in JavaScript.
- Any JavaScript code which is not inside a function is executed when the Web browser reaches it while first parsing the document.

c) Array

- An Array is an ordered collection of data values.
- In JavaScript, an array is just an object that has an index to refer to its contents. The elements in the array are numbered, and you can refer to the number position to access the element.
- The array index is included in square brackets immediately after the array name.
- In JavaScript, the array index starts with zero, so the first element in an array would be arrayName[0], and the fourth element would be arrayName[3].
- JavaScript does not have multidimensional arrays, but you can nest them, which means that you can have, an array as an element in another array.
- You access them listing the array numbers starting for the outermost array and working inward.
- Therefore, the third element (position 2) of or inside the ninth element (position 8) would be arrayName[8][2].

3.4 JavaScript Variable & Constant

3.4.1 Variable

- Variable is a name that can be used to store values.
- These variables can take different values but one at a time and the value can be changed during the execution of program.
- Variable names may consist of uppercase character, lowercase character and underscore.

Rules to giving the name of variable are as:

- First character should be a either a letter or an underscore.
- The keywords cannot be a variable name..
- Uppercase and lowercase letters are considered different for example code, Code, CODE are three different variables.

Typing

- JavaScript is an un-typed language.

- This means you can use variables directly where you want to use it.
- The variables are declared with the var keyword.

```
var variablename=value;
```

- This keyword declares all types of variables and you can use a same variable as string, as Integer, as Number and any type of Objects.

Example:

```
var a=”Mr. ABC”;
a=12345;
```

Both statements are valid in JavaScript as JavaScript is un-typed language.

Scope of a Variable

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

Local Variables

- The variables which are defined within the body of the function are local to that function and it is called local variable.
- They cannot be referred outside the function.
- Their values cannot be changed by the main code or other functions.

Example:

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

Output:

```
Local value of a is: 5
```

Global Variables

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

Example:

```
<script language=” javascript”>
    var a=10;
    function testGlobal()
    {
        alert(“The global value of a is: ” + a);
```

```
        //displays the value of a as 10
    }
    testGlobal();
</script>
```

Here the value of a is global and is accessed within the function testGlobal.

3.4.2 Constant.

- A constant is a value which cannot be changed while the script is running.
- You can create a read only, named constant with the const keyword.
- Its value cannot be even changed through assignment or even cannot be redeclared.
- A constant identifier, just like variable identifier, must start with a letter or underscore and can contain alphabetic, numeric, or underscore characters.

Example:

```
const prefix = '212';
```

- If the keyword 'const' is omitted, the identifier is assumed to represent a var.

3.5 Operators in JavaScript

JavaScript has different types of operator, which can be classified by some criteria, such as:

3.5.1 Number of Operands

In this type of category, JavaScript supports 3 basic types of operators:

a) Unary: A unary operator requires a single operand, either before or after the operator.

Example:

```
i--;
++i;
```

b) Binary: A binary operator requires two operands, one before the operator and one after the operator.

Example:

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

c) Ternary: This operator is also known as conditional operator. It is the only JavaScript operator that takes three operands. The operator can have one of two values based on a condition.

The syntax is:

```
condition ? value1 : value2 ;
```

If condition is true, the operator has the value of value1 otherwise it has the value of value2.

Example:

```
var c = (a < b) ? a : b;
```

This statement assigns the value of a to the variable c if a is less than value of variable b. Otherwise, it assigns the value of b.

3.5.2 Number of Operation.

- JavaScript supports many types of operators. They can be categorized as:

a) Conditional Operator: JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename=(condition)?value1:value2

Example

var greeting = (visitor=="GUEST")?"Welcome Guest":"Hello ";

- If the variable visitor has the value of "GUEST", then the variable greeting will be assigned the value "Welcome Guest " , otherwise it will be assigned "Hello".

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

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

c) Comparison Operators

- A comparison operator compares its operands and returns a logical value either true or false based on the comparison.
- The operands can be numerical, string, logical, or object values.
- Strings are compared based on standard lexicographical ordering, using unicode values.
- The following table describes the comparison operators.

Operator	Description	Example
Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert	Var1=3 then expression (Var1 == “3”)

	the operands to an appropriate type for the comparison.	returns true
Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	expression (Var1 !="3") returns false
Strict equal (===)	Returns true if the operands are equal and of the same type.	expression (Var1 ==="3") returns false
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	expression (Var1 !=="3") returns true
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2

d) Assignment Operators

- An assignment operator assigns a value to its left operand based on the value of its right operand
- The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand.

  i.e. x = 5 assigns the value 5 to x.
- The other assignment operators are shorthand for standard operations, as shown in the following table.

Shorthand Operator	**Meaning**
x + = y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

e) Logical Operators

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

Operator	Description
&& Logical AND	Expression (expr1 && expr2) () Returns true if both operands are true; otherwise, returns false.
|| Logical OR	Expression (expr1 || expr2) Returns true if either operand is true; if both are false, returns false.
! Logical NOT	Expression (!expr1) Returns false if expr1 is true; otherwise, returns true.

f) delete

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

Example:

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

g) new

- You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.

Syntax:

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

3.6 JavaScript Conditional Statements

- In JavaScript we have the following conditional statements:
  - if statement - use this statement to execute a few code only if a specified condition is true.
  - if...else statement - use this statement to execute a few code if the condition is true and another code if the condition is false.
  - if...else if...else statement - use this statement to select one of many blocks of code to be executed depending on the condition matched.

- switch statement - use this statement to select one of many blocks of code to be executed depending on the case matched.

a) if statement

- Use if statement to execute some code only if a specified condition is true.

Syntax

```
if ( condition)
{
    code to be executed if condition is true
}
```

Example

```
<script language="javascript">
    //Write a "Happy Sunday" greeting if the day is 0 of the week
    var d=new Date();
    var weekday=d.getDay();
    if (weekday==0)
    {
        document.write("<b>Have a happy Sunday</b>");
    }
</script>
```

- There is no else in this syntax. You tell the browser to execute some code only if the specified condition is true.

b) if ….. else statement

- Use if...else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if ( condition)
{
    // code to be executed if condition is true
}
else
{
    // code to be executed if condition is not true
}
```

Example:

```
<script language="javascript">
    // Write a "Happy Sunday" greeting if the day is 0 of the week
    // Otherwise it is not happy sunday
    var d = new Date();
    var weekday = d.getDay();
    if (weekday == 0)
    {
        document.write("Happy Sunday");
    }
    else
    {
        document.write("Not a Sunday”)’;
    }
</script>
```

c) if ….. else if…..else statement

- Use if...else if...else statement to select one of several blocks of code to be executed.

  Syntax

```
if ( condition1)
{
   code to be executed if condition1 is true
}
else if ( condition2)
{
   code to be executed if condition2 is true
}
else
{
   code to be executed if neither condition1 nor condition2 is true
}
```

  Example

```
<script language="javascript">
   var d = new Date()
   var weekday = d.getDay()
   if (weekday==0)
   {
      document.write("<b>Happy Sunday</b>");
   }
   else if (weekday==6)
   {
      document.write("<b>Happy Weekend It's a Saturday</b>");
   }
   else
   {
      document.write("<b>Hectic WeekDay</b>");
   }
</script>
```

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

d) switch statement

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

  Syntax

```
switch(n)
{
   case 1:
      execute code block 1
      break;
   case 2:
      execute code block 2
      break;
   default:
      code to be executed if n is different from case 1 and 2
}
```

- This is how it works:
  - First we have a single expression n (most often a variable), that is evaluated once.
  - The value of the expression is then compared with the values for each case in the structure.
  - If there is a match, the block of code associated with that case is executed.
  - Use break to prevent the code from running into the next case automatically.

Example

```
<script language="javascript">
   //You wil receive a different greeting based on what day it is. Note that
   //Sunday=0, Monday=1, Tuesday=2, etc.
   var d=new Date();
   var weekday=d.getDay();
   switch (weekday)
      {
      case 0:
         document.write("Happy Sunday");
         break;
      case 6:
         document.write("Super Saturday");
         break;
      default:
         document.write("Hectic Week Day");
      }
</script>
```

3.7 JavaScript Looping Statements

- Often when you write code, you want the repeat some block of code to run over and over again. Instead of adding the same lines in a script repeatedly, we can use loops to perform a job like this.
- In JavaScript, there are following types of loops:
  1. for - loops through a block of code a specified number of times
  2. while - loops through a block of code while a specified condition is true
  3. do..whle - loops through a block of code once, and then repeats the loop while a condition is true

a) The for loop

- The for loop is used when you know in advance how many times the script should run.

Syntax

```
for ( initialization; terminating condition; increment/ decrement)
{
   code to be executed
}
```

Example

- The example below defines a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs.

Example:

```
<html>
   <body>
      <script language="javascript">
         var i=0;
         for (i=0;i<=5;i++)
         {
            document.write("The number is " + i);
            document.write("<br />");
         }
      </script>
   </body>
</html>
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

b) The while loop

- The while loop loops through a block of code till a specified condition is true.

Syntax

```
while (condition)
   code to be executed
}
```

Example

```
<html>
   <body>
      <script language="javascript">
         var i=0;
         while (i<=5)
         {
            document.write("The number is " + i);
            document.write("<br />");
            i++;
         }
      </script>
   </body>
</html>
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

c) The do…..while loop

- The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
{
code to be executed
}while ( condition);
```

Example

```
<html>
   <body>
      <script language="javascript">
      var i=0;
      do
      {
         document.write("The number is " + i);
         document.write("<br />");
         i++;
      } while (i<=5);
      </script>
   </body>
</html>
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

3.8 Break & Continue Statements

3.8.1 The break statement

- The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
   <body>
      <script language="javascript">
         var i=0;
         for (i=0;i<=5;i++)
            {
            if (i==3)
            {
               break;
            }
            document.write("The number is " + i);
            document.write("<br />");
         }
      </script>
```

```
    </body>
</html>
```

Output:

```
The number is 0
The number is 1
The number is 2
```

3.8.2 The continue Statement

- The continue statement will break the current loop and continue with the next value.

  Example:

```
<html>
    <body>
        <script language="javascript">
            var i=0
            for (i=0;i<=10;i++)
            {
                if (i==3)
                {
                    continue;
                }
                document.write("The number is " + i);
                document.write("<br />");
            }
        </script>
    </body>
</html>
```

  Output:

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
```

3.9 JavaScript Strings

- JavaScript string is a series of characters.
- A string is any text between quotes. It can be written using single or double-quotes.

  Following are the valid string:

```
book_name = "Concept of web technology";
book_name = ' Concept of web technology ';
```

- It is possible to use quotes inside a string. If you want a single quote inside a string, then write a string in double-quotes. If you want a double quote inside a string, then write a string in single quotes.
- Example:

```
book_name = "It's web technology ";
book_name = "it is web technology  'book'";
book_name = 'it is web technology "book "';
```

3.9.1 String functions

- Following is the list of a string function. Java script string starts at index position 0.

Method	Description
charAt()	Returns the character at the specified index position.
concat()	Combines two strings and returns a new string.
indexOf()	Returns the location of the first occurrence of the specified value. Returns -1 if the value is not found.
lastIndexOf()	Returns the location of the last occurrence of the specified value. Returns -1 if the value is not found.
replace()	It is used to replaces a part of a given string with the specified string.
search()	It searches a specified value or regular expression in a given string and returns its position. If a match is not found then it returns -1.
trim	It is used to remove white space from the left and right of the string.
split()	It is used to convert a string to an array of strings.
slice()	It is used to extract string between two given positions.
substring()	It is used to extract string between two given positions. It is similar to slice, except that substring () does not accept a negative value.
substr()	It is used to extract subpart of string. Here subpart of the string is starting at the specified position and extending for a given number of characters.
toLowerCase()	It is used to converted string to lower case.
toUpperCase()	It is used to convert string to uppercase.

- Following example demonstrate various string functions:

```
<script>
        str1 = "Web Technology";
        str2="Concept"
        document.write( str1.substr(3,5));
        document.write("<br>");
        document.write(str1.charAt(2));
        document.write("<br>");
        document.write(str1.concat(str2));
        document.write("<br>");
        document.write(str1.indexOf("Tec"));
        document.write("<br>");
```

```
        document.write(str1.replace("Technology", "Designing"));
        document.write("<br>");
        document.write (str1.search("Tec"));
        document.write("<br>");
        document.write(str1.slice(3,7));
        document.write("<br>");
        document.write(str1.substr(3,7));
        document.write("<br>");
        document.write(str1.toLowerCase())
        document.write("<br>");
        document.write(str1.toUpperCase())
        document.write("<br>");
</script>
```

Output:

```
b
Web TechnologyConcept
4
Web Designing
4
Tec
Techno
web technology
WEB TECHNOLOGY
```

3.10 JavaScript Events & Event Handlers

- Events are some activity performed by the user or by the browser. HTML events are action that happens to HTML elements which can trigger a JavaScript.
- Events are keyboard, mouse, or other actions that can be detected by JavaScript.
- For example, we can use the *click* event of a button to execute a function.
- We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page loading, finished loading
- Image loading
- Entering a value in an input field
- Submitting a form

a) Mouse events:

Following is the list of mouse events:

Action/Event	Event name/ Handler	Description
click	onclick	This event occurs when user clicks on an element.
mouseover	onmouseover	This event occurs when the cursor of the mouse bring over the element

mousemove	onmousemove	This event occurs when the mouse pointer is moved
mouseout	onmouseout	This event occurs when the cursor of the mouse leaves an element
mouseup	onmouseup	This event occurs when the mouse button is released over the element

b) Keyboard events:

Following is the list of Keyboard events:

Action/Event	Event name/ Handler	Description
Keyup	onkeyup	This event occurs when the user release the key
Keydown	onkeydown	This event occurs when the user press the key

c) Form events:

Following is the list of form events:

Action/Event	Event name/ Handler	Description
focus	onfocus	This event occurs when an element gets focus
blur	onblur	This event occurs when an element lost focus. i.e the focus is away from an element.
submit	onsubmit	This event occurs when the user submits the form
change	onchange	This event occurs when the user modifies the value of a form element

- Focus and blur are opposite events.

Note: Events are used in combination with functions. You will learn more about functions in chapter 5.

Event Handlers

- Event handlers are nothing but functions that executes when a particular event occurs.
- Event handlers are a way to run JavaScript code in case of user actions.
- In JavaScript, all the event handlers start with the word *on*.
- The syntax of **event handlers** is:

```
name_of_handler="JavaScript code"
```

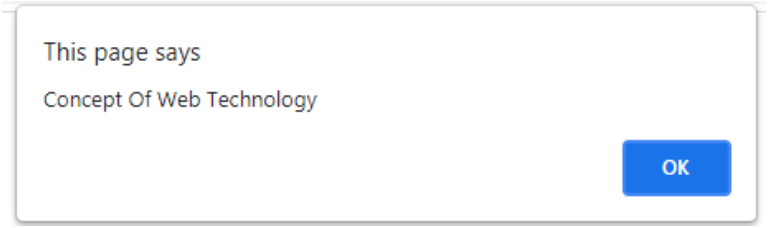
Example of click event:

Following code display message in alert box when user click button.

```
<script>
    function function_name() {
        alert("Concept Of Web Technology")
    }
</script>
<body>
    <button onclick="function_name()">demo of click event   </button>
</script>
```

Output:

When a user clicks on “demo of click event”, the following alert box will display.



Example of *onmouseover* and *onmouseout*:

Following code demonstrate *onmouseover* and *onmouseout* event. When user move mouse over the image, image will bigger and when mouse is out, image will display normally.

```
<!DOCTYPE html>
<html>
    <script>
        function bigger() {
            document.getElementById("sample").style.height = "100px";
            document.getElementById("sample").style.width = "100px";
         }
        function normal() {
           document.getElementById("sample").style.height = "50px";
           document.getElementById("sample").style.width = "50px";
        }
    </script>
<body>
    <img id="sample" onmouseover="bigger()" onmouseout="normal()"
    src="jagin.jpg"  width="40" height="40">
</body>
</html>
```

Example of *onfocus* and *onblur*:

- Following example demonstrate *onfocus* and *onblur* events. When user click on text box, text box get focus and *onfocus* event occur. Here it will display text “focus is with text box” in text box. When user click outside textbox, textbox lost focus and *onblur* event occur. It will display text “focus is away from text box” in text box.

```
<script>
    function get_focus() {
        var x = document.getElementById("sample");
        x.value = "focus is with text box";
    }
    function lost_focus() {
        var x = document.getElementById("sample");
        x.value = "focus is away from text box";
    }
```

```
</script>
<body>
   Demo of focus and blur event: <input type="text" id="sample"
   onfocus="get_focus()" onblur="lost_focus()" >
</body>
```