Chapter - 3: Web Scripting - Java Script

Introduction

JavaScript is an interpreted computer programming language. It was originally implemented as part of web browsers so that client-side scripts could interact with the user, control the browser, communicate asynchronously, and alter the document contents. Nowadays, JavaScript has become very useful in both game development and the creation of desktop applications.

JavaScript was developed in 1995 by Brendan Eich, at Netscape, and first released with Netscape 2 early in 1996. It was initially called as LiveScript, but was renamed as JavaScript in order to capitalize the popularity of Sun Microsystem's Java language. JavaScript's use in applications outside of web pages also like in PDF documents, site-specific browsers, and desktop widgets and other useful applications.

JavaScript was formalized in the ECMAScript language standard and is primarily used as part of a web browser (client-side JavaScript). This enables programmatic access to computational objects within a host environment. JavaScript very quickly gained widespread success as a client-side scripting language for web pages. Microsoft introduced JavaScript support in its own web browser, Internet Explorer, in version 3.0, released in August 1996.

3.1 Java Script Review

JavaScript is a cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language. It is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers. Inside a host environment, JavaScript can be connected to the objects of its environment to provide programmatic control over them.

Core JavaScript contains a core set of objects such as Array, Date, Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects.

- Client-side JavaScript extends the core language by supplying objects to control a browser (Navigator or another web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
- Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

Standardization

In November 1996, Netscape announced that it had submitted JavaScript to European Computer Manufacturers Association (ECMA) for consideration as an industry standard, and subsequent work resulted in the standardized version named ECMAScript. ECMA is an international standard organization for information and communication system. In June 1997, ECMA International published the first edition of the ECMA-262 specification. A year later, in June 1998, some modifications were made to adapt it to the ISO/IEC-16262 standard, and the second edition was released. The third edition of ECMA-262 is the version most browsers currently use.

Fourth edition of ECMAScript standard was not released and does not exist. Fifth edition of the ECMAScript standard was released in December 2009. The current edition of ECMAScript standard is 5.1 and it was released in June 2011. This way, JavaScript has become one of the most popular programming languages on the web.

Applications

Developing Multimedia Applications

The users can use JavaScript to add multimedia elements. With JavaScript you can show, hide, change, resize images and create images rollovers. You can create scrolling text across the status bar, thus making multimedia applications more interactive.

Create Pages Dynamically

Based on the user's choice, the date or other external data, JavaScript can produce pages that are customized to the user.

Interact with the User

JavaScript can do some processing of forms and can validate user input when the user submits the form.

JavaScript Objects are Similar to Dictionaries

In JavaScript, objects are just collections of name-value pairs. JavaScript objects are considered as a dictionary with string keys. The users can get and set the properties of an object using either the familiar "." (dot) operator, or the "[ ]" operator, which is typically used when dealing with a dictionary.

Features of JavaScript

Browser Support

All browsers have accepted JavaScript as a scripting language and provide integrated support for it. For example, to access flash content, you need to install flash plug-in in your browser. But to use JavaScript, you don't have to use any plug-in at all.

JavaScript can be used on Client Side as well as on Server Side

JavaScript has access to Document Object Model DOM of browser. You can change the structure of web pages at runtime. Thus, JavaScript can be used to add different effects to WebPages. On the other hand, JavaScript could be used on the server side as well.

Functional Programming Language

In JavaScript, function could be assigned to variables just like any other data types. A function can accept another function as a parameter and can also return a function. You can have functions with no name as well. This provides you the ability to code in functional programming style.

Support for Objects

JavaScript is an object oriented language. However, the way JavaScript handles objects and inheritance is bit different from conventional object oriented programming languages like C++/ Java. JavaScript supports most of the object oriented concepts while being simple to learn and use.

Run-time Environment

JavaScript typically relies on a run-time environment (e.g. in a web browser) to provide objects and methods by which scripts can interact with "the outside world". In fact, it relies on the environment to provide the ability to include/import scripts (e.g. HTML <script> elements). This is not a language feature as such but it is common in most JavaScript implementations.

Vendor-specific Extensions

JavaScript is officially managed by Mozilla Foundation, and new language features are added periodically. However, only some JavaScript engines support these new features.

Object based Features Supported by JavaScript

JavaScript supports various features related to object based language and JavaScript is sometimes referred to as an object-based programming language. Some robust features which JavaScript supports related to object based are as follows:

Object Type

JavaScript supports the development of object types and in this context JavaScript supports both predefined and user-defined objects. It is possible to assign objects of any type to any variable. It is possible to instantiate the defined object types to create object instances in JavaScript, which is a very powerful feature of Object based language.

Object Instantiation

In order to carry out the process of creating specific object instances available in JavaScript, you can make use of a new operator. These two powerful, object-based features supported by JavaScript described above make this an object model language. In

JavaScript, the object types are defined by properties and methods. Properties of Objects are used to access the data values contained in an object. You can make use of the properties of JavaScript objects for editing as well as reading depending on the object's nature. That is, if you want to carry out functions on the object, this is achieved by using methods that make use of the object's properties.

These are some of the features that give JavaScript an ability to handle simple as well as complex tasks. This way, JavaScript has remained as the most popular programming language for a long time. It is also a good language for web designers who want to learn computer programming as it supports object oriented as well as function concepts and to use it, you just need a browser and a text editor.

Writing Java Script

JavaScript code is typically embedded in the HTML, to be interpreted and run by the client's browser. Here are some tips to remember when writing JavaScript commands.

- JavaScript code is case sensitive
- White space between words and tabs are ignored
- Line breaks are ignored except within a statement
- JavaScript statements end with a semi- colon;

The Script Tag

The <SCRIPT> tag alerts a browser that JavaScript code follows. It is typically embedded in the HTML.

<SCRIPT language = "JavaScript">

statements

</SCRIPT>

Examples

Variables in JavaScript can be defined using the var keyword:

```
var x;
//defines the variable x, although no value is assigned to it by default
var y = 2;
 //defines the variable y and assigns the value of 2 to it
```

Note the comments in the example above, both of which were preceded with two forward slashes.

There is no built-in I/O functionality in JavaScript as such. It is the runtime environment which provides I/O functionality. The ECMAScript specification in edition 5.1 mentions.

Actually, there are no provisions in this specification for input of external data or output of computed results. However, most runtime environments have a console object that can be used to print output. Here we elaborate with the help of a Hello World program:

```
console.log("Hello world!");
```

Relationship Between JavaScript Versions and ECMAScript Editions

The following table describes the relationship between JavaScript versions and ECMAScript editions.

Table : JavaScript versions and ECMAScript editions

JavaScript version	Relationship to ECMA Script edition
JavaScript 1.1	ECMA-262, Edition 1 is based on JavaScript 1.1.
JavaScript 1.2	ECMA-262 was not complete when JavaScript 1.2 was released. JavaScript 1.2 is not fully compatible with ECMA-262, Edition 1, for the following reasons: • Netscape developed additional features in JavaScript 1.2 that were not considered for ECMA-262. • ECMA-262 adds two new features: internationalization using Unicode, and uniform behavior across all platforms. Several features of JavaScript 1.2, such as the Date object, were platform-dependent and used platform-specific behavior.
JavaScript 1.3	JavaScript 1.3 is fully compatible with ECMA-262, Edition 1. JavaScript 1.3 resolved the inconsistencies that JavaScript 1.2 had with ECMA-262, while keeping all the additional features of JavaScript 1.2 except == and !=, which were changed to conform with ECMA-262.
JavaScript 1.4	JavaScript 1.4 is fully compatible with ECMA-262, Edition 1. The third version of the ECMAScript specification was not finalized when JavaScript 1.4 was released.
JavaScript 1.5	JavaScript 1.5 is fully compatible with ECMA-262, Edition 3.

Why JavaScript?

JavaScript is a simple scripting language invented specifically for use in web browsers to make websites more dynamic. On its own, HTML is capable of outputting more-or-less static pages. Once you load them up your view doesn't change much until you click a link to go to a new page. Adding JavaScript to your code allows you to change how the

document looks completely, from changing text, to changing colours, to changing the options available in a drop-down list and much more.

JavaScript is a client-side language, which means all the action occurs on the client's (reader's) side of things. This means that no trips to the server are required for JavaScript to kick into operation, which would slow down the process enormously. JavaScript operations are usually performed instantaneously. In fact, JavaScript is often used to perform operations that would otherwise encumber the server, like form input validation. This distribution of work to the relatively quick client-side service speeds up the process significantly.

JavaScript is integrated into the browsing environment, which means they can get information about the browser and HTML page, and modify this information, thus changing how things are presented on your screen. This access to information gives JavaScript great power to modify the browsing experience. They can also react to events, such as when the user clicks their mouse, or points to a certain page element. This is also a very powerful ability.

Browser Compatibility

JavaScript is supported by Netscape 2+, Internet Explorer 3+, Opera 3+ and most of the other modern web browsers. Each new version of the main browsers has supported new generations of JavaScript commands, each more complex than the earlier one. However, script compatibility can still be a problem, as the language is not as standardised as HTML.

Writing First JavaScript

How are the users going to get JavaScript into web pages? JavaScript is written in the same way as HTML in a text-editor. JavaScript implementation is quite similar to CSS (Cascading Style Sheets). The users can link to outside files (with the file extension .js), or write blocks of code right into your HTML documents with the <script> tag.

Let us consider a simple example with an **embedded script**. This will simply print a line of text to the web page.

```
<script type="text/javascript">
<!--
document.write("<i>Hello World!</i>");
//comment line ……this is comment
</script>
```

When you place that in your code the text Hello World will appear on your screen wherever you put it. Like so:

Hello World!

The script tag encloses any script code you want to use. The type attribute we have to alert the browser to the type of script it is about to deal with, and so helps it to interpret the code.

The comments around the script code are there so that old browsers that don't understand the script tag won't display the code as text on the page. Any browser that can do JavaScript will disregard the comments. Also note that for Netscape's benefit, the end of the comment is itself commented out using a **JavaScript comment** (two forward-slashes to comment out the rest of the line). This stops errors from occurring in old versions of Netscape.

External Scripts

To import scripts from external JavaScript files, save the code in a text file with the .js extension without the script tags and comments. In this case the code would just be the document.write("Hello World!"). The users can link to this document in the page's <head> with the following code.

```
<script type="text/ javascript"
src="simplemethods.js"></script>
```

Now all the methods and variables that are in that file are available to use in the page. The users should always place includes in the head so that the browser is ready to execute scripts when the user calls for them. If a user clicked a button that called for a script that browser wasn't aware of yet, you may get an error. Having them in the head body means they're always ready before they're needed.

A Simple Script

```
<script type="text/javascript">
<!--
document.write("<i>Hello World!</i>");
//-->
</script>
```

Document object uses its write( ) method to output some text to the page-document. The text inside the double quotes is called a string, and this string will be added to the page. To use an object's methods or properties we write the object's name, a dot [.] and then the method/property name. Each line of script ends with a semicolon. JavaScript isn't very forgiving that is if you make any mistakes in typing, the users will get a script error.

Example for creating some HTML and text.

```
<script type="text/javascript">
<!--
document.write("<h1>Main Title</h1>");
document.write("<p align='right'>Body</p>");
```

```
//-->
</script>
```

Note that when quoting attribute values, you have to **use single quotes**, as if you used double quotes the write method would think the string was over prematurely, and you'd get an error.

How to Save and Run your Program in JavaScript

1. Open any editor like notepad and write the program
2. Save the program in a file with .html extension in a proper folder or subfolder on a drive.
3. Open the web browser like internet explorer or Mozilla Firefox
4. Open the file you created and save in step 2, to execute the program like to execute C:/JAVA/firstprogram.html

Where C is the drive, JAVA is the folder name and firstprogram.html is the name of program which is saved in the Java folder

Programming using JavaScript

JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment. It is up to the host environment to provide mechanisms for communicating with the outside world. The most commonly used host environment is the browser. Although, JavaScript interpreters can also be found in other formats such as Adobe Acrobat, Photoshop, Yahoo!'s Widget engine, and server side environments. JavaScript programs manipulate values, and these values all belong to a type. Different JavaScript's types include Number, String, Boolean and Objects.

Objects provide a great deal of utility in writing more maintainable code. If a function relies on one or two other functions that are not useful to any other part of your code, you can nest those utility functions inside the function that will be called from elsewhere. This keeps the number of functions that are in the global scope down.

Note: When writing complex code it is often tempting to use global variables to share values between multiple functions, which leads to code that is hard to maintain. Nested functions can share variables in their parent, so you can use this mechanism to couple functions together when it makes sense without disturbing your global namespace 'local/global' if you desire so. This approach should be applied with caution, but it's a useful ability to have for potential users.

Operators

JavaScript operators can be used to perform various operations such as:

- Arithmetic Operators
- Comparison Operators
- Logical Operators

- Relational Operators
- Assignment Operators
- Conditional Operators

The Arithmetic Operators:

There are following arithmetic operators supported by JavaScript language: Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

The Comparison Operators: There are following comparison operators supported by JavaScript language. Assume variable A holds 10 and variable B holds 20 then

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Logical Operators: There are following logical operators supported by JavaScript language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non-zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	! (A && B) is false.

The Bitwise Operators: There are following bitwise operators supported by JavaScript language. Assume variable A holds 2 and variable B holds 3 then:

Operator	Description	Example
&	Called Bitwise AND operator. It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2.
|	Called Bitwise OR Operator. It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3.
^	Called Bitwise XOR Operator. 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.	(A ^ B) is 1.
~	Called Bitwise NOT Operator. It is a unary operator and operates by reversing all bits in the operand.	(~B) is -4.
<<	Called Bitwise Shift Left Operator. It moves all 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 by 2, shifting two positions is equivalent to multiplying by 4, etc.	(A << 1) is 4.
>>	Called Bitwise Shift Right with Sign Operator. It moves all bits in its first operand to the right by the number of places specified in the second operand. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. If the first operand is positive, the result has zeros placed in the high bits; if the first operand is negative, the result has ones placed in the high bits. Shifting a	(A >> 1) is 1.


	value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is equivalent to integer division by 4, and so on.	
>>>	Called Bitwise Shift Right with Zero Operator. This operator is just like the >> operator, except that the bits shifted in on the left are always zero,	(A >>> 1) is 1.

The Assignment Operators: There are following assignment operators supported by JavaScript language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Miscellaneous Operator

- **The Conditional Operator (? :)**

  There is an operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation.

Operator	Description	Example
? :	Conditional Expression	If Condition is true? Then value X: Otherwise value Y

- **The Type of Operator**

The type of is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The type of operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is the list of return values for the type of Operator:

Type	String Returned by Type of
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Data Types

A **data type** is a classification of the type of data that a variable or object can hold. Data type is an important factor in virtually all computer programming languages, including visual basic, C#, C/C++ and JavaScript. When programmers create computer applications, both desktop and web-based, data types must be referenced and used correctly to ensure the result of the application's functions is correct and error-free. There are various data types discussed in details as given below:

Numbers

Numbers in JavaScript are double-precision 64-bit format. The standard numeric operators are supported, including addition, subtraction, modulus (or remainder) arithmetic and so forth. There's also a built-in object called Math to handle more advanced mathematical functions and constants.

Syntax: **var num=new Number(value);**

We can convert a string to an integer using the built-in parseInt() function. This takes the base for the conversion as an optional second argument, which you should always provide:

parseInt("123", 10)

123


parseInt("010", 10)

10

If you don't provide the base, you can get surprising results in older browsers:

parseInt("010")

8

It happens because the parseInt() function decided to treat the string as octal due to the leading 0.

If you want to convert a binary number to an integer, just change the base:

parseInt("11", 2)

3

Similarly, we can parse floating point numbers using the built-in parseFloat() function which uses base 10 .

You can also use the unary + operator to convert values to numbers:

+ "42"

42

A special value called NaN (short for "Not a Number") is returned if the string is non-numeric:

parseInt("hello", 10)

NaN

NaN is toxic: If we provide it as an input to any mathematical operation the result will also be NaN,

NaN + 5

NaN

We can test for NaN using the built-in isNaN() function:

isNaN(NaN)

true

JavaScript also has the special values +Infinity and -Infinity:

+1 / 0

+Infinity

-1 / 0

-Infinity

We always can test for +Infinity, -Infinity and NaN values using the built-in isFinite() function:

isFinite(1/0)

false

isFinite(-Infinity)

false

isFinite(NaN)

false

Note: The parseInt() and parseFloat() functions parse a string until they reach a character that isn't valid for the specified number format, then return the number parsed up to that point. However the "+" operator simply converts the string to NaN if there is any invalid character in it.

Strings

Strings in JavaScript are sequences of characters. More precisely, they're sequences of Unicode characters, with each character represented by a 16-bit number. If we want to represent a single character, we just need to use a string of length 1.

To find the length of a string, access its length property:

"hello".length

5 //length of given string “hello”

The strings are represented using objects and they have methods as well:

"hello, world".replace("hello", "goodbye") //by using replace() method

goodbye, world

"hello".toUpperCase() //by using toUpperCase() method

HELLO

Other types

JavaScript distinguishes between null, which is an object of type 'object' that indicates a deliberate non-value, and undefined, which is an object of type 'undefined' that indicates an uninitialized value.

In JavaScript it is possible to declare a variable without assigning a value to it. If we do this, the variable's type is undefined.

JavaScript has a boolean type, with possible values true and false (both of which are keywords). Any given value can be converted to a boolean according to the following rules:

1) false, 0, the empty string (" "), NaN, null, and undefined all become false

2) all other values become true

You can perform this conversion explicitly using the boolean() function:

```
  boolean("")
```

false

```
  boolean(234)
```

true

Variables

New variables in JavaScript are declared using the var keyword:

```
var a;
var name = "Student_Name";
```

If we declare a variable without assigning any value to it, by default its type is undefined.

Note: An important difference from other languages like Java is that in JavaScript, blocks do not have scope; only functions have scope. So if a variable is defined using var in a compound statement (like inside an if control structure), it will be visible to the entire function.

Control Structures

JavaScript has a similar set of control structures similar to other languages like C\C++ and Java. Conditional statements are supported by if and else,

```
var name = "kittens";
if (name == "puppies") {
  name += "!";
} else if (name == "kittens") {
  name += "!!";
} else {
  name = "!" + name;
}
name == "kittens!!"
```

JavaScript has while loops and do-while loops. The first is good for basic looping; the second for loops to ensure that the body of the loop is executed at least once:

```
while (true)
 {
  // an infinite loop!
  }
var input;
```

```
do
 {
  input = get_input();
} while (inputIsNotValid(input))
```

JavaScript's for loop is the same as that in C\C++ and Java which provides the control information for the loop on a single line.

```
for (var i = 0; i < 5; i++)
{
  // this loop will execute 5 times
}
```

The && and || operators are logical operators i.e., they will execute their second operand dependent on the first. This is useful for checking null objects before accessing their attributes:

var name = o && o.getName();

Or for setting default values:

var name = otherName || "default";

JavaScript has a ternary operator for conditional expressions:

```
var allowed = (age > 18) ? "yes" : "no";
```

The switch statement can be used for multiple branches based on a number or string:

```
switch(action) {
    case 'draw':
        drawit();
        break;
    case 'eat':
        eatit();
        break;
    default:
        donothing();
}
```

If we don't add a break statement, execution will "fall through" to the next level. In fact it's worth specifically labelling deliberate fallthrough with a comment :

```
switch(a)
```


```
{
   case 1: // fallthrough
   case 2:
      eatit();
      break;
   default:
      donothing();
}
```

The default clause is optional. We can have expressions in both the switch part and the cases, comparisons take place between the two using the “==” operator:

```
switch(1 + 3)
{
   case 2 + 2:
      yay();
      break;
   default:
      neverhappens();
}
```

Objects

JavaScript objects are simply collections of name-value pairs. The "name" part is a JavaScript string, while the value can be any JavaScript value including more objects.

There are two basic ways to create an empty object:

var obj = new Object( ); and **var obj = { };**

These are semantically equivalent, the second is called object literal syntax, and is more convenient. Once created, an object's properties can again be accessed in one of two ways:

```
obj.name = "Simon";
var name = obj.name;
```

And...

```
obj["name"] = "Simon";
var name = obj["name"];
```

These are also semantically equivalent. The second method has the advantage that the name of the property is provided as a string, which means it can be calculated at run-time.It can also be used to set and get properties with names that are reserved words:

obj.for = "Simon"; // Syntax error, because 'for' is a reserved word

obj["for"] = "Simon"; // works fine

Object literal syntax can be used to initialise an object :

```
var obj = {
 name: "Carrot",
 "for": "Max",
   details: {
      color: "orange",
      size: 12
   }
}
```

Attribute access can be chained together:

To find the color of the given object,

obj.details.color

Output: Orange

To get the size of the given object,

obj["details"]["size"]

Output: 12

Arrays

Arrays in JavaScript are actually a special type of object. They work similar to regular objects but they have one magic property called 'length'. The length of the array (size of the array) is always one more than the highest index in the array. The traditional way of creating arrays is as follows:

Example:

```
var a = new Array();
a[0] = "dog";
a[1] = "cat";
a[2] = "hen";
a.length
```

Output: 3

A more convenient notation can be used while dealing with arrays:

Example:-

```
var a = ["Apple", "Mangoes", "Orange", “Banana”];
 a.length
```

Output: 4

Note: Leaving a trailing comma at the end of an array literal is inconsistent across browsers, so don't do it. The length of the array is one more than the highest index. However, the array.length isn't necessarily the number of items in the array.

Example:

```
var a = ["dog", "cat", "hen"];
a[100] = "fox";
a.length
```

Output: 101

If we query a non-existent array index, we get undefined,

Example:

```
typeof a[90]
```

Output: undefined

If we take the above example into account, we can iterate over an array using the following:

```
for (var i = 0; i < a.length; i++)
{
   // Do something with a[i]
}
```

Arrays and string come with a number of methods:

Method Name	Description
substring()	Extracts the characters from a string, between two specified indices
concat()	Joins two or more strings, and returns a copy to the joined strings
join()	Joins all elements of an array into a string
pop()	Removes and returns the last item.
push()	Push adds one or more items to the end.
reverse()	Reverses the order of an element in an array

shift()	Removes the first element of an array and returns that element
slice()	Returns a sub-array.
sort()	Takes an optional comparison function.
splice()	Modify an array by deleting a section and replacing it with more items.
unshift()	Prep ends items to the start of the array.

3.2 Functions

Functions are the core component for understanding JavaScript. A simple basic function can be declared as:

```
function add(x, y) {
   var total = x + y;
   return total;
}
```

A JavaScript function can take 0 or more named parameters. The function body can contain as many statements and can declare its own variables which are local to that function. The return statement can be used to return a value at any time, or terminating the function. If no return statement is used (or an empty return with no value) JavaScript returns undefined.

We can call a function without passing the parameters it expects, in which case they will be set to undefined.

```
 add()
NaN                          // We can't perform addition on undefined
 add(2, 3, 4)
5                            // added the first two; 4 was ignore
```

Functions have access to an additional variable inside their body called arguments, which is an array-like object holding all of the values passed to the function. Let's re-write the add function to take as many values:

```
function add() {
   var sum = 0;
   for (var i = 0, j = arguments.length; i < j; i++) {
   sum += arguments[i];
 }
   return sum;
 }
```

Output:

add(2, 3, 4, 5)

14

That's really not any more useful than writing 2 + 3 + 4 + 5 though. Let's create an averaging function:

```
function avg() {
    var sum = 0;
    for (var i = 0, j = arguments.length; i < j; i++) {
    sum += arguments[i];
    }
    return sum / arguments.length;
}
```

Output:

avg(2, 3, 4, 5)

3.5

In this example, avg() function takes a comma separated list of arguments — but if we want to find the average of an array we could just write the function as follows:

```
function avgArray(arr) {
    var sum = 0;
    for (var i = 0, j = arr.length; i < j; i++)
   {
    sum += arr[i];
    }
    return sum / arr.length;
    }
```

Output:

avgArray([2, 3, 4, 5])

3.5

JavaScript allows reusability feature by calling a function with an arbitrary array of arguments, using the apply() method of any function object.

Output:

avg.apply(null, [2, 3, 4, 5])

3.5

JavaScript allows us to create anonymous functions.

```
var avg = function() {
var sum = 0;
 for (var i = 0, j = arguments.length; i < j; i++) {
 sum += arguments[i];
 }
 return sum / arguments.length;
 }
```

This is semantically equivalent to the function avg() form.

JavaScript allows us to call functions recursively. This is particularly useful for dealing with tree structures, such as we get in the browser DOM (Document Object Model).

```
function count Chars(elm)
 {
    if (elm.nodeType == 3) {                    // TEXT_NODE
    return elm.nodeValue.length;
  }
    var count = 0;
    for (var i = 0, child; child = elm.childNodes[i]; i++) {
    count += count Chars(child);
  }
    return count;
  }
```

Inner Functions

JavaScript function declarations are allowed inside other functions. An important feature of nested functions in JavaScript is that they can access variables in their parent function's scope:

```
function betterExampleNeeded() {
    var a = 1;
    function oneMoreThanA() {
     return a + 1;
    }
    return oneMoreThanA();
    }
```

List of Some Common Programs Using JavaScript

A Simple Program: Printing a Line of Text in a Web Page Using JavaScrip

```
<html>
<body>
(h1>my first web page</h1>
<script>
Document.write(“Hello World!”)
</script>
</body>
</html>
```

Output:

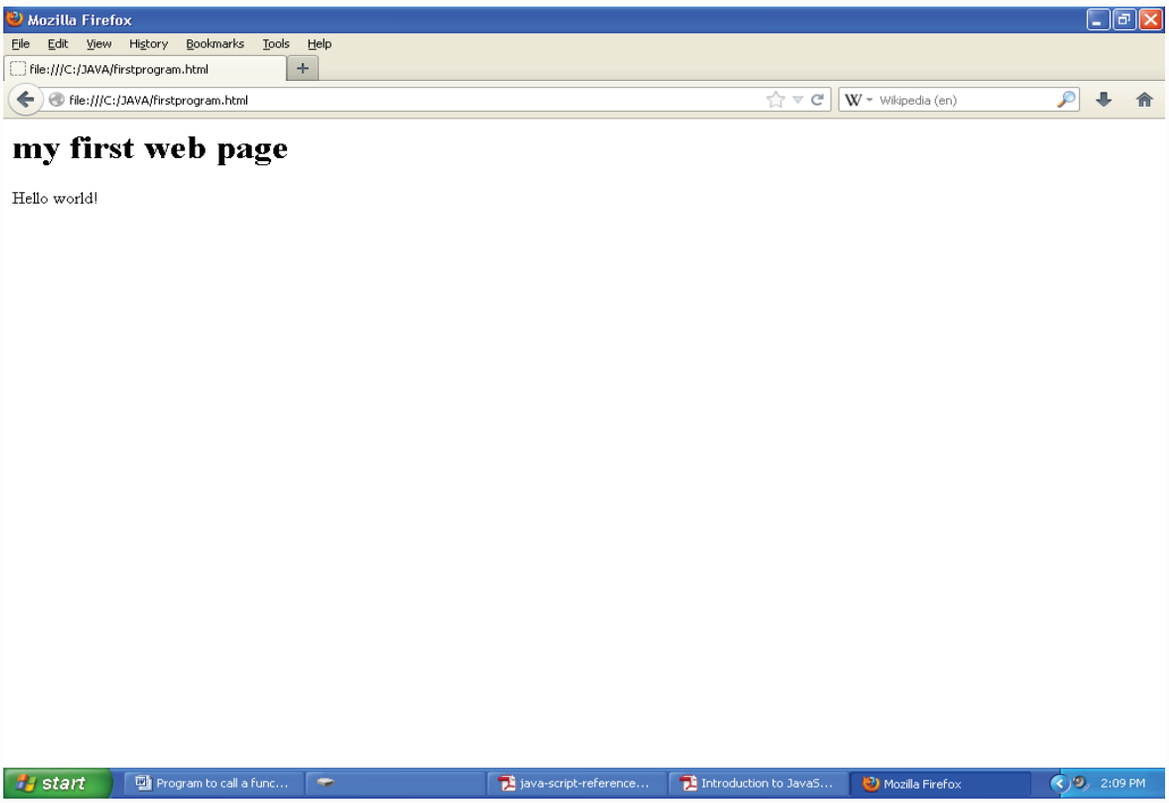

Figure – 3.1

Variables: JavaScript variables are “containers” for storing information. Creating a variable is most often referred to as “declaring” a variable.

- To declare variables, use the keyword var and the variable name:

  var username.

- To assign values to variables, add an equal sign and the value:

  var userName = "JOHN"

  var price = 50

Example:

```
//Program to show date and time. This is a comment line.
<html>
<body>
<script>
var d=new Date();
document.write(d);
</script>
</body>
</html>
```

Output:

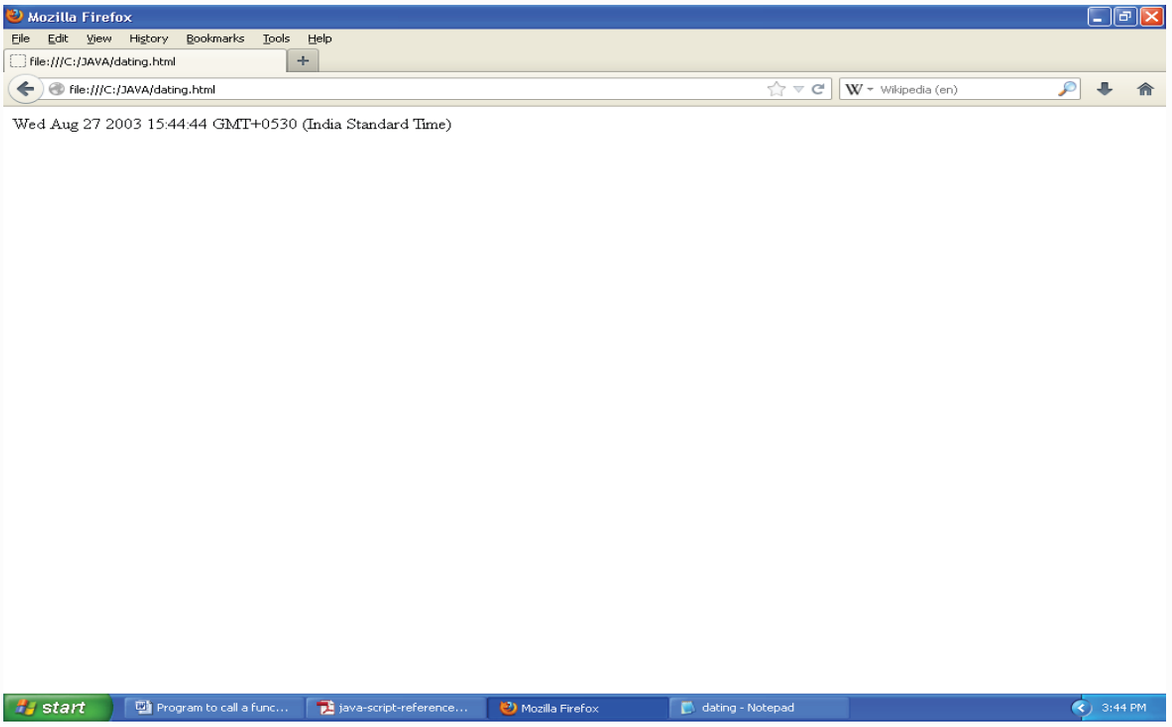

Figure – 3.2

Functions - Predefined

A function is a block of code that will be executed when someone calls it. With functions, you can give a name to a whole block of code, allowing you to reference it from anywhere in your program. JavaScript has built-in functions for several predefined operations. Here are three some functions.

- alert("message")
- confirm("message")
- prompt("message")

Example: Alertbox

```
//Program to show the alertbox.
<html>
<head>
<script>
function myFunction()
{
alert("HELLO!");
}
</script>
</head>
<body>
<input type="button"onclick="myFunction()"value="show me the alert box"/>
</body>
</html>
```

Output:

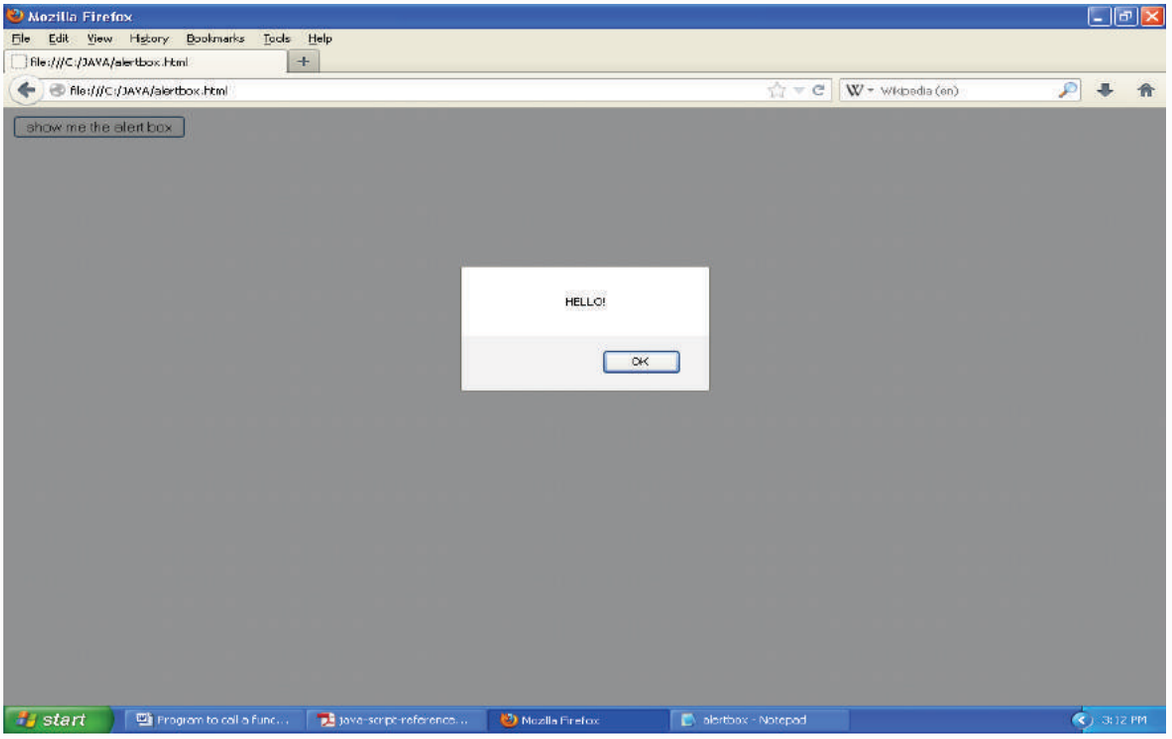

Figure – 3.3

```
//Program to show the confirm box.
<html>
<head>
<p>Click the button to display a confirm box.</p>
<button onclick="myFunction()">try it</button>
<p id="demo"></p>
<script>
function myFunction()
{var x;
var r=confirm("press a button");
if(r==true)
{
x="you pressed OK";
}
else
{
x="you pressed cancel";
}
document.getElementById("demo").innerHTML=x;
}
</script>
</body>
</html>
```

Output:

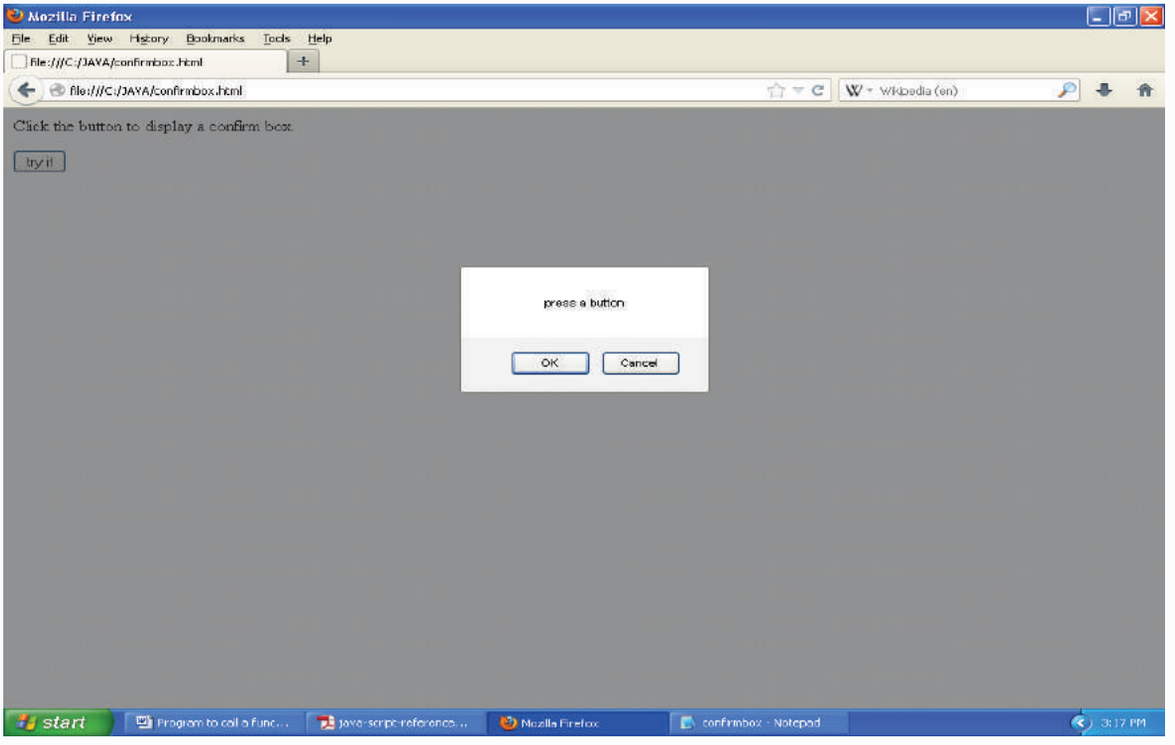

Figure – 3.4

Output continues…….

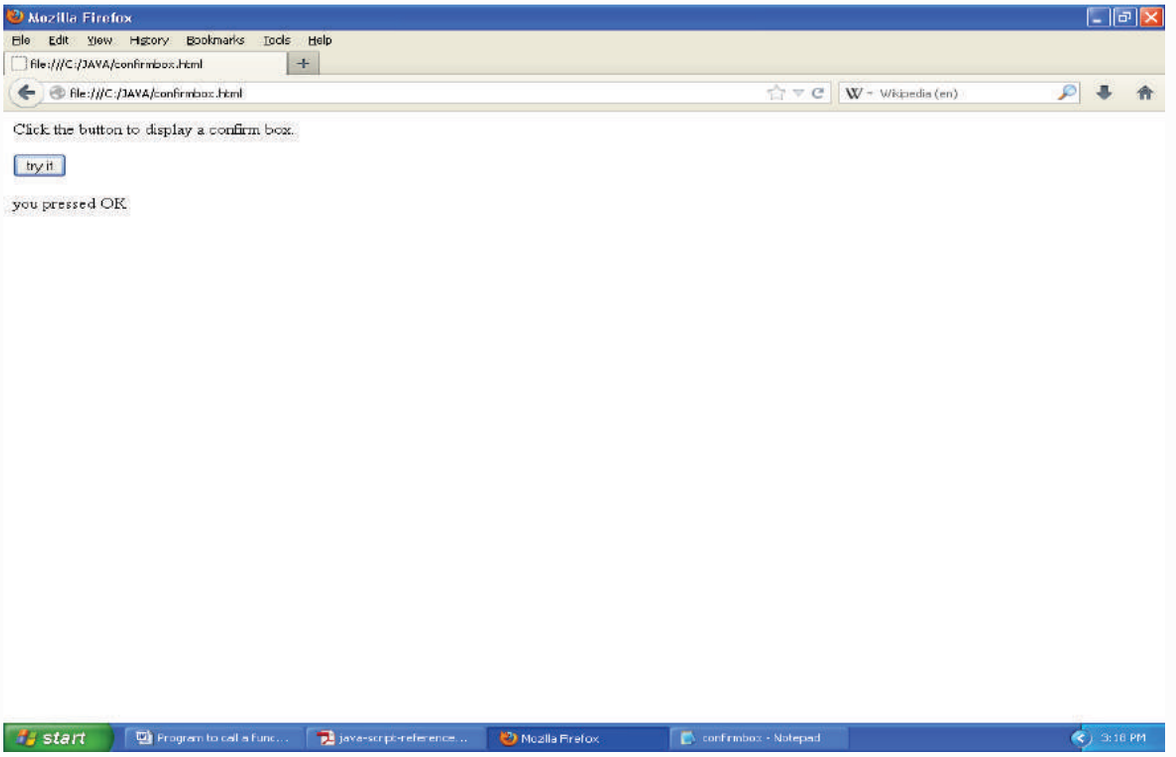

Figure – 3.5

```
//Program to show the prompt box.
<html>
<body>
<p><strong>Click the button to demonstrate the prompt box.</strong></p>
<button onclick="myFunction()">try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
var x;
var person;
person=prompt("please enter your name"," ");
if(person!=null)
{
x=("hello" +person+ "!how are u today?");
}
document.getElementById("demo").innerHTML=x;
}
```

</script>
</body>
</html>

Output:

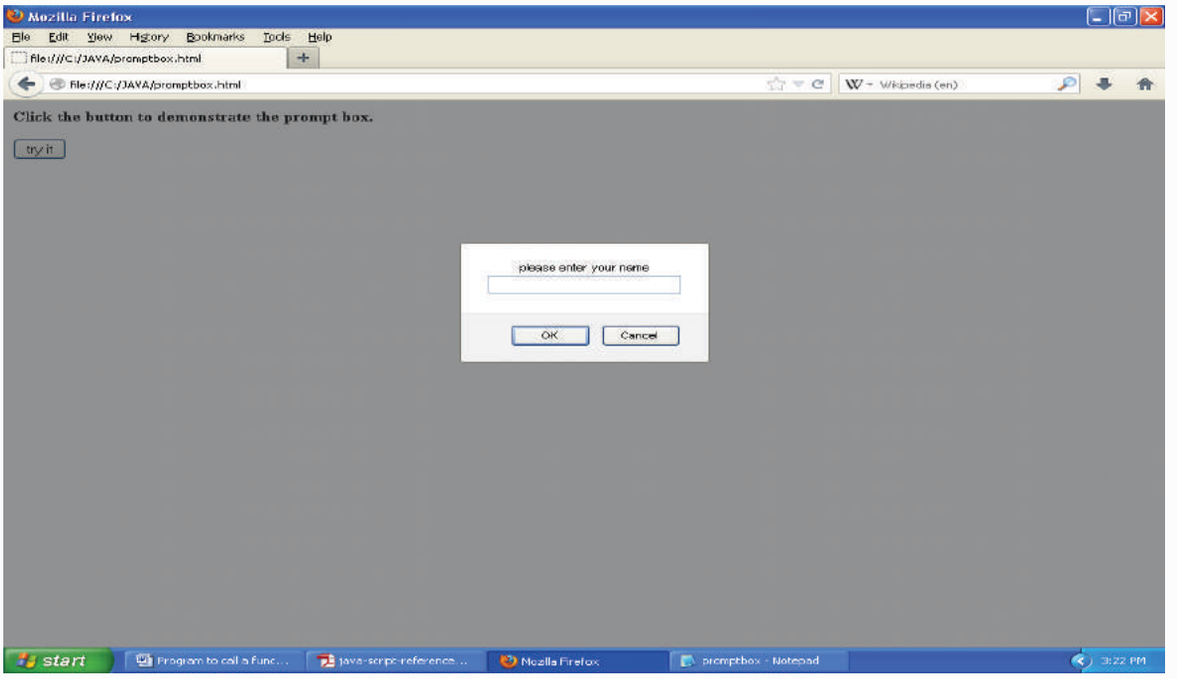

Figure – 3.6

Output continues…..

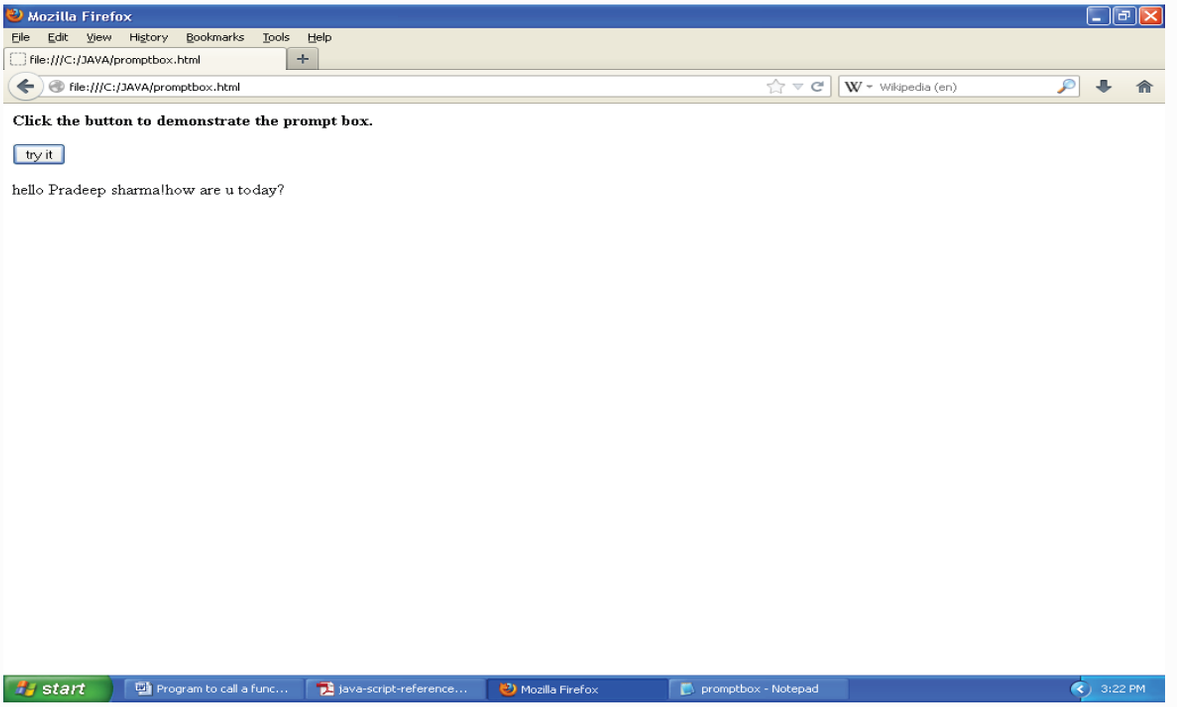

Figure – 3.7

Functions – User Defined

With user-defined functions, you can name a block of code and call it when you need it. You define a function in the HEAD section of a web page. It is defined with the function keyword, followed by the function name and any arguments.

```
function functionName(argument)
{
statements
}
//Program to call a function.
<html>
<head>
<script>
function myFunction()
{
confirm("Welcome!!!");
}
</script>
</head>
<body>
<button onclick="myFunction()">try it</button>
<p><strong>By clicking the above button,a function will be called.The function will confirm a message.</strong></p>
</body>
</html>
```

Output:

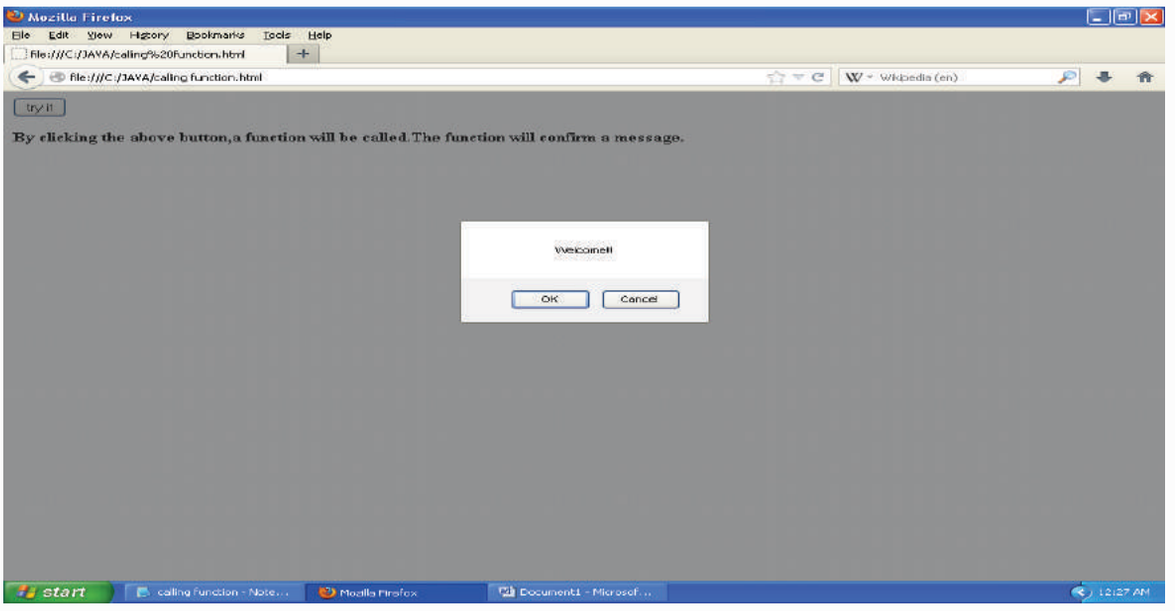

Figure – 3.8

```
//Program to show function with an argument.
<html>
<body>
<p>click the button to call a function with arguments</p>
<button onclick="myFunction('The Earth','around the Sun')">try it</button>
<script>
function myFunction(name,object)
{
alert(name+"revolves"+object);
}
</script>
</body>
</html>
```

Output:

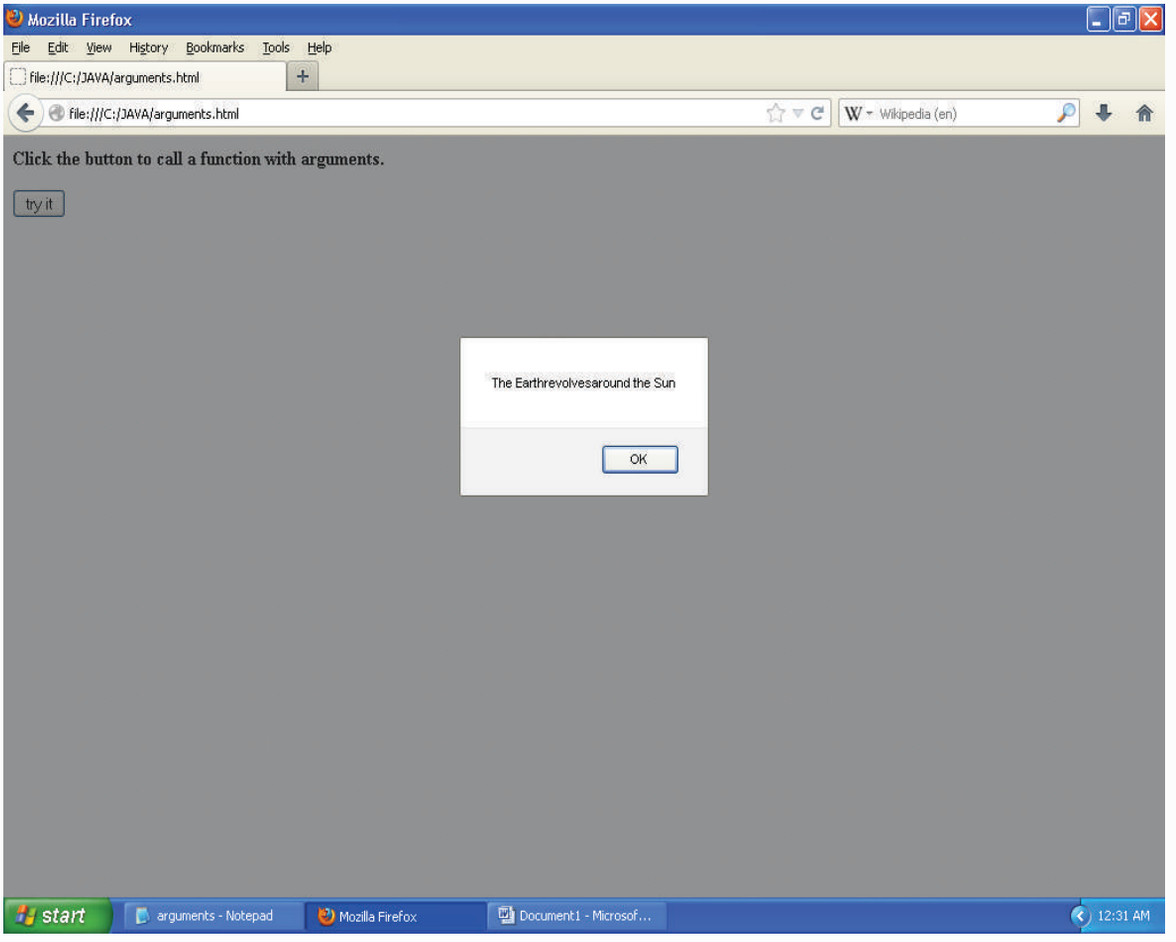

Figure – 3.9

```
//Function that returns a value.
<html>
<head>
<script>
function myFunction()
{
return("Have a nice day!!!");
}
</script>
</head>
<body>
<script>
document.write(myFunction());
</script>
</body>
</html>
```

Output:

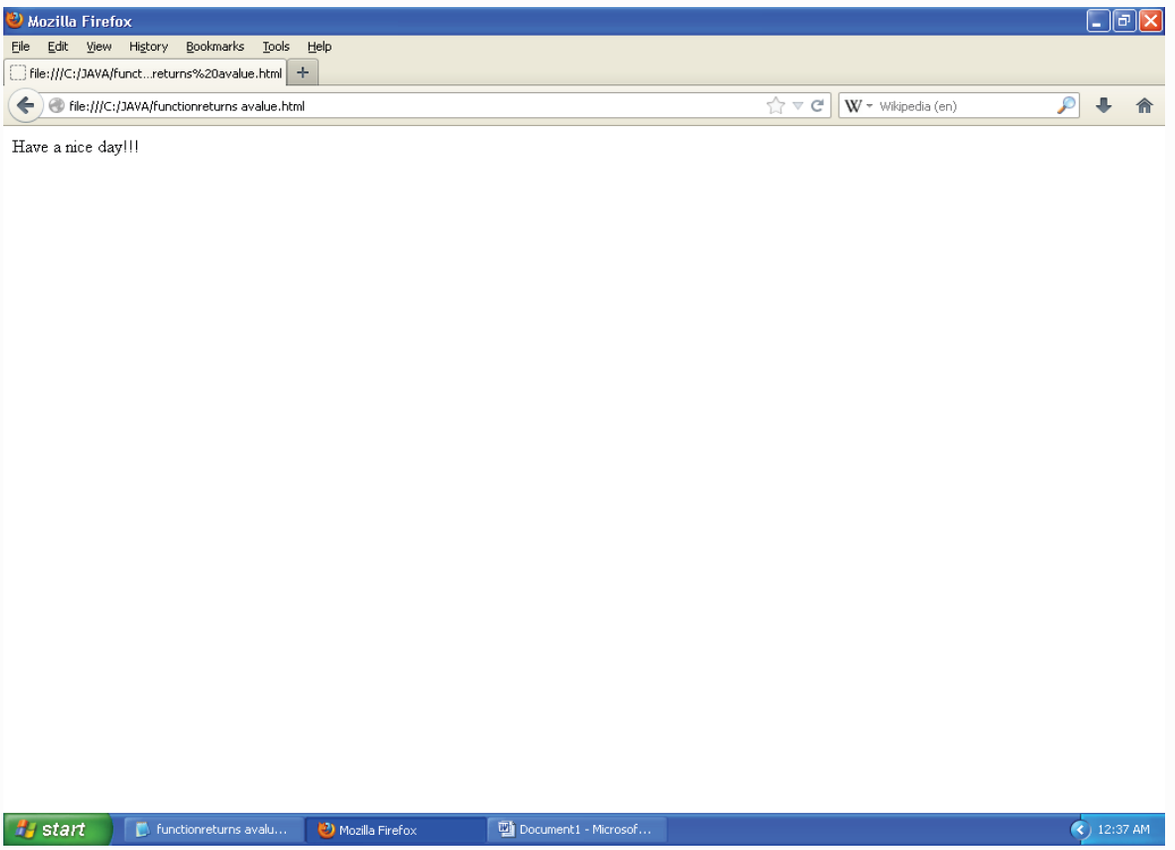

Figure – 3.10

```
//Function with arguments that returns a value.
<html>
<body>
<p>Function with arguments that returns the result</p>
<p id="demo"></p>
<script>
function myFunction(x,y)
{
return x+y;
}
document.get ElementById("demo").innerHTML=myFunction(5,8);
</script>
</body>
</html>
```

Output:

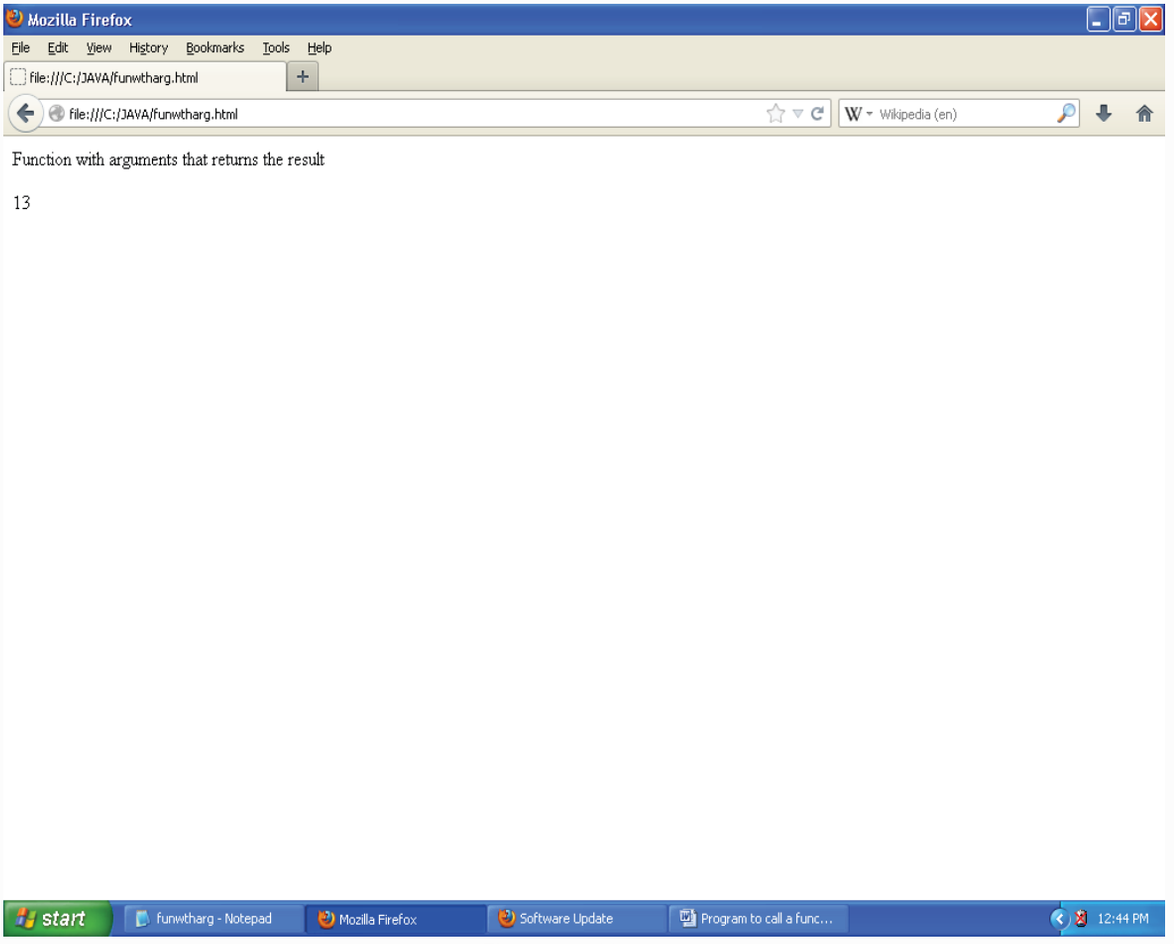

Figure – 3.11

3.3 Objects

JavaScript supports programming with objects. Objects are a way of organizing the variables. The different screen elements such as Web pages, forms, text boxes, images, and buttons are treated as objects.

Properties and Methods

Every object has its own properties and methods.

- Properties define the characteristics of an object. Examples are color, length, name, height, width etc.
- Methods are the actions that the object can perform or that can be performed on the object. Examples are alert, confirm, write, open, close etc.

Naming Objects

- Objects are organized in a hierarchy: to refer to an object use object Name
- To refer to a property of an object use: object Name. property Name
- To refer to a method of an object use: object Name. method Name()

Built-In Objects

Some of the built-in language objects of JavaScript offer more advanced operations such as:

- String – provides for string manipulation,
- Math-provides for maths calculations,
- Array-provides the collection of similar data types.

3.4 String Object

The String object provides methods and properties for string manipulation and formatting.

Format: stringName.method()

```
//Program to find the length of the array
<html>
<body>
<p>Length of the given string =
<script>
var txt="Hello world";
document.write(txt.length);
</script>
</p>
</body>
</html>
```

Output:

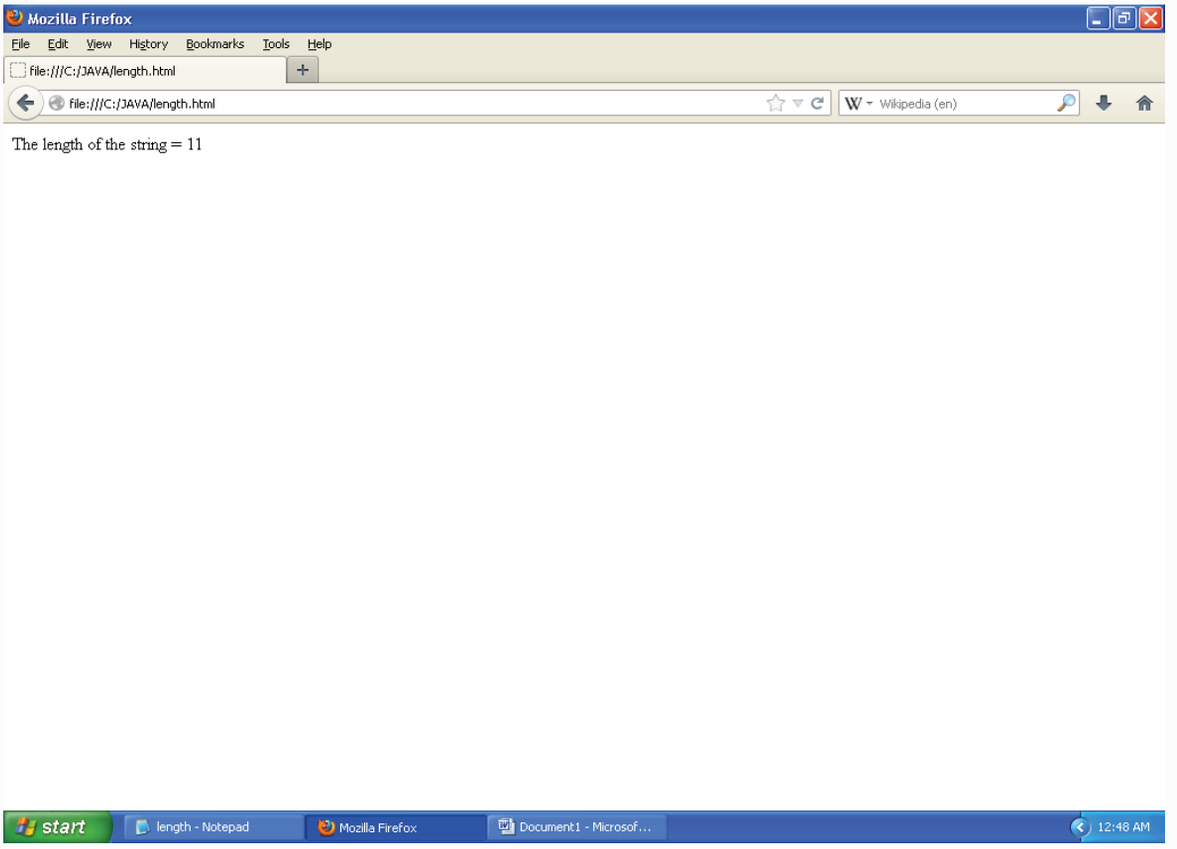

Figure – 3.12

```
//Program to find the position of the first occurrence of a text in a string using indexOf()
<html>
<body>
<p id="demo"><strong>click the button to locate where in the string a specified value occurs.</strong></p>
<button onclick="myFunction()">try it</button>
<script>
function myFunction()
{
var str="hello! welcome to my world";
var n=str.indexOf("world");
document.getElementById("demo").innerHTML=n;
}
</script>
</body>
</html>
```

Output:

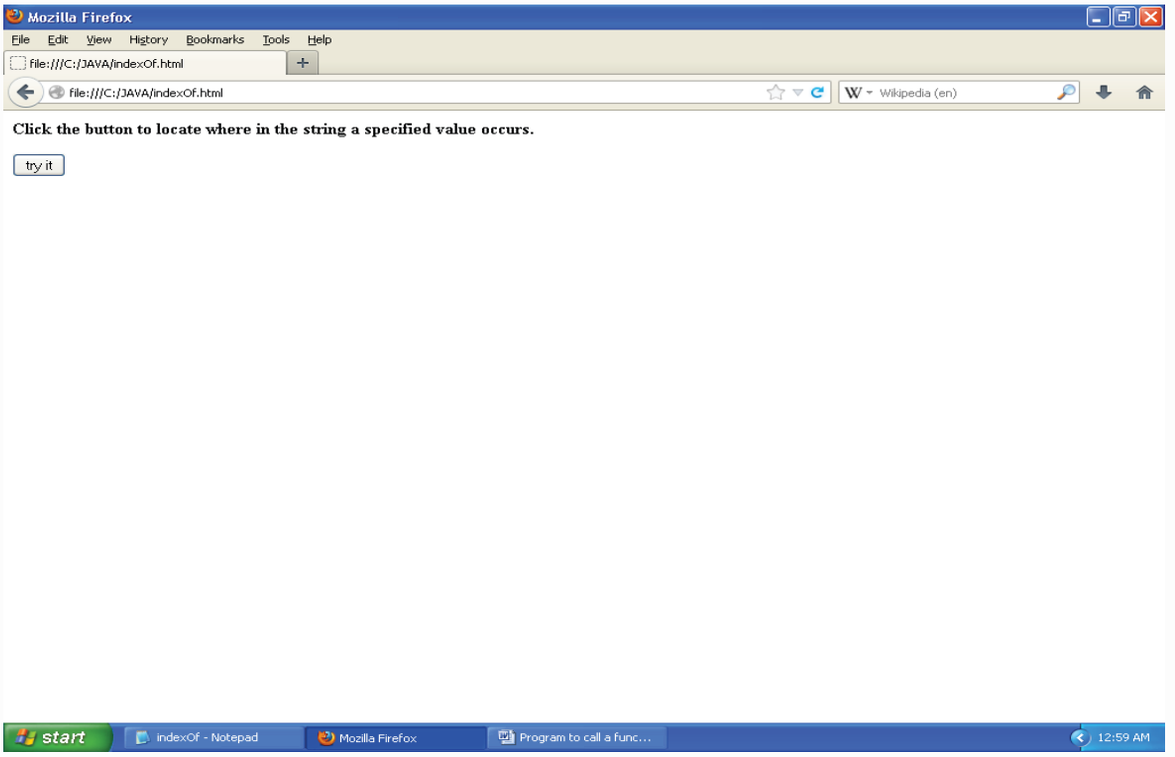

Figure – 3.13

Output continued…….

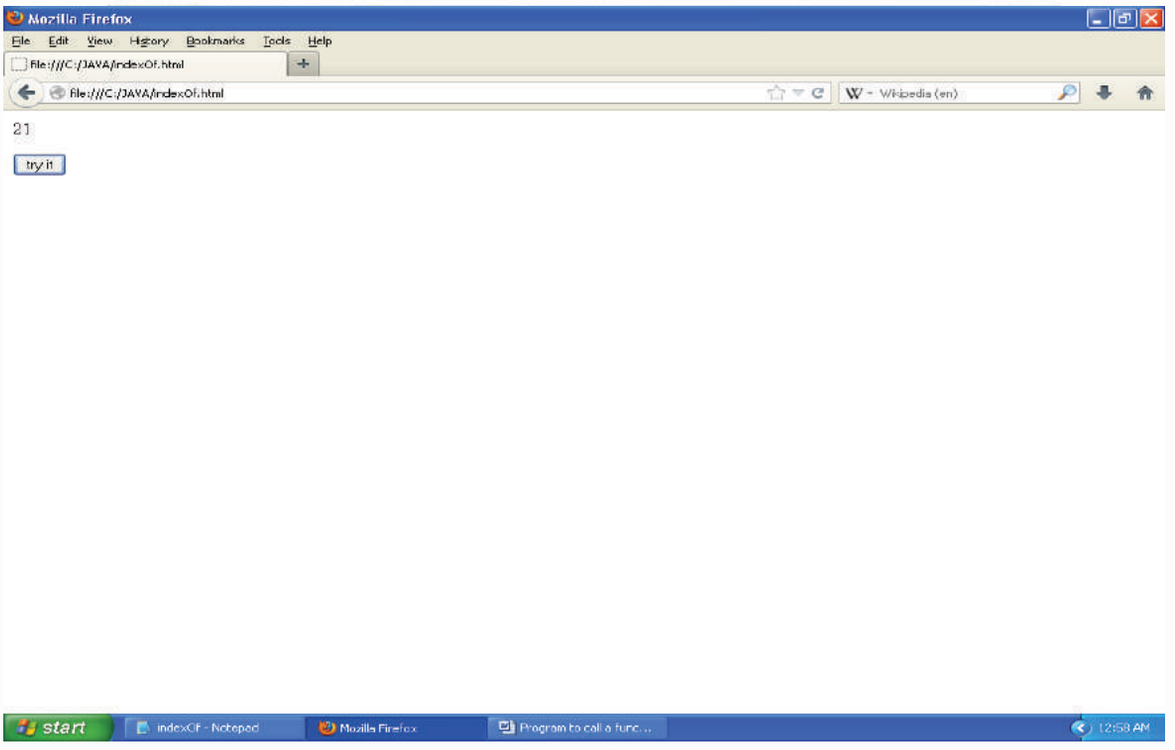

Figure – 3.14

```
//Program to search for a text in a string and return the text if found using match()
<html>
<body>
<script>
var str="Honesty is the best policy";
document.write(str.match("policy")+"<br>");
document.write(str.match("Police")+"<br>");
document.write(str.match("pollicy")+"<br>");
document.write(str.match("policy")+"<br>");
</script>
</body>
</html>
```

Output:

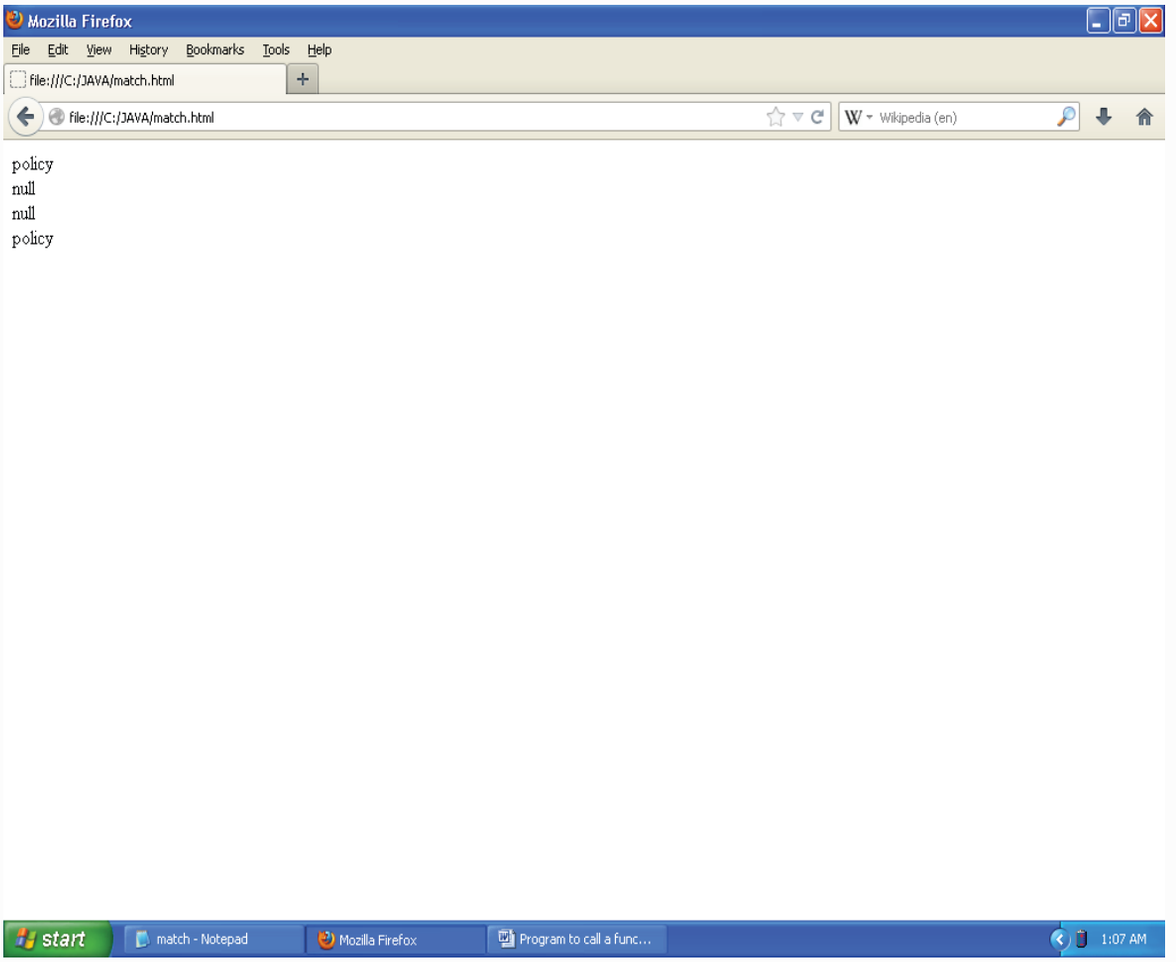

Figure – 3.15

```
//Program to replace characters in a string using replace()
   <html>
<body>
<p>click the button to replace the characters</p>
<p id="demo">hello prachi</p>
<button onclick="myFunction()">try it</button>
<script>
function myFunction()
{
var str=document.getElementById("demo").innerHTML;
var n=str.replace("hello","good morning");
document.getElementById("demo").innerHTML=n;
}
</script>
</body>
</html>
```

Output:

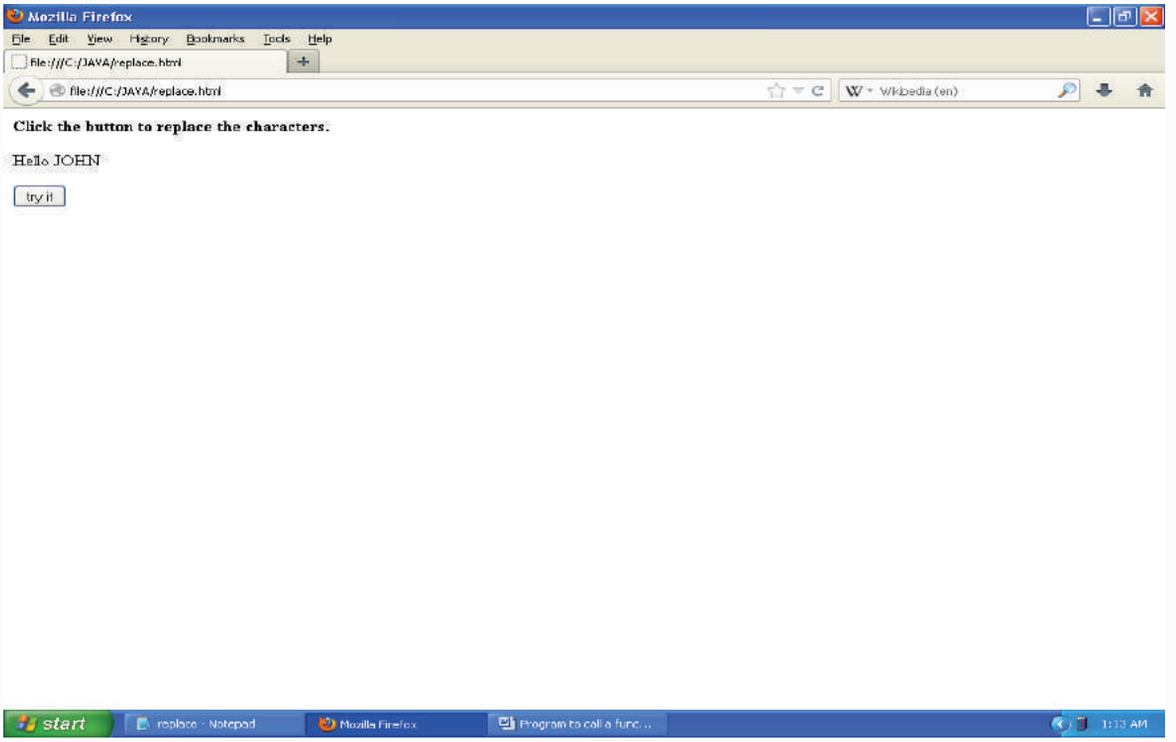

Figure – 3.16

When we click the button the output will be……

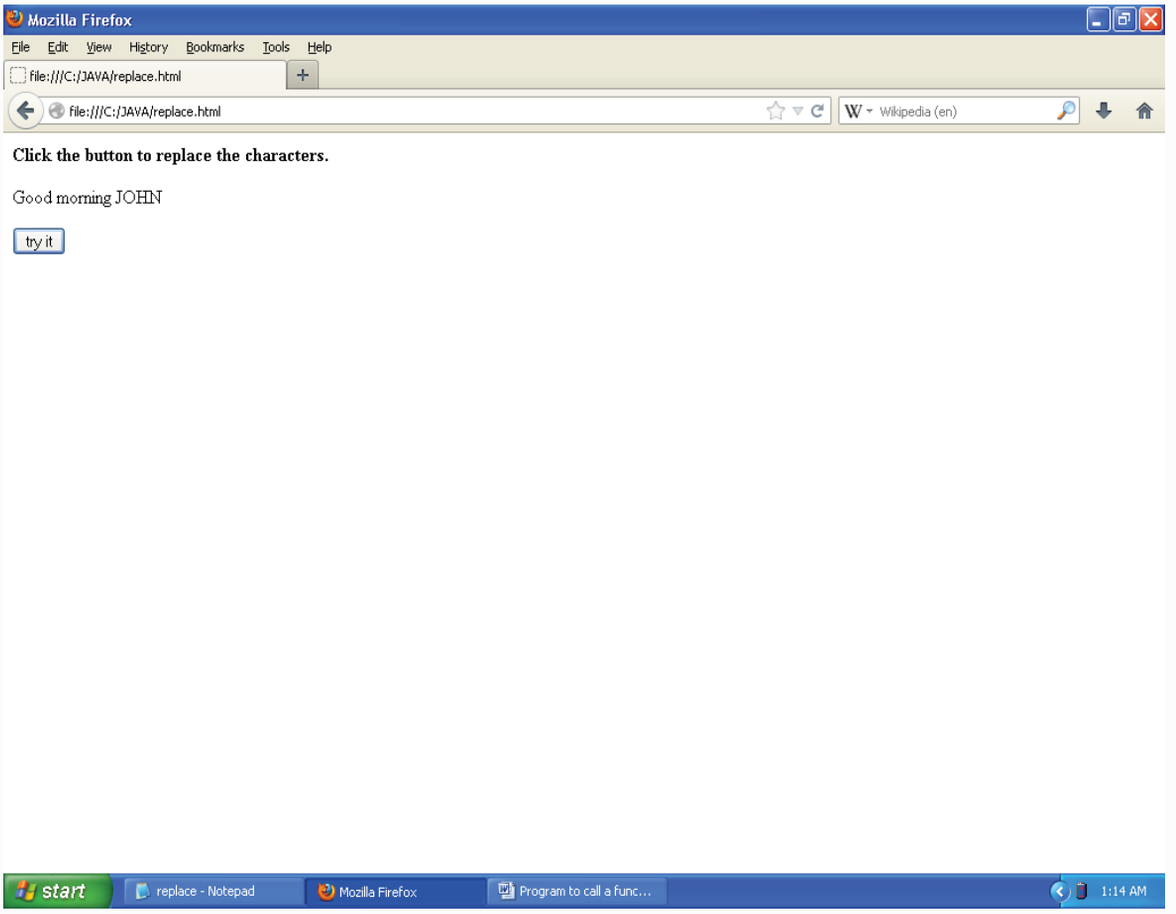

Figure – 3.17

3.5 Math Object

The Math object provides methods for many mathematical calculations like abs(), log(), pow(), random(), round(), sqrt() etc.

Format: Math.method(#)

- **round() method**

  The round() method returns a number to the nearest integer.

  Syntax:

  Math.round()

  //Program to round off any number using round()

  <html>

  <body>

  click the button to round the no.to its nearest integer.

```
<button onclick="myFunction()">try it</button>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML=Math.round(8.7896);
}
</script>
</body>
</html>
```

Output:

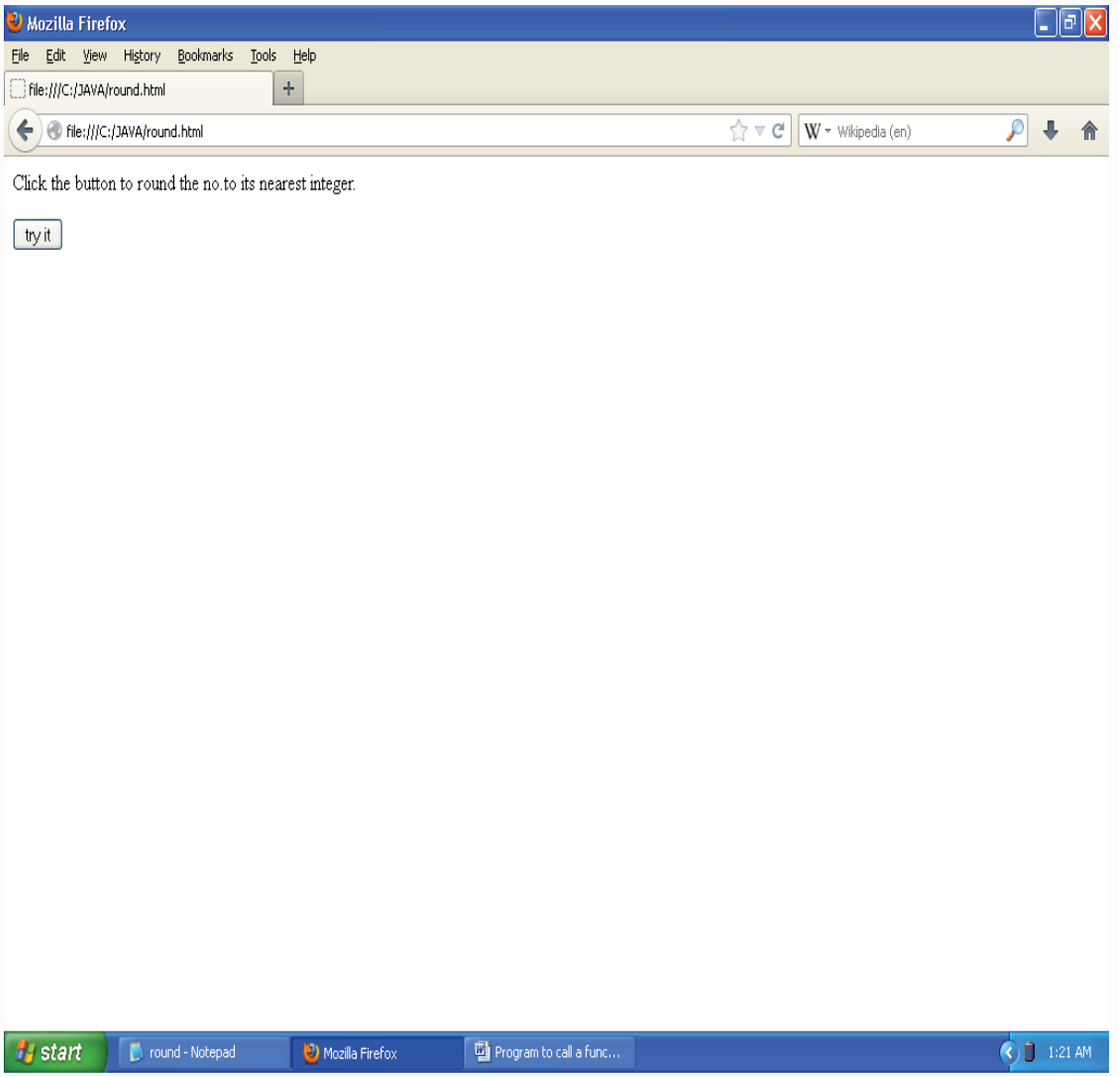

Figure – 3.18

Output continues….

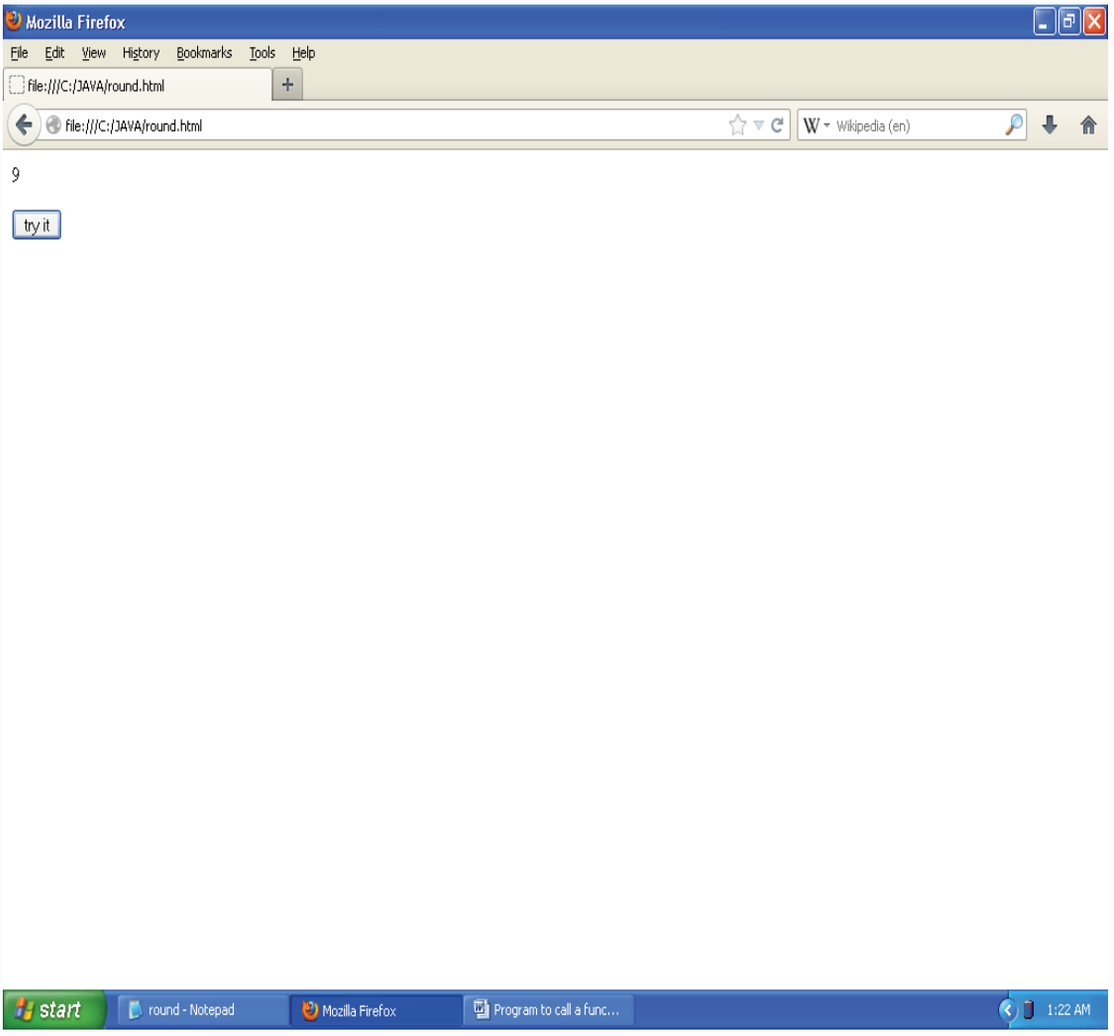

Figure – 3.19

- **random()**

The random() method returns a random number from 0(inclusive) up to but not including 1(exclusive).

Syntax:

Math.random()

```
//Program to return a value random number between 0 and 1 using random()
<html>
<body>
<p id="demo">click the button to display a number</p>
<button onclick="myFunction()">try it</button>
<script>
```

```
function myFunction()
{
document.getElementById("demo").innerHTML=Math.random();
}
</script>
</body>
</html>
```

Output:

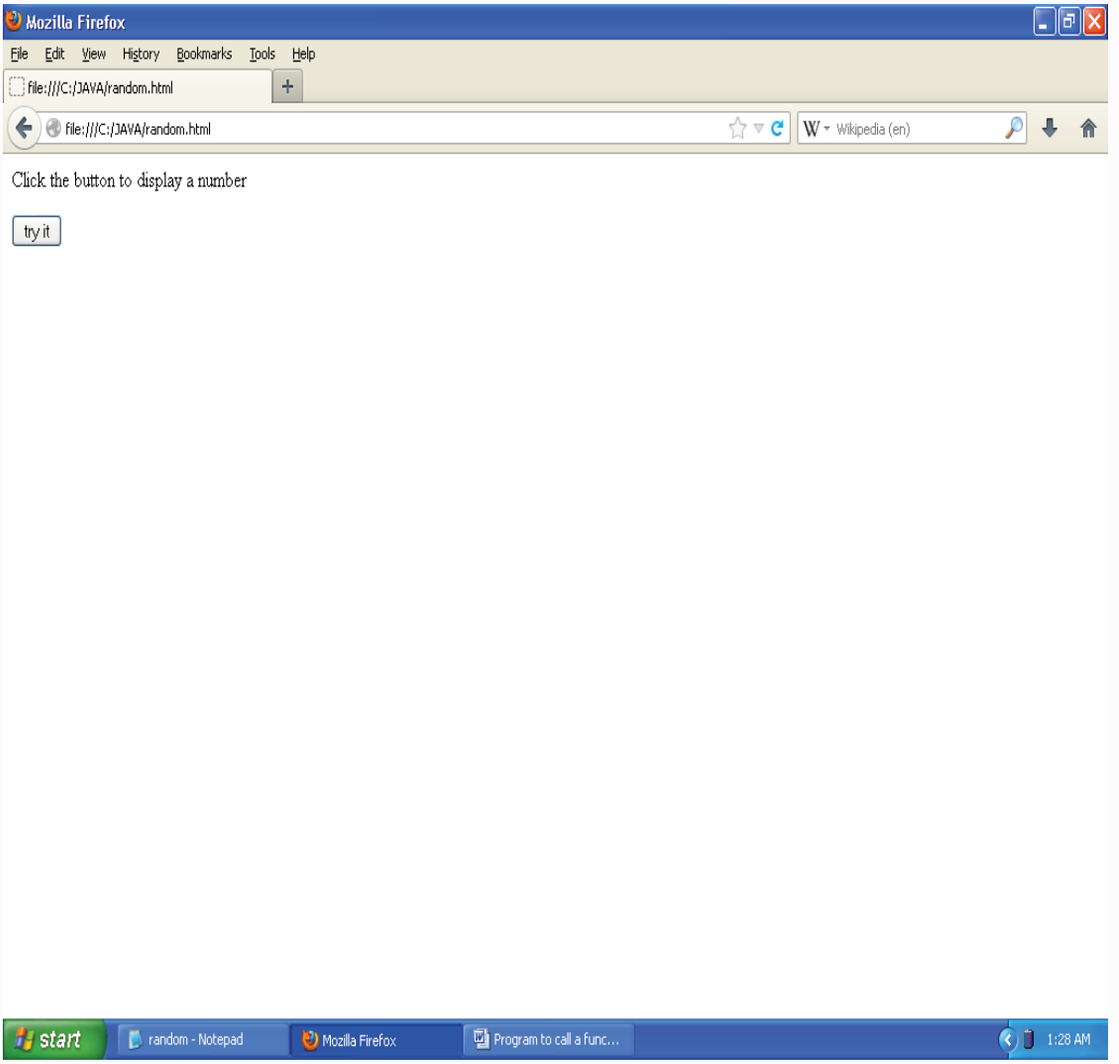

Figure – 3.20

Output continues…….

On pressing try it key continuously we got various random numbers

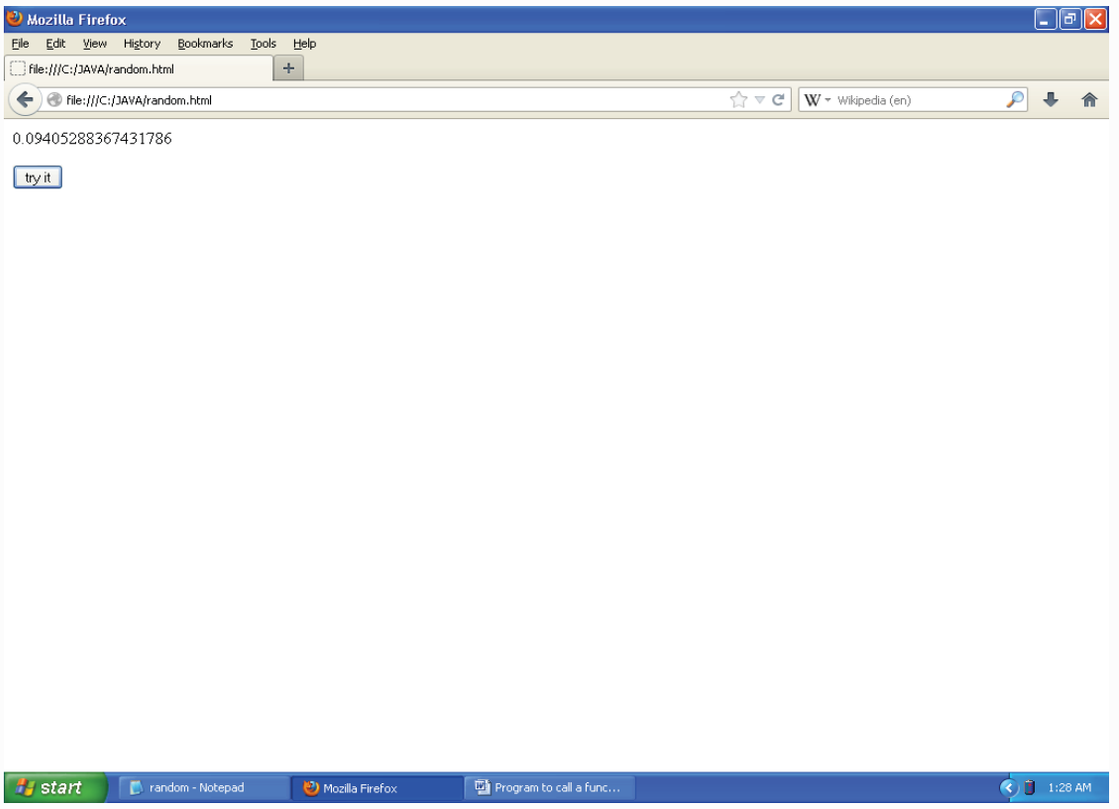

Figure – 3.21

- **max()**

The max() method returns the number with the highest value.

Syntax:

Math.max(n1,n2,n3……..nx)

//Program to return the number with highest value of two specified numbers using max()

```
<html>
<body>
<p id="demo">Click the button to return the highest no. between 5 and 10.</p>
<button onclick="myFunction()">try it</button>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML=Math.max(5,10);
}
</script>
</body>
</html>
```

Output:

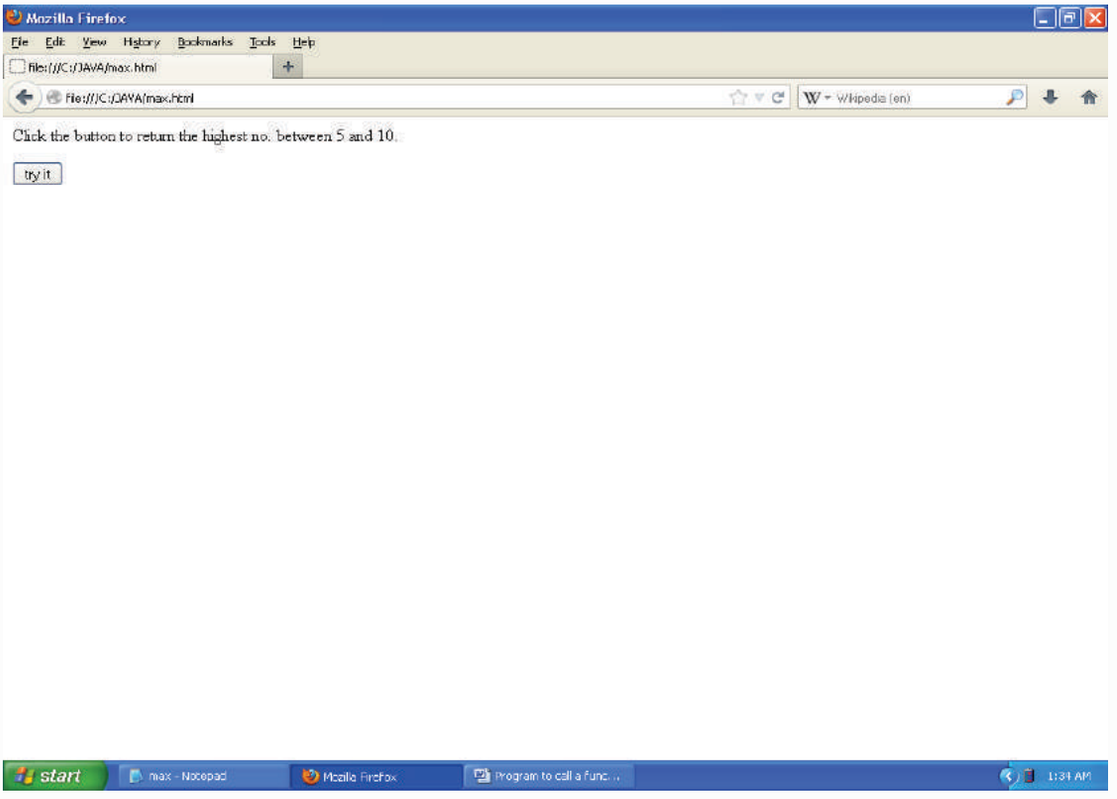

Figure – 3.22

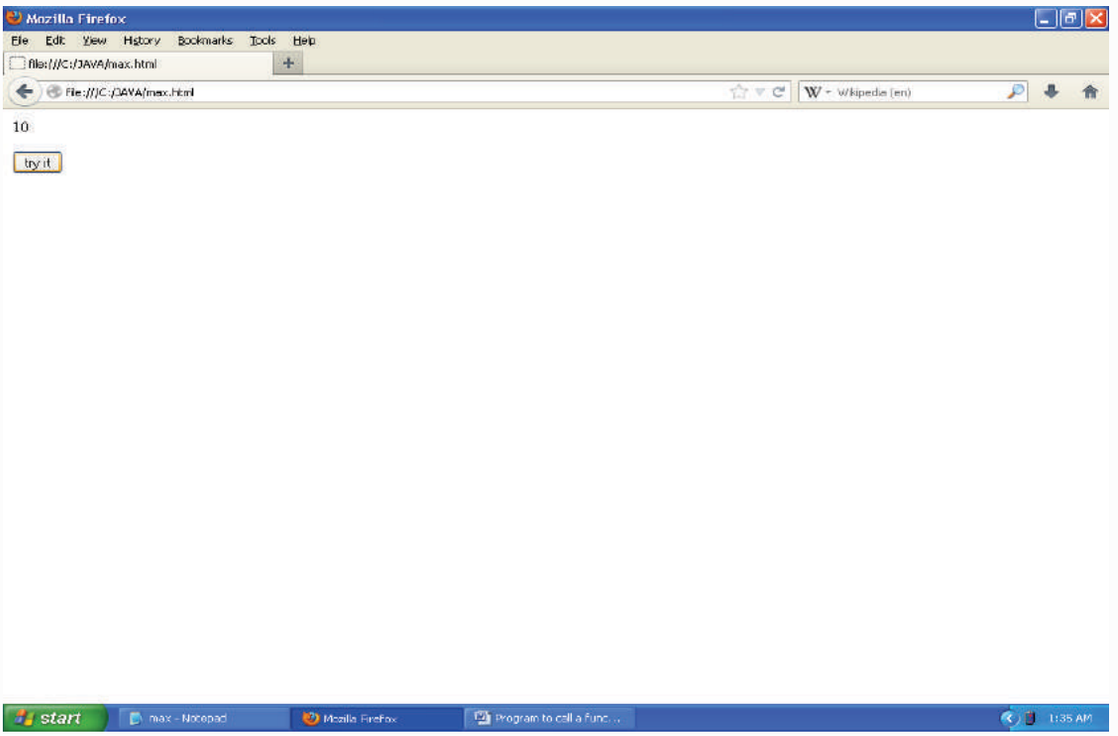

Figure – 3.23

- **min()**

The min() method returns the number with the lowest value.

Syntax:

Math.min(n1,n2,n3…….nx)

//Program to return the number with the lowest value of two specified number using min().

```
<html>
<body>
<p id="demo">Click the button to return the lowest no. between 77 and 9.</p>
<button onclick="myFunction()">try it</button>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML=Math.min(77,9);
}
</script>
</body>
</html>
```

Output:

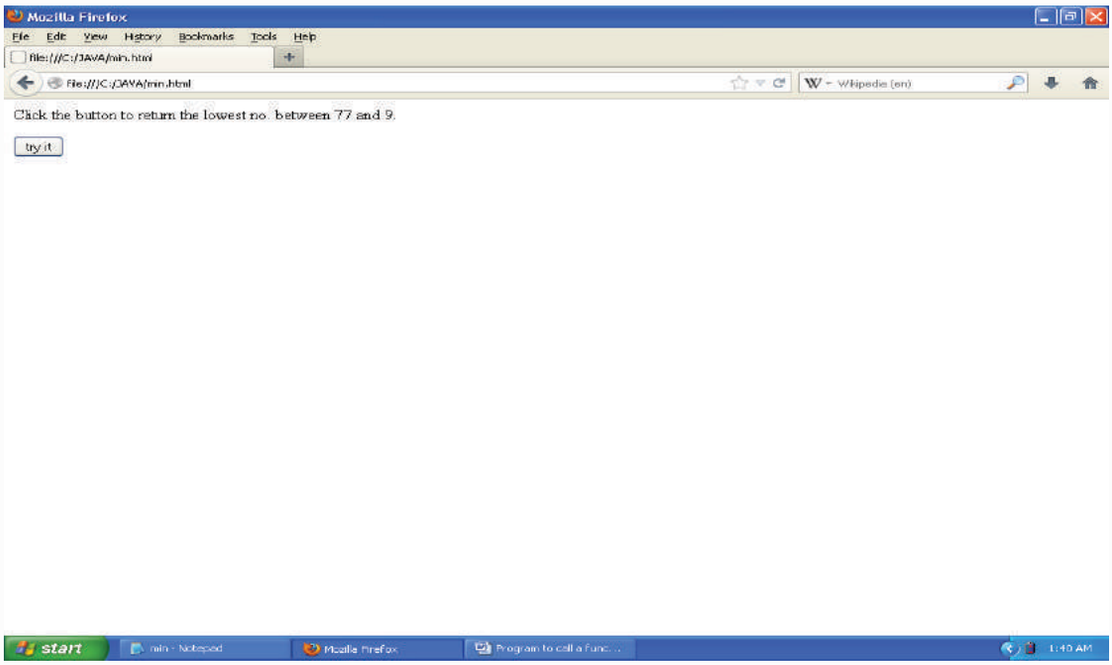

Figure – 3.24

Output continues……

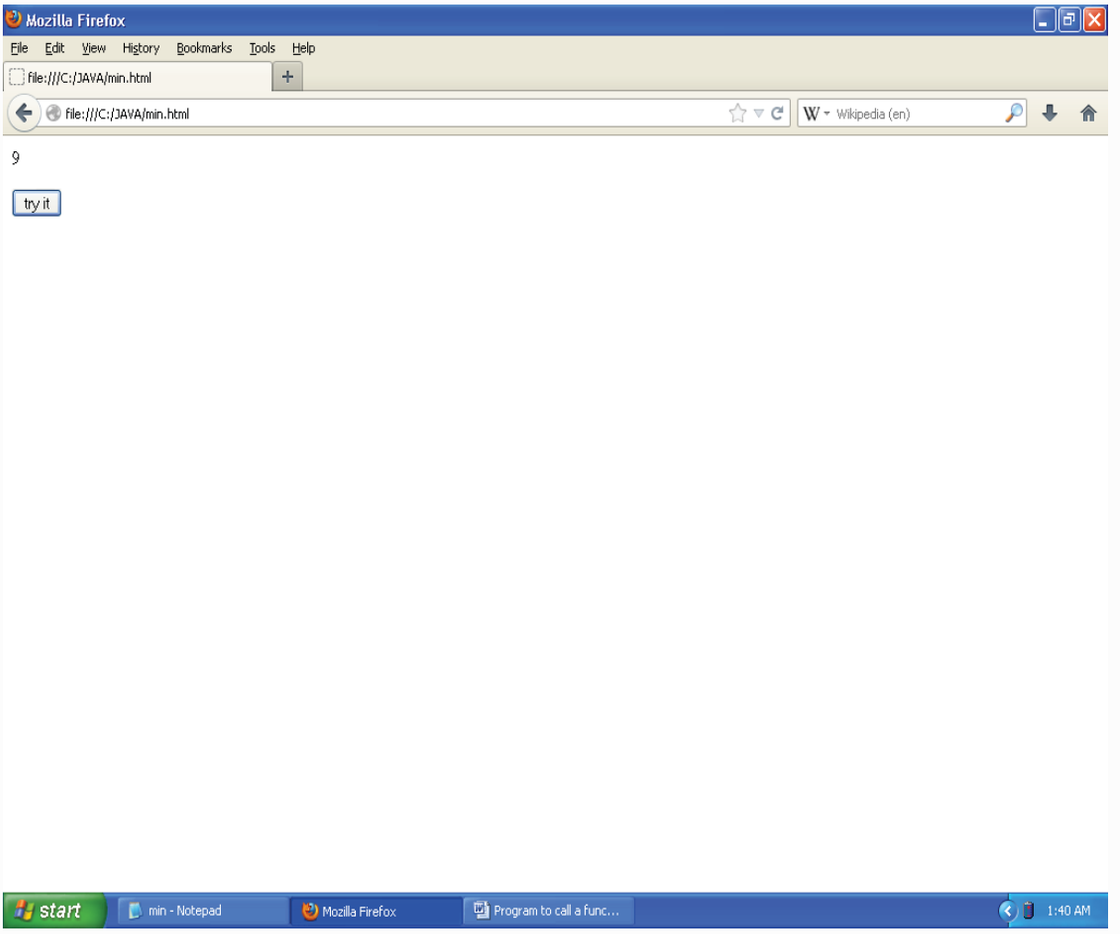

Figure – 3.25

3.6 Array object:

The array object is used to store multiple values in a single variable.

```
//Program to create an array.
<html>
<body>
<script>
var i;
var fruits=new Array();
fruits[0]="apple";
fruits[1]="banana";
fruits[2]="orange";
for(i=0;i<fruits.length;i++)
```

```
{
document.write(fruits[i]+"<br>");
}
</script>
</body>
</html>
```

Output:

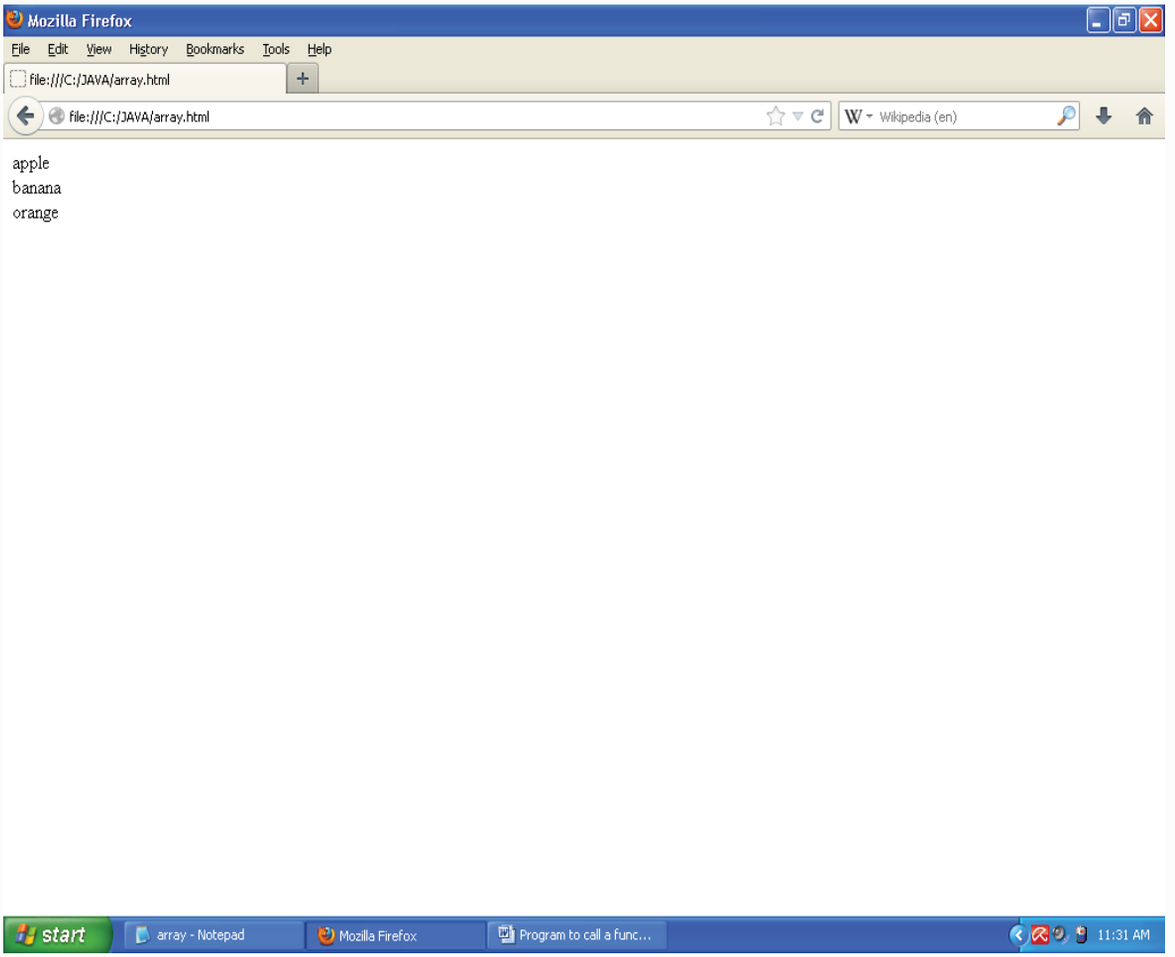

Figure – 3.26

```
//Program to join two arrays using concat().
<html>
<body>
<p id="demo">Click the button to join three arrays</p>
<button onclick="myFunction()">Click me</button>
```

```
<script>
function myFunction()
{
var fruits=["Apple","Orange"];
var vegetables=["Cucumber","Carrot","Raddish"];
var grains=["Wheat","Maize"];
var mix=fruits.concat(vegetables,grains);
var x=document.getElementById("demo");
x.innerHTML=mix;
}
</script>
</body>
</html>
```

Output:

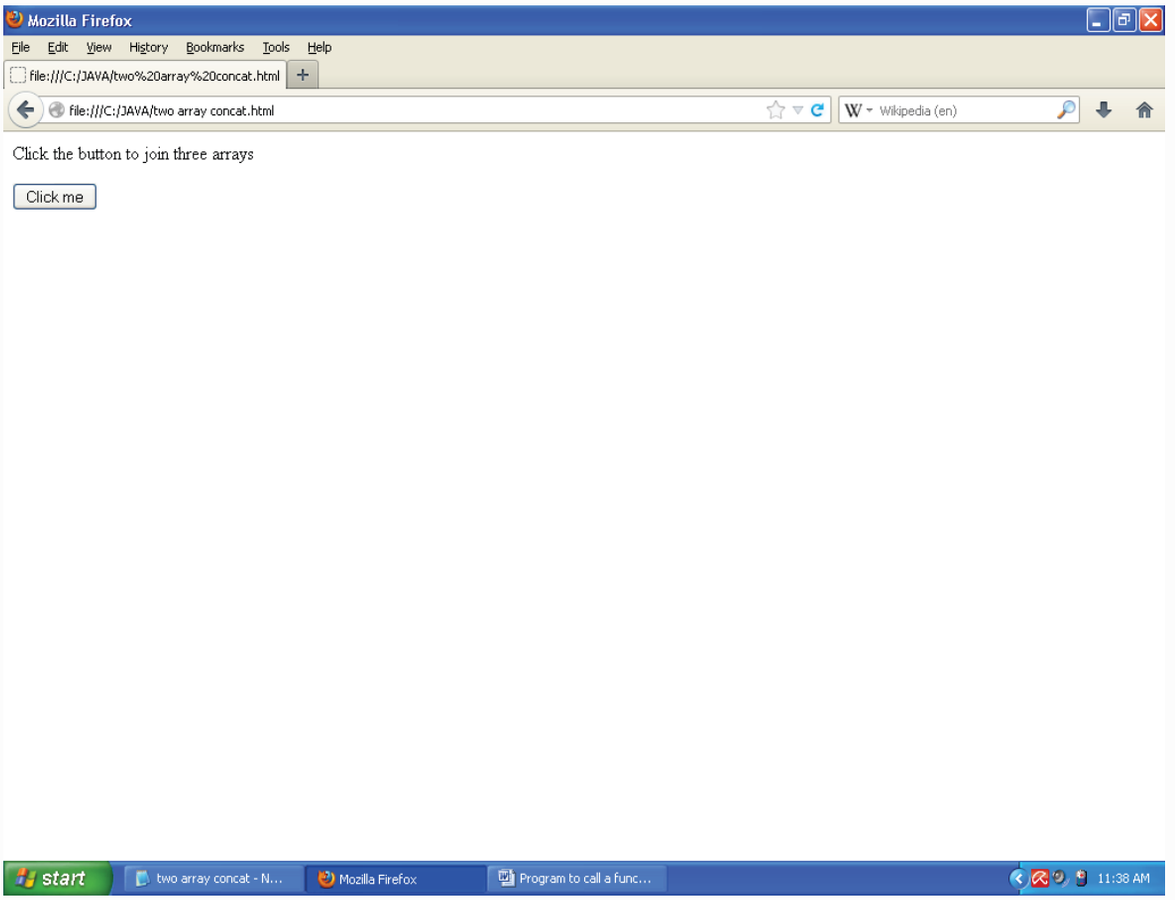

Figure – 3.27

Output continues……..

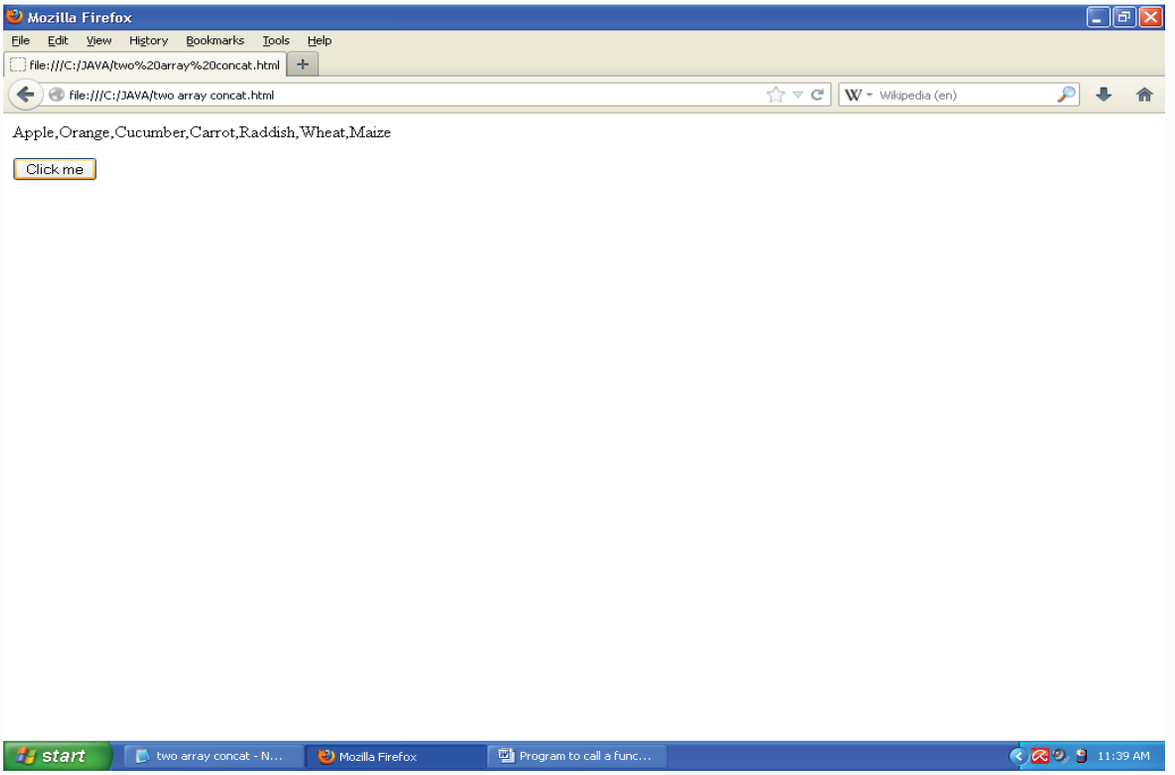

Figure – 3.28

```
//Program to remove the last element from the array by using pop()
<html>
<body>
<p id="demo">Click the button to remove the last array element.</p>
<button onclick="myFunction()">Click me</button>
<script>
var name=["John","Mary","Oliver","Alice"];
function myFunction()
{
name.pop();
var x=document.getElementById("demo");
x.innerHTML=name;
}
</script>
</body>
</html>
```

Output:

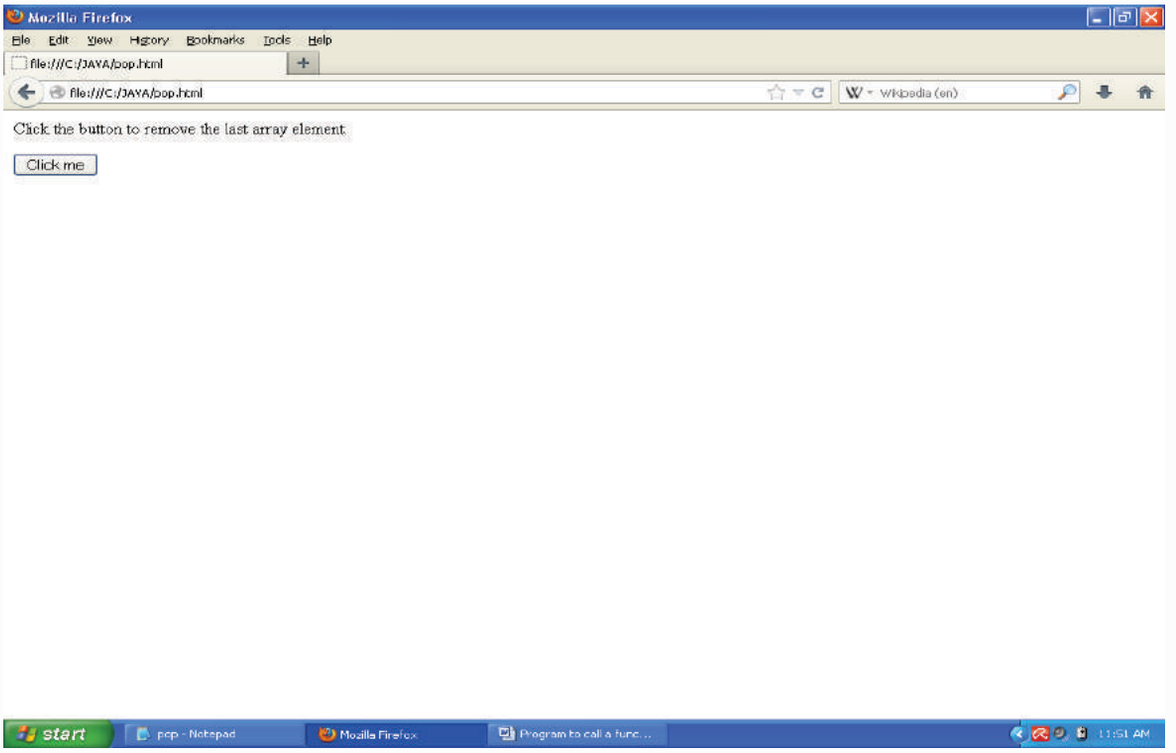

Figure – 3.29

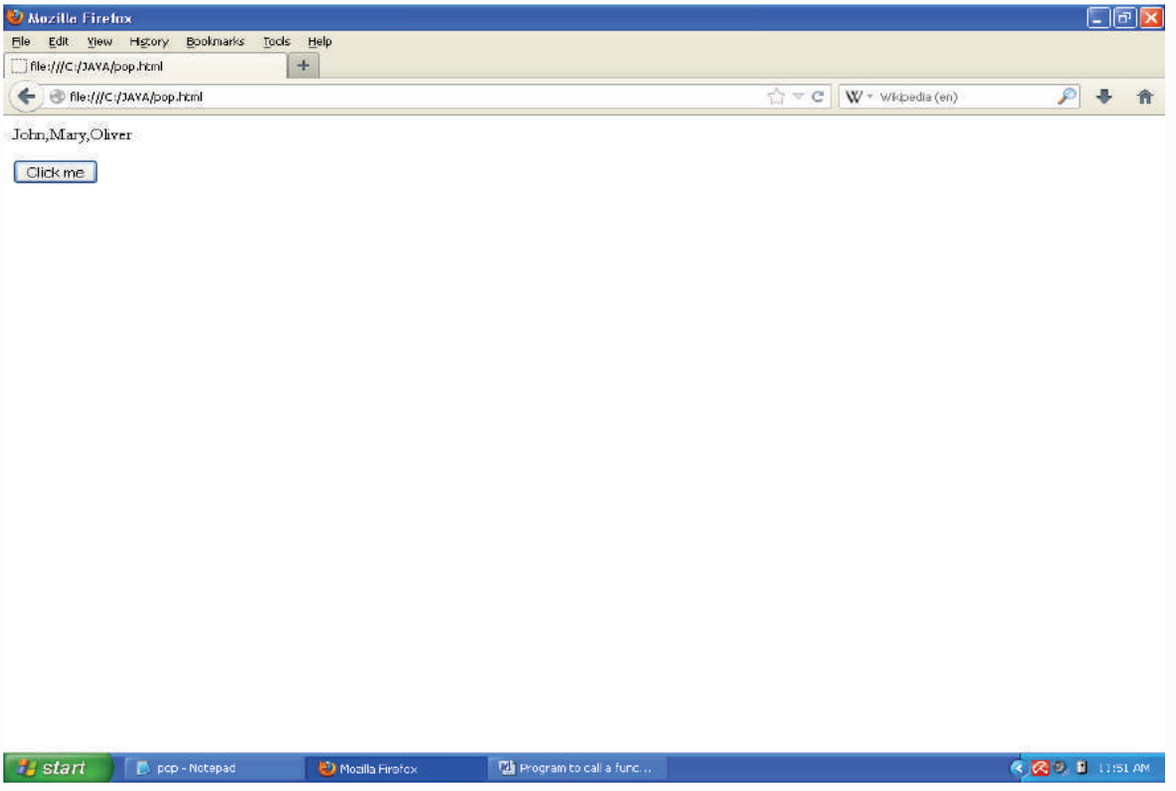

Figure – 3.30

```
//Program to add a new element to the array using push().
<html>
<body>
<p id="demo">Click the button to add a new element to the array.</p>
<button onclick="myFunction()">Click me</button>
<script>
var name=["Alice","Henry","John","Leo"];
function myFunction()
{
name.push("Aayush");
var x=document.getElementById("demo");
x.innerHTML=name;
}
</script>
</body>
</html>
```

Output:

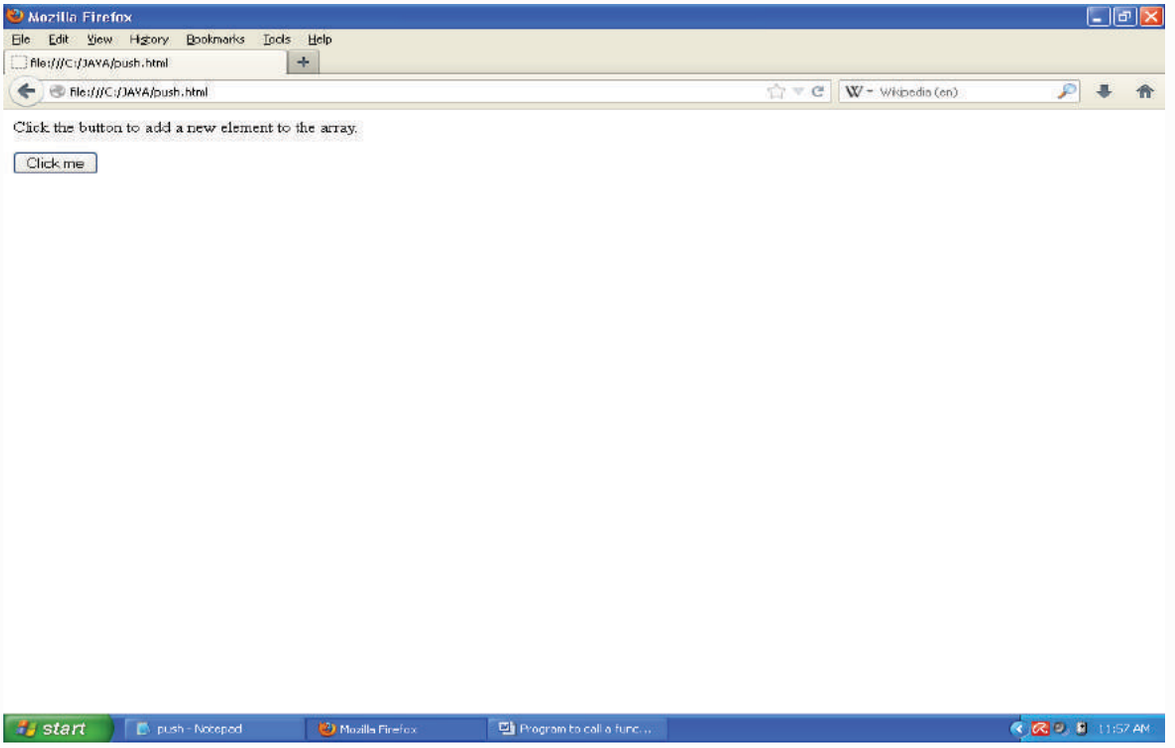

Figure – 3.31

Output continues on clicking the button……..

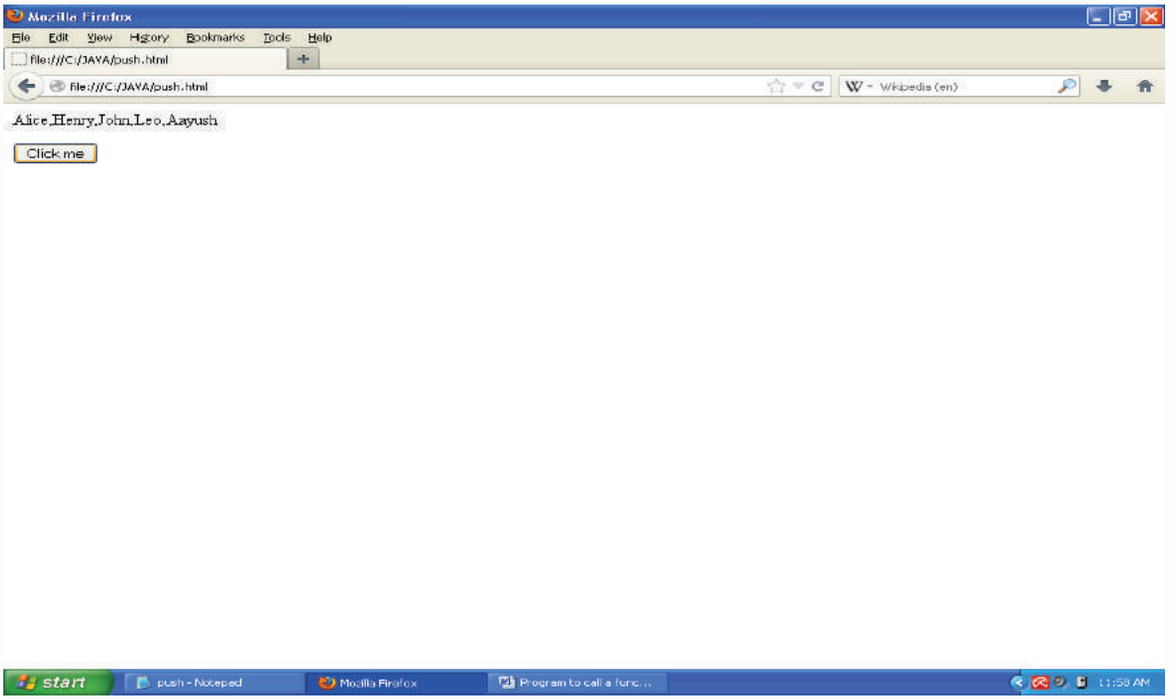

Figure – 3.32

```
//Program to reverse the order of the elements in the array.
<html>
<body>
<p id="demo">Click the button to reverse the order of the element in the array.</p>
<button onclick="myFunction()">Click me</button>
<script>
var alphabet=["z","k","j","h","e"];
function myFunction()
{
alphabet.reverse();
var x=document.getElementById("demo");
x.innerHTML=alphabet;
}
</script>
</body>
</html>
```

Output:

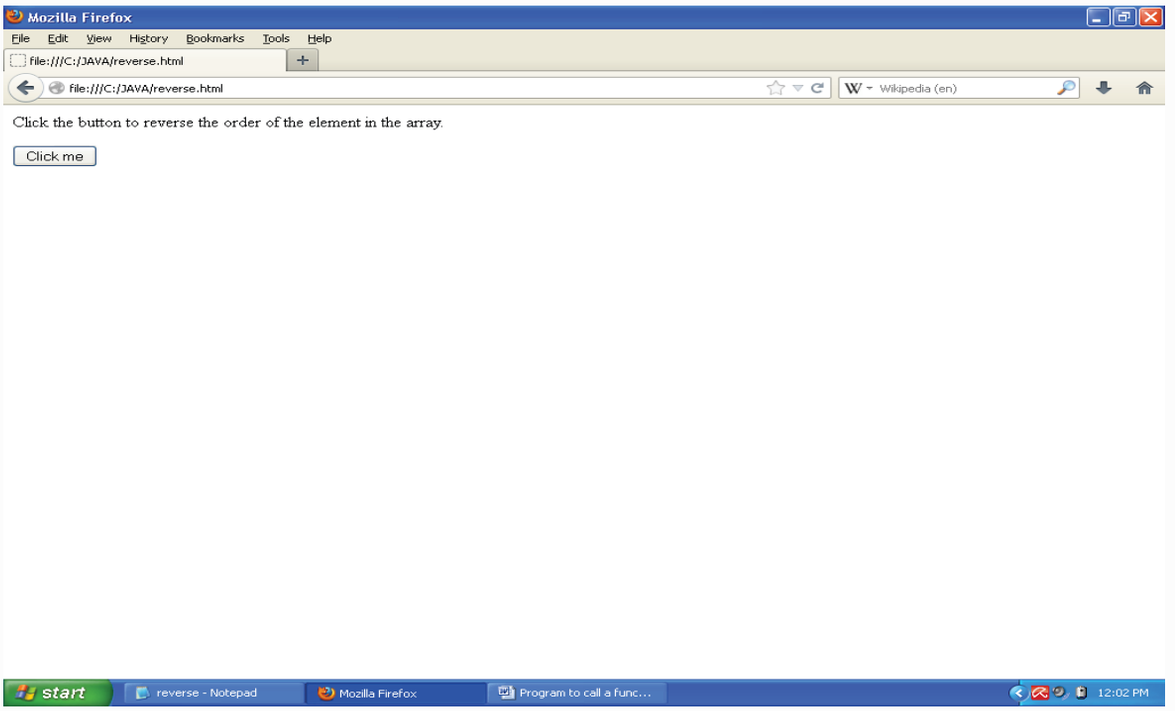

Figure – 3.33

Output continues……

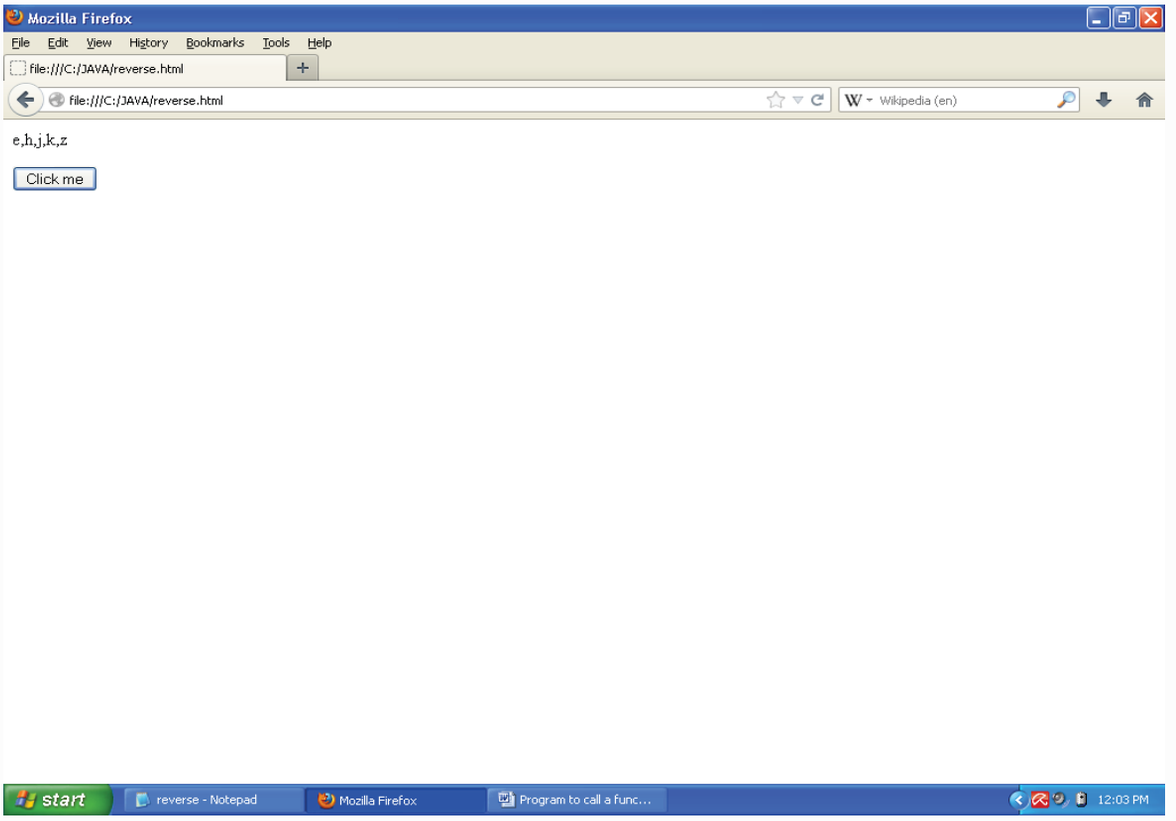

Figure – 3.34

```
//Program to sort the array.
<html>
<body>
<p id="demo">Click the button to sort the array</p>
<button onclick="myFunction()">Click me</button>
<script>
function myFunction()
{
var fruits=["Banana","Orange","Apple","Mango"];
fruits.sort();
var x=document.getElementById("demo");
x.innerHTML=fruits;
}
</script>
</body>
</html>
```

Output:

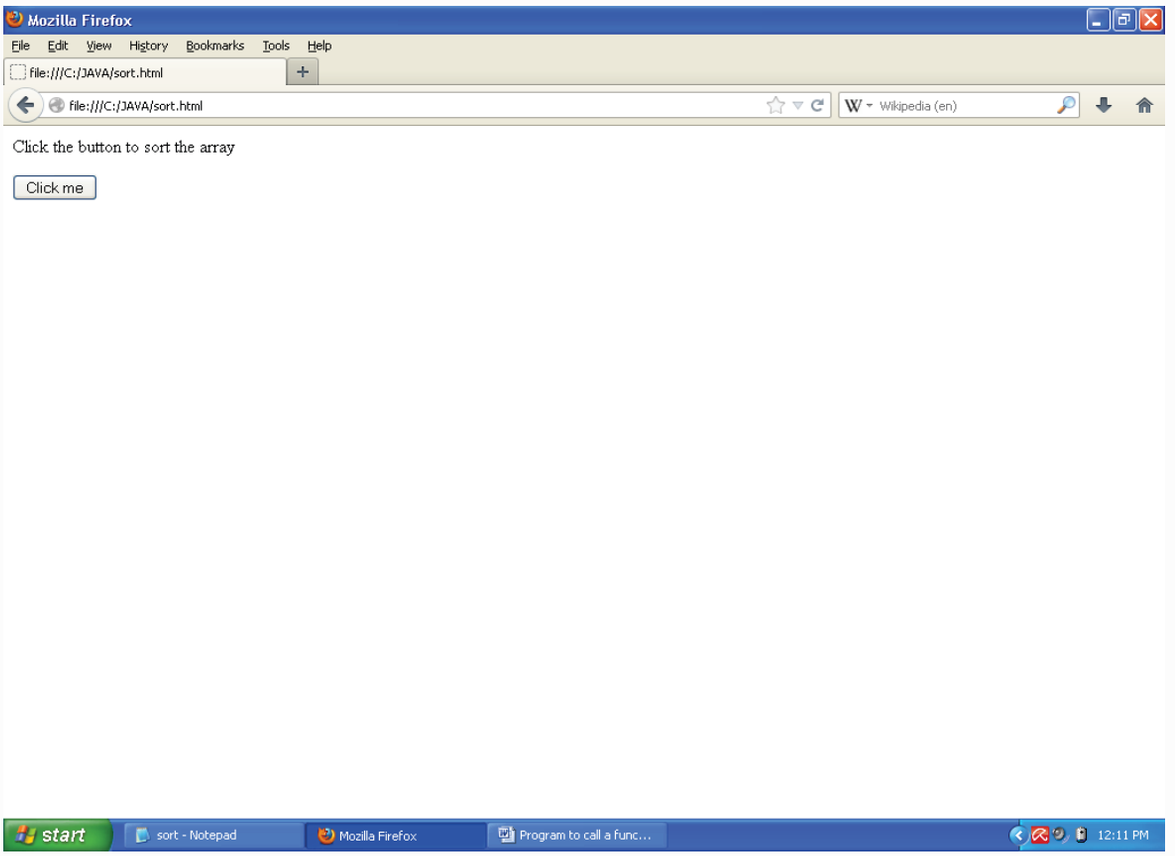

Figure – 3.35

Output continues……

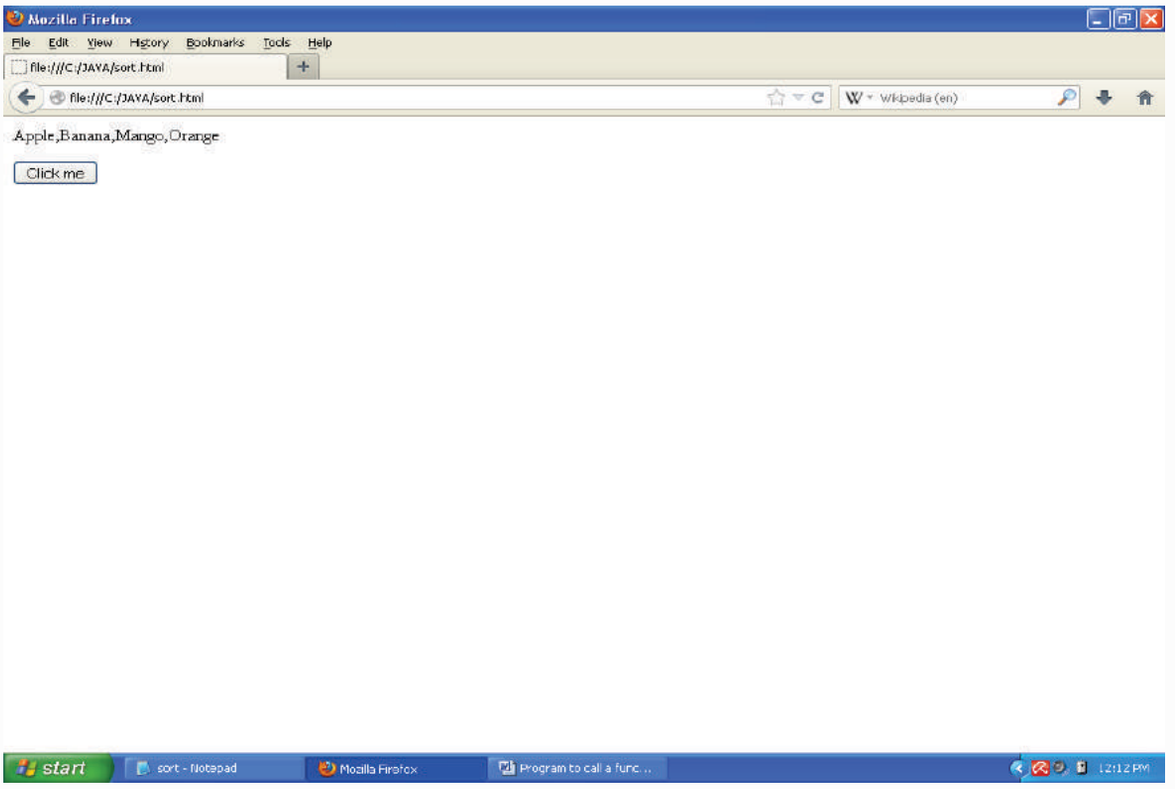

Figure – 3.36

3.7 Events

The objects in Web pages are organized in a hierarchical structure. All objects have properties and methods. In addition, some objects also have "events". Events are things that happen, usually user actions that are associated with an object. The "event handler" is a command that is used to specify actions in response to an event. Below are some of the most commonly used events:

- onLoad - occurs when a page loads in a browser
- onUnload - occurs just before the user exits a page
- onMouseOver - occurs when you point to an object
- onMouseOut - occurs when you point away from an object
- onSubmit - occurs when you submit a form
- onClick - occurs when an object is clicked

```
//Execution of javascript immediately after a page has been loaded.
<html>
<head>
<script>
```

```
function myFunction()
{
confirm("Welcome to the loaded browser");
}
</script>
</head>
<body onload="myFunction()">
<h1>Event handling!</h1>
</body>
</html>
```

Output:

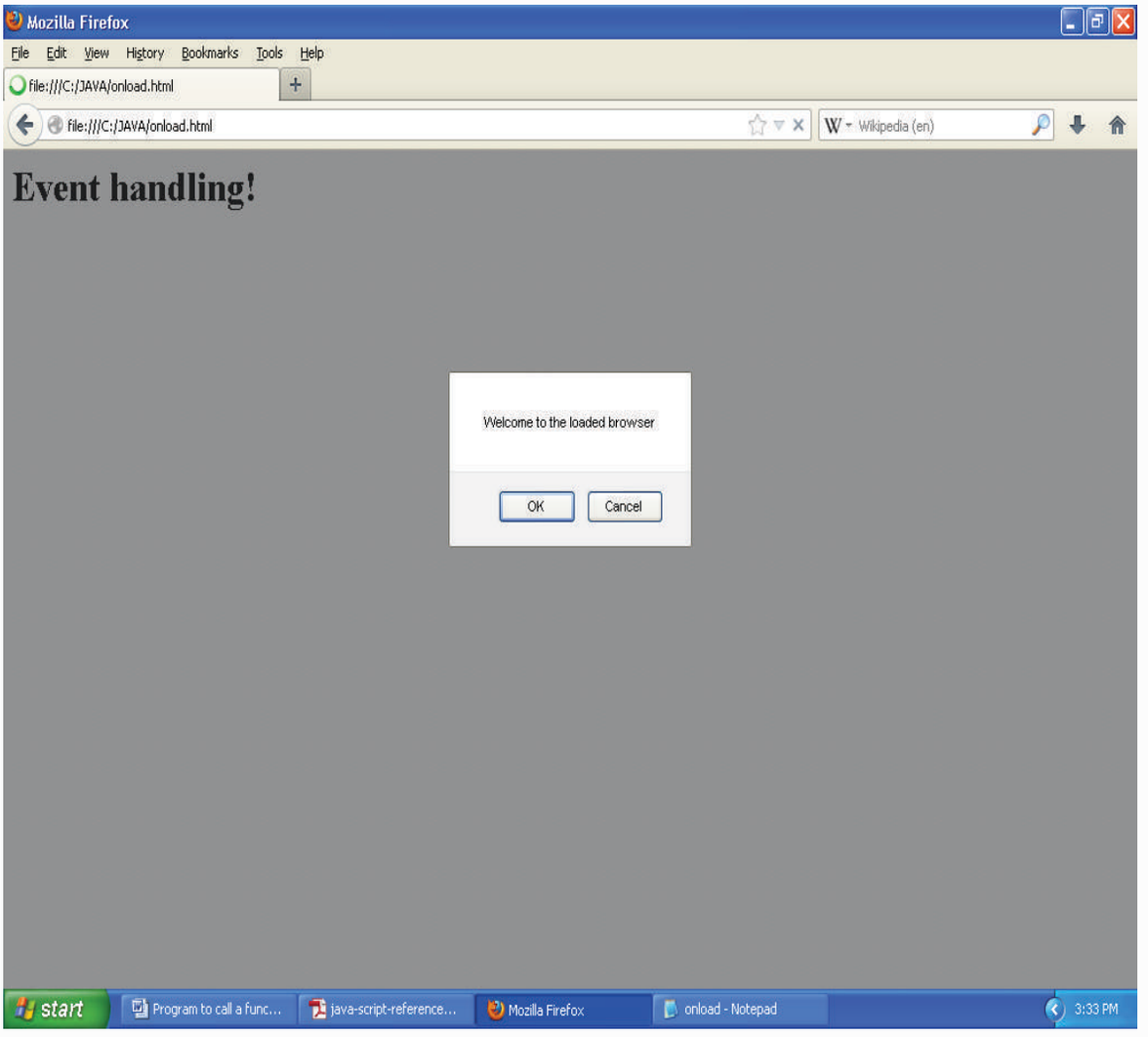

Figure – 3.37

```
//Execute a javascript when a button is clicked.
<html>
<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML="Hello World";
}
</script>
</head>
<body>
<p>Click the button</p>
<button onclick="myFunction()">Click me</button>
<p id="demo"></p>
</body>
</html>
```

Output:

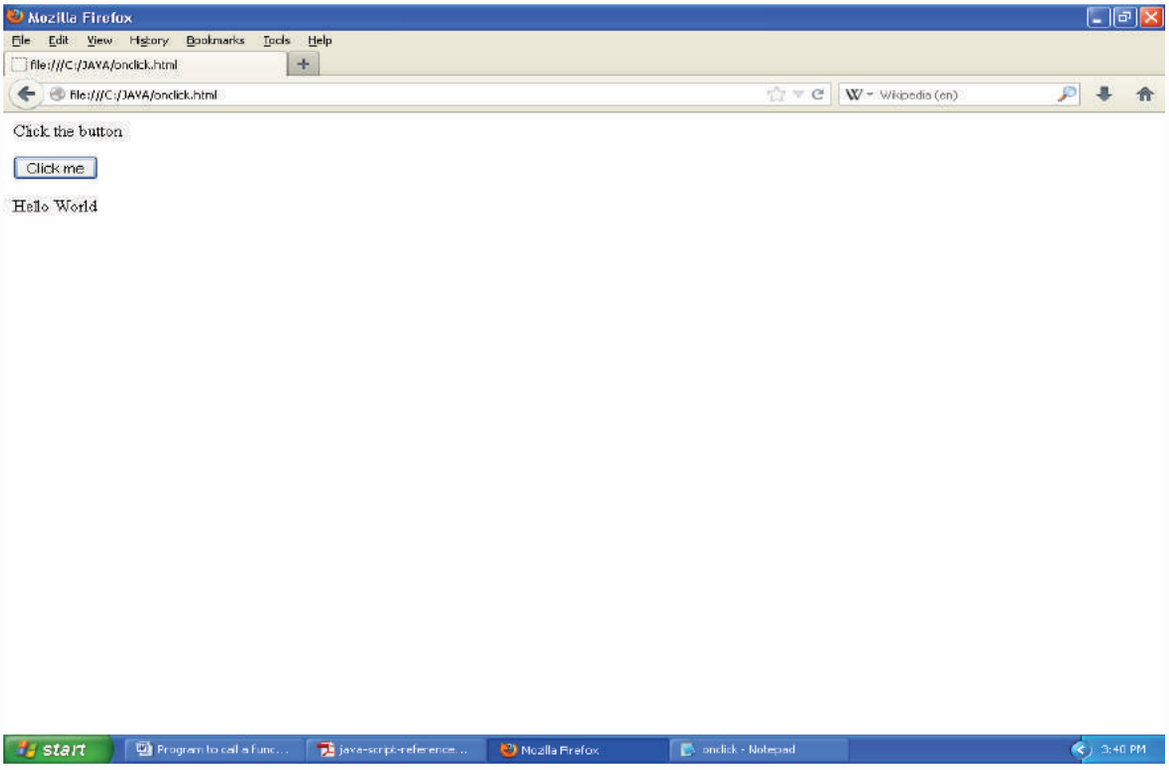

Figure – 3.38

3.8 Case Studies

Case Study I: Online Shopping

In this section we consider the scenario of online shopping web-site. Online shopping deals with purchasing products or services over the Internet any time anywhere. Online shopping or e-shopping is a form of electronic business which allows consumers to directly buy goods or services from a seller over the Internet using a web browser. Alternative names are: e-web-store, e-shop, e-store, Internet shop, web-shop, web-store, online store, online storefront and virtual store. Online shopping has grown in popularity over the years, mainly because people find it convenient and easy to bargain shop from the comfort of their home or office. The students can use the concept of Java Script for designing a commercial web-site. Students may design the online web-site with the following specifications:

- Create a home page using java scripts specifying login page and products details. The home page includes details of the user account with products details and description of items and their availability with price tags and discounts if any (for some specific offer).
- Either user can login or register to the websites as per the choice of shoppers/users.
- For Registration page javascript will be used – which will check for user validation such as having a valid e-mail_id to check the authenticity of the user.
- After login page – link the other pages with home page which will display about products and transaction details.
- A fresh session starts for each user after login for some specific time which will expires once you log out or leave the web page idle for some time.
- User can select product for purchasing from product list provided on the home page.
- After that the user will be redirected to payment page / online payment.
- Payment can be done either using cash on delivery or e-payment (online transaction through some payment gateway i.e., Online bill pay is an electronic payment service that allows you to set up a secure online account to make one time or recurring bill payments for online transactions)
- Generate a receipt for each transaction or combined transactions with payment details and item details to be dispatch to the user which can be used in case of transaction failure or if items not received with desired descriptions as per the user's requirements.
- Host the web-site.

Case Study – II: Designing a Web-site for your School

One of the most important steps involved in creating a school web site is deciding what content will be included and how it will be organized. Building a school web site provides a unique addition to the curriculum. In the development phase, students and teachers should discuss the kinds of materials they want to add to the site. Students may consider the following issues while designing web-site for your own school.

- **School Information:** General information such as the school's telephone number and address along with driving directions should be included. Making mention of the school boundaries is also useful. The school's history, mission statement, and awards received are interesting and serve to promote a feeling of community. The names of the administration and faculty along with their school email addresses may also be added. Other information about the office hours, the bell and bus schedules, and after-school care may also be included.

- **School Policy:** The school policy is a very important document that may be made readily available by posting it on the school site. School policy information might include use of the Web in school, admission procedures / requirements, dress code, absences, and behaviour expectations. Publishing school policies on your site can help increase awareness and access to this information.

- **Calendar of Events:** A calendar of events provides parents, teachers and students with an easy way to keep up-to-date with the happenings at the school. Special events such as school plays, sports, field trips, and standardized tests are just a few examples of what may be included. The school lunch menu may be added to the calendar as well. Calendars are easily updated each month and this method of showing events cuts down on the amount of paper and printing that needs to be done. In addition, parents can always have access to the most recent calendar without worrying about it becoming lost.

- **Extracurricular Activities:** School clubs or organizations involved in organizing special events may create their own pages describing their purposes and anticipated activities for the entire year. Meeting and event schedules with detailed information are useful for informing students and gaining their interest.

- **Newsletters:** Newsletters written by the students are another great way to relay information about the school. Students may submit articles, reports on class trips, and special school events. Involving students gives them a chance to share their thoughts in writing and build school spirit. A message from the principal that provides motivation and encouragement to students and promoters parents involvement may be included.

- **Parent Involvement:** An excellent way to encourage parent involvement is to keep them informed of opportunities such as volunteering, PTA meetings, and fundraising

activities. By providing a wide range of choices, the school can help parents find the activities that fit their time and schedule constraints as well as their interests.

- **Links:** There are many excellent resources available on the Web for teachers, students and parents. Providing links to some of these sources is very helpful. Be sure to check the links periodically to make sure the resources are still available. Teachers can have easy access to learning communities and lesson plan ideas. Students can link to reference or ask-an-expert sites and sites with virtual museums. Links of interest to parents can also be included.

Exercise:

Q1. Define the characteristics of JavaScript? Is java and java script both are similar? Write the difference between both of them.

Q.2. Point the errors in the following code and write the corrected script:

```
<SCRIPT LANG="JavaScript">
var A=10;
var B=6;
switch A%B
{
case 0;
document.write("The input is:");
break;
case 1;
document.write("The colour is:")
break;
else;
document.write("You didn't choose")
}
</SCRIPT>
```

Q3. Give the output of the following code segment:

```
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var sum = new Array(6);
var Total = 0;
sum[0] = 0;
```

```
for(var icount = 1; icount < 4; icount++)
{
Total += icount;
sum[icount] = Total;
document.write(sum[icount]+"<br>");
}
document.write(Total);
</SCRIPT>
</BODY>
```

Q4 (a) Explain the functions and arrays in javascript with example?

(b) How the forms are handled in javascript? Explain.

Q5. How can javascript make a website easier to use? That is, are there certain javascript techniques that make it easier for people to use a website?

Q6. How do you convert numbers between different bases in javascript?

Q7. How do we get javascript on a web page explain?

Q.8 Write a program in javascript using various data types to find the round off integer values?

Q9. Write the program in javascript to solve the given polynomial:

$$x^3 + 2x^2.y + 3x.y^2 + y^3 = 0$$

Q10. Design a home page of your school. Include some images/pictures and other events of an academic calendar for session 2013-2014. Also include some audio/video features to make your web-site more interactive.