Compiled By: Er. Ranjan Raj Aryal
Add. Associate Professor
Cosmos College of Management & Technology

JAVAScript

Introduction

- JavaScript can be divided into three parts : the core, client side and server side.
- The **core** is the heart of the language, including its operators, expressions, statements and subprograms.
- **Client-side** JavaScript is a collection of objects that support control of the browser and interactions with users. Eg. with JS, an HTML document can be made to be responsive to user inputs such as mouse clicks and keyboard use.
- **Server-Side** JS is a collection of objects that make the language useful on a Web server. For example, to support communication with a database management system.

Origins of JavaScript

- JavaScript was originally named LiveScript, was developed by Netscape.
- In late 1995, LiveScript became a joint venture of Netscape and Sun Microsystems and its name was changed to JavaScript.
- A language standard for JavaScript was developed in the late 1990's by the European Computer Manufacturers Association (ECMA) as ECMA-262
- This standard has also been approved by the International Standards Organization (ISO) as ISO-16262.
- Microsoft's JavaScript is named Jscript
- Client Side JavaScript is an HTML-embedded scripting language.
- Every collection of JavaScript code is referred as a script.

JavaScript and JAVA

- Although JavaScript's name appear to connote (imply) a close relationship with Java, JavaScript and Java are actually very different.
- One important difference is support for the Object-oriented Programming.
- Although JavaScript is sometimes said to be an Object-Oriented Language, its object model is quite different from that of Java and C++.
- In fact, JS does not support the object-oriented software development paradigm, Java on the other hand, provides complete support for object-oriented programming.
- Java is completely typed language. Types are all known at compile time, and operand types are checked for compatibility. Variables in JS need not be declared and are dynamically typed, i.e making compile -time type checking impossible.
- One more difference between Java and JavaScript is that, objects in Java are static in the sense that their collection of data member methods is fixed at compile time.
- JavaScript objects are dynamic-the number of data members and methods change during execution.

The main similarity between Java and JavaScript is the syntax of the expression, assignment statements and control statements.

Uses of JavaScript

- The original goal of JavaScript was to provide programming capability at both the server and the client ends of a Web connections.
- Client-Side JavaScript can serve as an alternative for some of what is denoted with server-side programming, in which computational capability resides on the server and is requested by the client.
- Client-Side JS, on the other hand, is embedded in HTML documents and is interpreted by the browser. This transfer of load from the often overloaded server to the normally under loaded client can obviously benefit all other clients.
- Client-side JS, cannot replace all of the server-side computing. In particular, while server side software supports file operation, database access and networking, client-side JavaScript supports none of these.
- JS can be used as an alternative to Java applets. JS has the advantages of being easier to learn and use than Java. Also Java applets are downloaded separately from the HTML documents that call them; JS however, are an integral part of the HTML document, so no secondary downloading is necessary. On the other hand , Java applets are far more capable of producing graphics in documents than are JS scripts.
- Interactions with users through form elements, such as buttons and menus are easy to describe in JavaScript. Because events such as button click, are easily detected with JS, they can be trigger computations and provide feedback to the user. For example, the appropriateness of form values can be checked on the client with JavaScript before sending those values to the server for processing.
- Most exciting capability of JS has been made possible by the development of the Document Object Model (DOM), which allows JS scripts to access and modify the controls and content of any element of a displayed HTML document, making formally static documents highly dynamic.

JavaScript Objects

- In JS, objects are collections of properties, which correspond to the members of classes in Java and C++. Each Property is either a data property or a method property.
- Data properties appear in two categories: primitive values and references to other objects.
- In JS, variables that refer to objects are often called objects, rather than references.
- Data properties and method properties are often called properties and methods respectively.
- Like many languages that support object-oriented programming, JS uses non object types for some of its simplest data types-these types are called primitives.
- Primitives are used because they often can be implemented directly in hardware, resulting in faster operations on their values(faster than if they were treated as objects).
- Primitives are like the simple scalar variables of non-object-oriented languages, such as C.
- C++, Java and JS all have both primitives and objects.

- All objects in JS program are indirectly accessed through variables. Such a variables is like a reference in Java.
- All primitive values in JS are accessed directly-these are like scalar type in Java and C++.
- The properties of an object are referenced by attaching the name of the property to the variable that references the object. Eg. In "myCar.engine" myCar is a variable that is referencing an object and engine is the property.
- A JS object appears, both internally and externally as a list of property/value pairs.
- The properties are names; the values are data values or functions. All functions are objects and are referenced through variables.
- The collection of a JS object is dynamic i.e properties can be added or deleted at any time.

General Syntactic Characteristics

- All JS scripts are embedded , either directly or indirectly.
- In HTML documents, scripts can appear directly as the content of a <script> tag.
- The JS script can be indirectly embedded in an HTML document using the src attribute of a <script> tag.
  Eg.

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

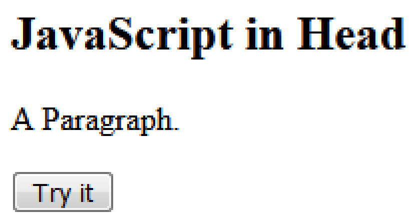
- The indirect method of embedding JS in HTML document has the advantage of hiding the script from the browser user.
- Identifiers or names in JS are similar to those o other common programming languages. They must begin with a letter, an underscore(_) or a dollar sign ($).
- There is no length limitation for identifiers
- Letters in the variables are case sensitive.

Example of Internal JS

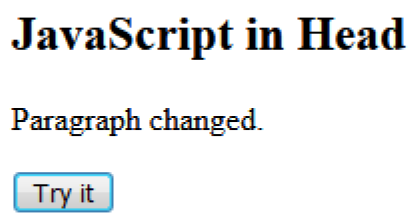
```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h2>JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>
```

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

</body>
</html>
```

Output


JavaScript in Head

A Paragraph.


JavaScript in Head

Paragraph changed.

<u>Example of External JS</u>

```
<html>
<head>
<script src="myScript.js"></script>
</head>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

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

</body>
</html>
```

<u>Reserved words of JavaScript</u>

break	delete	function		return	typeof	
case	do	if	switch	var		
catch	else	in	this	void		
continue		finally	instanceof		throw	while
default		for	new	try	with	

<u>Primitives, Operations and Expressions</u>

Primitive Types

- JavaScript has five primitive types: Number, String, Boolean, Undefined, Null.
- All primitive values have one of these types.

- JS defined objects are closely related to the Number, String and Boolean types, named Number, String and Boolean.
- These three objects are wrapper objects. Each contains a property that stores a value of the corresponding primitive type. Their purpose is to provide properties and methods that are convenient for use with values of the primitive types.
- In the case of number, the properties are more useful; in case string , the methods are more useful. Because JS coerces values between the Number type and Number objects and between the String type and String objects, the methods of Number and String can be used on variables of the corresponding primitive types.
- In fact, in most cases you can simply treat Number and String type values as if they were objects.

  The differences between primitive and objects is illustrated in the following example. Suppose prim is a primitive variable with the value 17 and obj is a Number object whose property value is 17. Following figure shows how prim and obj are stored.

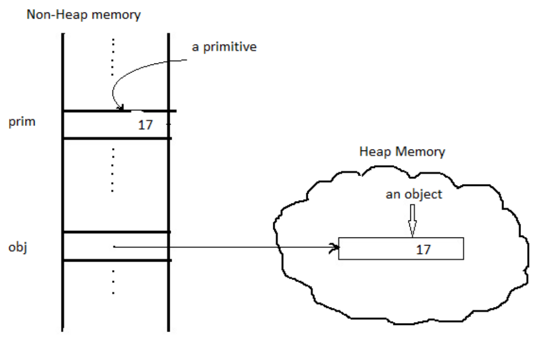


Fig: Primitives and objects

Declaring Variables

- Variables can have value of an primitive type, or it can be a reference to any object.
- The type of a particular appearance of a variable in a program is determined by the interpreter.
- Initial values can be included in "var" declaration
- Example:

```
var counter,
        index,
        pi = 3.14159,
        stop_flag =  true;
```

Numeric Operators

+, - , * , /, %, ++, --

- Increment and decrement operator can be either prefix or postfix
  Eg;
  If a = 7, then (++a) * 3 is equal to 24
  If a = 7, then (a++) * 3 is equal to 21
- All numeric operations are done in double- precision floating point.

Precedence and Associativity of the numeric operators

Operators	Associativity
++, - -, -	Right
*, /, %	Left
+, -	Left

Eg:

```
Var a = 2,
        b = 4
        c,
        d;
        c = 3 + a * b; //here c = 11 not 24
        d =    b / a /2; //associates left, so d is now 1 not 4
```

The String Concatenation Operator

- JS string are not treated as arrays of characters; rather they are unit scalar values.
- String concatenation is specified with the operator denoted by a plus sign (+) .For example, if the first value is "Cosmos", the value of the following expression is "Cosmos College".

```
        First + "College"
```

Implicit Type Conversions

- JS interpreter performs several different implicit type conversions. Such conversions are called "coercions".
- In general, when a value of one type is used in a situation that requires a value of different type, JS attempts to convert the value to the type is required.
- The most common examples of these conversions involve primitive string and number values.

  Eg; "January" + 2019

  In this example, because left operand is a string, the operator is considered as concatenation operator. This forces string context on the right operand, so the right operand is implicitly converted to a string. Thus the expression becomes:

  "January 2019"

- If either operand of the catenation operator is a string, the other operand is converted to string. Thus in following expression , the first operand is converted to string:
  2019 + " January"
- Consider another example,
  7 * "3"
  In this expression, the operator is one that is only used with numbers. This forces numeric context on the right operand. JS therefore attempt to convert it to a number. Thus the value in above expression will be 21.
  Here , if the second operand is string i.e. 7 * "January", that could not be converted to a number, this type of conversion would produce NaN.

String Properties and Methods

- The string object and string type in JS have little effect on scripts because JS coerces primitive values to and from string objects when necessary.
- String methods can always be used through String primitive values, as if the values were objects
- String object includes one property, length and a large collection of methods.
  Eg: var str = "Cosmos";
  var len = str.length;
  In this code len is set to the number of characters in str 6. Here str is a primitive variable but we have treated it as an object
- Note: when str is used with the length property, JS implicitly builds a temporary string object with a property whose value is that of the primitive variable.
- Eg; var str = "cosmos";
  The following expressions have shown the values :

  str.charAt(2) is 's'
  str.indexOf('m') is 3
  str.substring(2,4) is 'smo'
  str.toLowercase( ) is 'cosmos'

String Methods

Methods	**Parameters**	**Result**
charAt	A number	The character in the string object is at the specified position
indexOf	One-character string	The position in the string object parameter
Substring	Two numbers	The substring of the string object of the first parameter position to the second
toLowerCase	None	Converts any uppercase letters in string to lowercase
toUpperCase	None	Converts any lowercase letters in an string to uppercase.

Assignment Statements

Eg: a += 5;
means the same as this one:
a = a + 7;

Screen Output

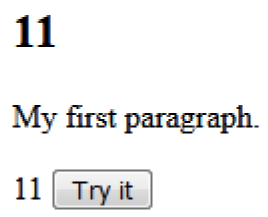
- The document object has several properties and methods. The most interesting and useful methods, is "write".
- Example:
  document.write(" Hello World!!!");
- The **"alert"** method opens a dialog window and displays its parameter in that window. It also displays a button labeled OK. The parameter string to alert is not a HTML code; it is a plain text. Thus the string parameter to alert may include \n but never <br/>
  Eg; alert("Hello!!! Are you sure what you are doing ? \n");
- **"confirm"** method opens a dialog window in which it displays its string parameter, along with two buttons, OK and Cancel.
- "confirm" returns a Boolean value that indicates the user's button input: true for OK and False for Cancel.
- "confirm" method is often used to offer the user the choice of continuing some process.
  Eg; var question = confirm (" Do you want to continue ?");
- **"prompt"** method creates a dialog window that contains a text box. The text box is used to collect a string of input from the user, which prompt returns as its value. The window also includes two buttons, OK and Cancel.
- "prompt" takes two parameters: the string that prompts the user for input and default string in case the user does not type a string.
  Eg: name = prompt(" What is your name ?, " ");

Example of write

<!--
Writing into an HTML element, using innerHTML.

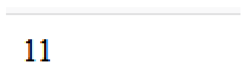
```
Writing into the HTML output using document.write().
Writing into an alert box, using window.alert().
Writing into the browser console, using console.log().
        -->
<!DOCTYPE html>
<html>
<body>

<h2 id="demo1">My First Web Page</h2>
<p id="demo">My first paragraph.</p>

<script>
document.write(5 + 6);
document.getElementById("demo1").innerHTML = 5 + 6;
console.log(5 + 16);
</script>
<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Output



```
11

My first paragraph.

11
```

After pressing **"Try it"** Button:

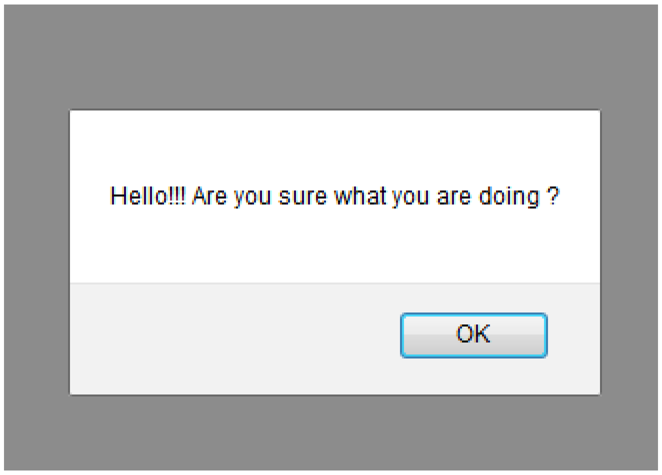


```
11
```

<u>Example of "alert"</u>

```
<html>
<body>
<script>
alert("Hello!!! Are you sure what you are doing ?\n");
</script>
</body>
</html>
```

Output

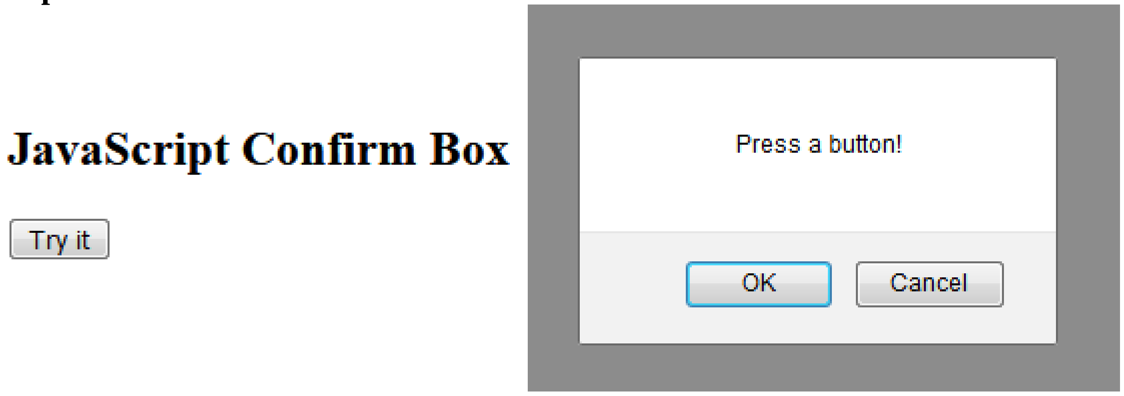


```
Hello!!! Are you sure what you are doing ?
```

<u>Example of "confirm"</u>

```
<!DOCTYPE html>
<html>
<body>
        <h2>JavaScript Confirm Box</h2>
        <button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var txt;
    if (confirm("Press a button!")) {
        txt = "You pressed OK!";
    } else {
        txt = "You pressed Cancel!";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

Output

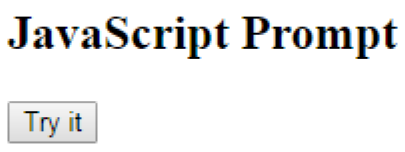


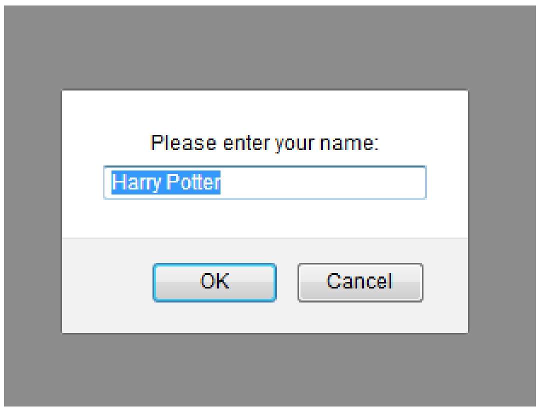
JavaScript Confirm Box

```
Press a button!
```

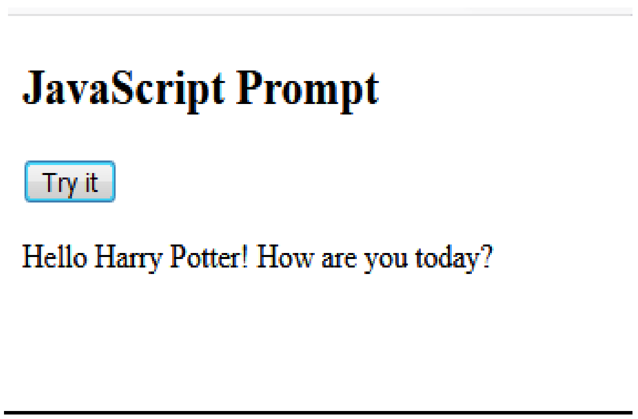
Example of “prompt”

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Prompt</h2>

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

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

<script>
function myFunction() {
    var txt;
    var person = prompt("Please enter your name:", "Harry Potter");
    if (person == null || person == "") {
        txt = "User cancelled the prompt.";
    } else {
        txt = "Hello " + person + "! How are you today?";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```




Please enter your name:

Harry Potter



JavaScript Prompt

Hello Harry Potter! How are you today?

Control Statements

- Also called flow control statements.
- Control Statements often require some syntactic container for sequences of statements whose execution they are meant to control
- In JS, that container is the compound statement.
- A compound statement in JS is a sequence of statements delimited by braces
- A control construct is a control statement(s) whose execution it controls
- JS   does not allow compound statement to create local variables.
- If a variable is declared in a compound statement, access to it is not confined to that compound statement.

Control Expressions

- The result of evaluating a control expression is one of the Boolean values true or false
- If the value of a control expression is s string, it is interpreted as true unless it is either empty string or a zero string(“0”).
- If the value is number, it is true unless it is zero (0).

Relational Operators

= =, ! =, <, >, < =, > =, = = = (is strictly equal to), ! = = (is strictly not equal to)

Eg: “3” = = = 3 evaluates to false while,

“3” = = 3 evaluates to true. Because the last two operators (= = = , ! = =) disallow type conversion of either operand.

Selection Statements

- The selection statement : if-then, if-then-else of JS are similar to those of the common programming languages.
- Either single statements or compound statements can be selected.

Eg:

```
if (a > b)
        document.write(" a is greater than b <br/ >" );
else {
        b = a;
        document.write(" a was not  greater than b <br/ >" );

    }
```

Switch Statement

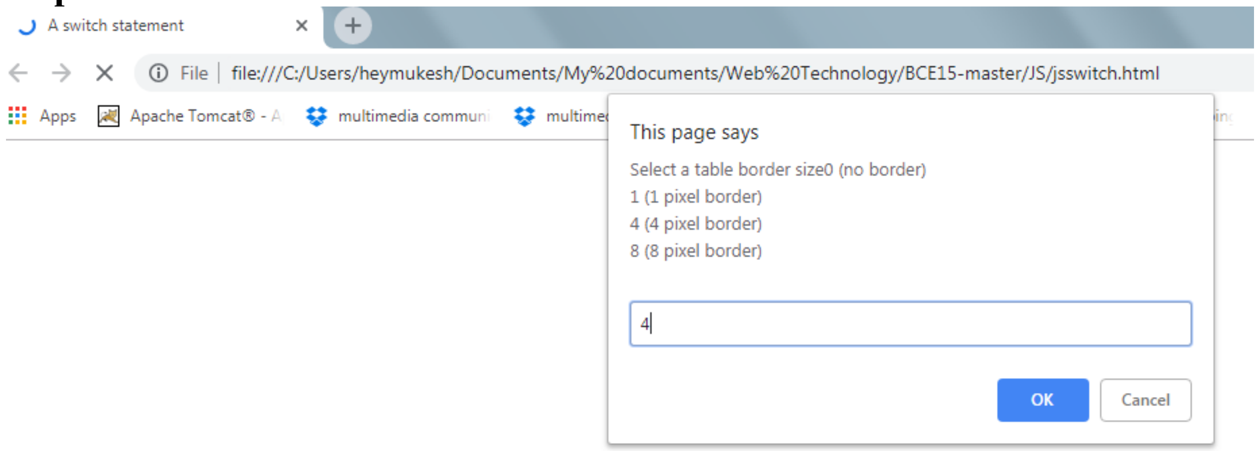
- JS has a switch statement that is similar to that of C

```
switch ( expression ) {
                case value_1:
//statement
case value_2:
//statement
....
default:
//default statement

}
```

Example

```
<!DOCTYPE html>
<html>
<head>
        <title> A switch statement</title>
</head>
<body>
        <script type="text/javascript">
                var bordersize;
                bordersize = prompt ("Select a table border size" +
                                        "0 (no border) \n" +
                                        "1 (1 pixel border) \n" +
                                        "4 (4 pixel border) \n" +
                                        "8 (8 pixel border) \n" );

                switch (bordersize) {
                        case "0" : document.write("<table>");
                        break;
                        case "1" :document.write("<table border =1>");
                        break;
                        case "4":document.write("<table border = 4>");
                        break;
                case "8":document.write("<table border = 8>");
                break;
                default: document.write("Error-invalid choice, <br />");

        }
        document.write("<caption>Nagarpalika Gaupalika details</caption>");
        document.write("<tr>",
                                "<th/>",
                                "<th>No.of Gaupalika</th>",
                                "<th>No.of Nagarpalika</th>",
                    "<tr/>",
                    "<tr>",
                                "<th>Pradesh 1</th>",
                                "<td>Ten</td>",
                                "<td>Seven</td>",
                        "</tr>",
                        "<tr>",
                                "<th>Pradesh 2</th>",
                                "<td>Five</td>",
                                "<td>Nine</td>",
                        "</tr>",
                                "<th>Pradesh 3</th>",
                                "<td>Eleven</td>",
                                "<td>Eight</td>",

                        "</tr>",
                        "</table>"
                    );
        </script>
</body>
</html>
```

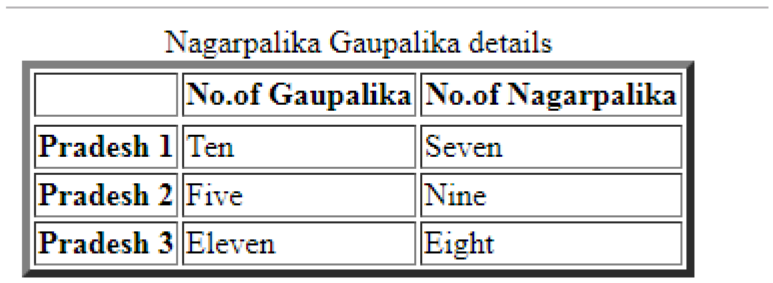
output


A switch statement
File
file:///C:/Users/heymukesh/Documents/My%20documents/Web%20Technology/BCE15-master/JS/jsswitch.html
Apps
Apache Tomcat® - A
multimedia communi
multimed
OK
Cancel

This page says

Select a table border size0 (no border)
1 (1 pixel border)
4 (4 pixel border)
8 (8 pixel border)

4


Nagarpalika Gaupalika details

	No.of Gaupalika	No.of Nagarpalika
Pradesh 1	Ten	Seven
Pradesh 2	Five	Nine
Pradesh 3	Eleven	Eight

Loop Statement

- The JS "while" and "for" statements are similar to those of JAVA and C++.
  Eg:

```
var sum = 0;
        count;
        for (count =0; count <=10; count++)
                sum += count;



do{
        count++;
        sum = sum + (sum * count);
}while count <=50;
```

Object Creation and Modification

- Objects are often created with a new expression, which must include a call to a constructor method.
- The constructor that is called in a new expression creates the properties that characterize the new object.
- In JS, however, the new operator creates a blank object, or one with no properties.
- JS objects do not have types.
  Eg:
  `var my_object = new Object( );`
  Here variable my_object references the new object.

Arrays

- Arrays in JS are objects that have some special functionality.
- Arrays values can be primitive values or references to other objects, including other arrays.

- JS Arrays have dynamic length.

Array Object Creation

- Array objects, unlike most other JS objects, can be created in two distinct ways.
- Usual way is with new operator and a call to a constructor.
- In case of arrays, the constructor is named Array.
```
var my_list = new Array( 1, 2, "three ", "four" );
var your_list = new Array (100);
```
- In the first declaration, a new Array object of length 4 is created and initialized.
- In the second, new Array object of length 100 is created, without actually creating any elements. Because, whenever a call to the Array constructor has a single parameter, that parameter is taken to be the number of elements not the initial value of a one-element array.
- Every Array object is given the length property by the Array constructor.
- Next way to create an Array object is with a literal array value, which is a list of values enclosed in brackets:
```
var my_list_2 = [1, 2, "three" , "four" ];
```
- The array my_list_2 has the same values as the Array object my_list created as above.

Characteristic of an Array Objects

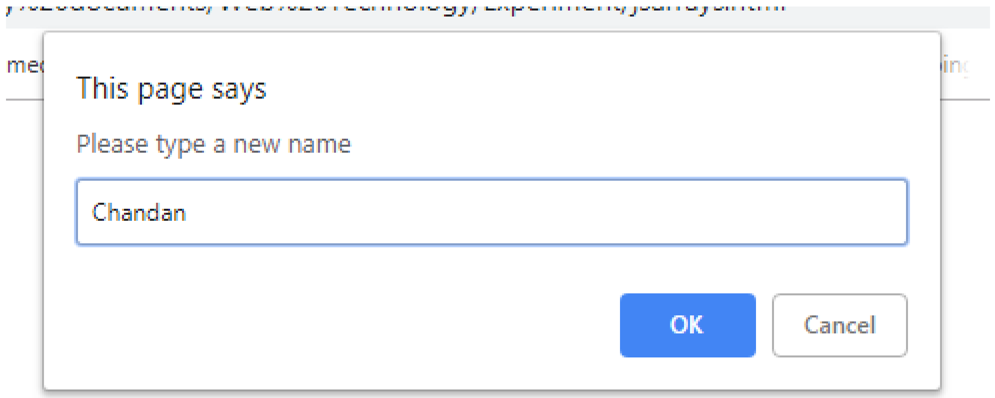
- The lowest index of every JS array is zero.
- Array element access is specified with numeric subscript placed in the brackets.
- The length will be assigned value plus 1.
- The length of an array can be set whatever you like by assigning the length property:
```
my_list.length = 1000;
```

Example program of Array

<!--program to read name using prompt and sort alphabetically in the existing list-->

```
<!DOCTYPE html>
<html>
<head>
	<title>Name List</title>
</head>
<body>
<script type="text/javascript">

var name_list = new Array("Arun", "Barun","Karun",
				"Marun","Ramun","zamun");
var new_name, index, last;
			//loop to get new name and insert it
while(new_name = prompt("Please type a new name","")){
	//loop to find new place for the new names
	last = name_list.length-1;
```

```
          while (last >=0 && name_list[last] > new_name){

          name_list[last + 1] = name_list[last];
          last--;
}//end of inner while loop

name_list[last + 1] = new_name;

document.write("<p><b>The new name list is: </b>", "<br/>");
for (index = 0;index < name_list.length; index++)
          document.write(name_list[index],"<br/>") ;
          document.write("</p>")
}// end of outer while loop
</script>
</body>
</html>
```



```
The new name list is:
Arun
Barun
Chandan
Karun
Marun
Ramun
zamun
```

Array Methods

- Arrays objects have a collection of useful methods
- **join** method converts all of the elements of an array to strings and concatenates them into a single string.
- If no parameter is provided to join , the values in the new string are separated by commas.

Eg:
```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Results

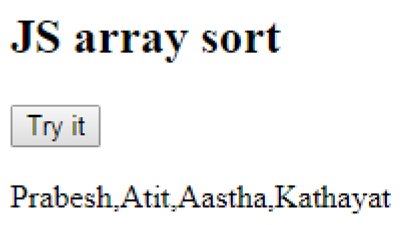
Banana * Orange * Apple * Mango

- **sort** method coerces the elements of the array to strings, if they are not already strings, and sort them into alphabetical order.
  Eg: names.sort( );
- **concat** method catenates its actual parameter to the end of the Array object on which it is called.
  Eg:
  ```
  var new_names = names.concat( "cosmos", "college");
  ```
- **slice** method does foe arrays what the substring method does for the string.
  The returned array has the elements of the array object through which it is called from the first parameter up to but not including the second parameter.
  ```
  Eg;  var list = [2, 4, 6, 8, 10];
           var list2 =  list.slice[1, 3];
  ```
  The value of list2 is now [4, 6]
  If slice has one value then returned array has all the elements of objects, starting with the specified index.
  var list2 = list.slice[2]; will return list2 = [6, 8, 10]
- The push, pop, unshift, and shift methods of array allow the easy implementation of stacks and queues in arrays. The pop and push methods remove and add an element to the end of an array, respectively.
  Eg
  ```
  var list = ["Dinesh", " Dancer", "Donner", "Bold"];
  var d = list.pop( ); //d is now "Bold"
  list.push( "Bold"); //this puts "Bold" back on list.
  ```
- The shift and unshift methods remove and add an element to the beginning of an array respectively.
  Eg:
  ```
  var d = list.shift ( ); // d is now "Dinesh"
  list.unshift(" Dinesh");
  // This puts "Dinesh" back on list
  ```

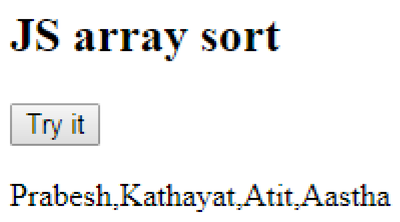
Example: sorting an array

```
<!DOCTYPE html>
<html>
<body>

<h2>JS array sort</h2>

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

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

<script>
var fruits = ["Prabesh", "Atit", "Aastha", "Kathayat"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
   fruits.sort();
         //  reverse it:
   fruits.reverse();
   document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Output



```
Prabesh,Atit,Aastha,Kathayat
```



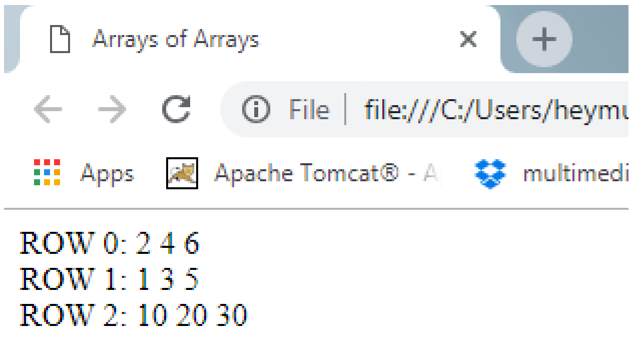
```
Prabesh,Kathayat,Atit,Aastha
```

- A two dimensional array is implemented in JS as an array of arrays. This can be done with the new operator or with nested array literals. The following example illustrates this.

```
<!DOCTYPE html>
<html>
<head>
        <title>Arrays of Arrays</title>
</head>
<body>
```

```
<script type="text/javascript">
        //create an array object with three arrys as its elements
        var nested_array = [[2, 4, 6],
                                        [1, 3, 5],
                                        [10, 20, 30]
                                        ];
//display the elements of nested_list
for(var row=0; row <= 2; row++){
        document.write("ROW ", row, ": ");

        for(var col = 0; col <=2; col++)
                document.write(nested_array[row][col], " ");

        document.write("<br/>");
}
</script>
</body>
</html>
```

Output



```
ROW 0: 2 4 6
ROW 1: 1 3 5
ROW 2: 10 20 30
```

<u>Functions</u>

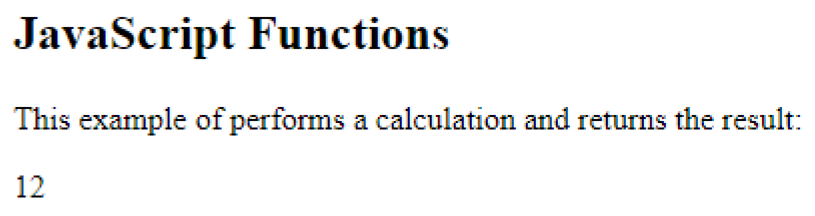
- A function definition consists of the function's header and a compound statement that describes its actions.
- The compound statement is called the body of the function.
- A function header consists of the reserved word 'function' as the function's name, and a parenthesized list of parameters, if there are any.
- A return statement returns control to the function's caller.
- JS functions are objects, so variables that reference them can be treated as other references.

function *functionName*(*parameters*) {
  // *code to be executed*
}

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example of performs a calculation and returns the result:</p>

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

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```

Output



```
JavaScript Functions

This example of performs a calculation and returns the result:

12
```

<u>Local Variables</u>

- The scope of a local variable is in the range of statements over which it is visible.
- When JS is embedded in an HTML document, the scope of a variable is the range of lines of the document over which the variable is visible.
- A variable that is not declared with a var statement is implicitly declared by the JS interpreter at the time it is first encountered in the script, these function definition have global scope.
- Variable declared outside the function definition also have global scope.
- If a variable defined both as a local variable and as a global variable appears in a script, the local variable has precedence, effectively hiding the global variable with the same name. This is the advantages of a local variable. When you make up their names, you need not be concerned that a global variable with the same name may exist somewhere in the collection of scripts in the HTML documents.

Example 1

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";

  // code here CAN use carName

}  // code here can NOT use carName
```

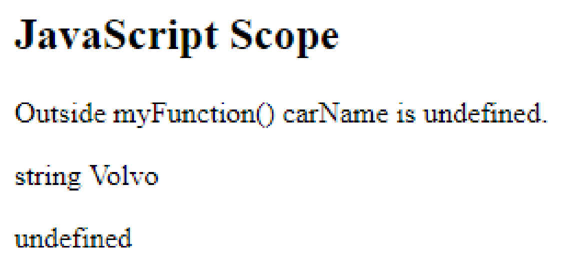
Example 2

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

Example program

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Scope</h2>

<p>Outside myFunction() carName is undefined.</p>

<p id="demo1"></p>

<p id="demo2"></p>

<script>
myFunction();

function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML = typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML = typeof carName;
</script>

</body>
</html>
```

Output



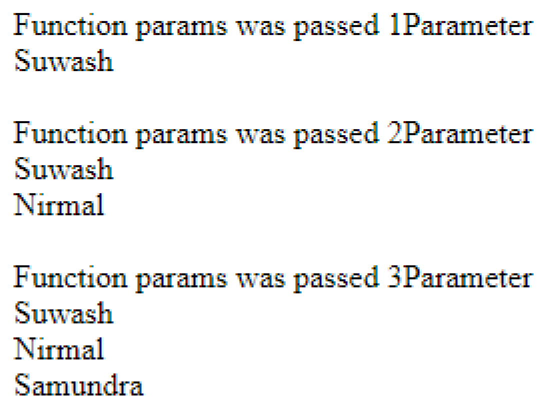
```
JavaScript Scope

Outside myFunction() carName is undefined.

string Volvo

undefined
```

Parameters

- The parameter values that appear in a call to a function are called actual parameter
- Parameter names that appear in the header of a function definition which corresponds to actual parameter are called formal parameters.
- JS uses pass-by-value parameter passing method.
- Because of JavaScript's dynamic typing, there is no type checking parameters.
- The called function itself check the types of parameters with the "typeof" operator. Also, the number of parameters in a function call is not checked against the number of formal parameter in the called function.
  Example

```
<!DOCTYPE html>
<html>
<head>
        <title>Parameters</title>
        <script type="text/javascript">
                function params(a,b){

                        document.write("Function params was passed
",arguments.length,"Parameter <br/>");
                        for (var arg = 0; arg < arguments.length; arg++)
                                document.write(arguments[arg],"<br/>");
                        document.write("<br/>");
                }
        </script>
</head>
<body>
        <script type="text/javascript">

        params("Suwash");
        params("Suwash","Nirmal");
        params("Suwash","Nirmal","Samundra");
        </script>
</body>
</html>
```

Output



```
Function params was passed 1Parameter
Suwash

Function params was passed 2Parameter
Suwash
Nirmal

Function params was passed 3Parameter
Suwash
Nirmal
Samundra
```

(See one example of "median" function in book).

Constructors

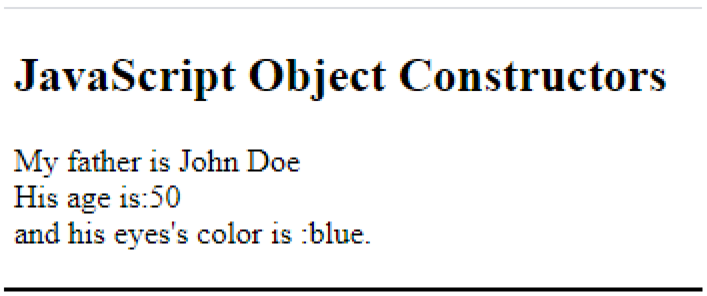
- JavaScript constructors are special methods that create and initialize the properties for newly created objects
- Every new expression must include a call to a constructor, whose name is the same as the object being created.
- A constructor obviously must be able to reference the object on which it is to operate.
- JavaScript has a predefined reference variable for this purpose, named "this".
- The "this" variable is used to construct and initialize the properties of the object.
  Eg:
  ```
  function car (new_make, new_model, new_year ) {
              this.make = new_make;
              this.model = new_model;
              this.year = new_year;
  }
  ```
  This constructor could be used as in the following:
  ```
  my_car = new car("Ford", "Contour SVT", "2000" );
  ```

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Constructors</h2>
<p id="demo"></p>

<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
```

```
}
// Create a Person object
var myFather = new Person("John", "Doe", 50, "blue");

// Display age
document.getElementById("demo").innerHTML =
"My father is "+ myFather.firstName + " "+ myFather.lastName +"<br/> His age is:" +
myFather.age + "<br/>and his eyes's color is :" + myFather.eyeColor +".";

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

Output



JavaScript Object Constructors

```
My father is John Doe
His age is:50
and his eyes's color is :blue.
```

Pattern Matching Using Regular Expression

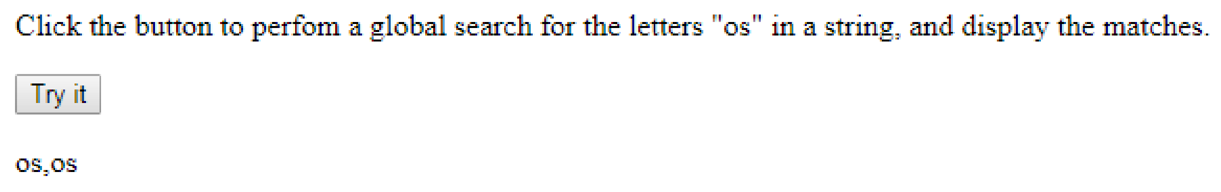
- JS has a powerful pattern-matching capabilities based on regular expressions.
- There are two approaches to pattern matching in JS
- "RegExp" and "String" objects are used for pattern matching.

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to perfom a global search for the letters "os" in a string, and display the
matches.</p>

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

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

<script>
function myFunction() {
  var str = "cosmos college of management and Technology";
  var res = str.match(/os/g);
  document.getElementById("demo").innerHTML = res;
}
</script>

</body>
</html>
```

Note: If the regular expression does not include the *g* modifier (to perform a *global* search), the match( ) method will return only the first match in the string.

Output



```
Click the button to perfom a global search for the letters "os" in a string, and display the matches.

Try it

os,os
```

Errors in Scripts

- JS interpreter is capable of detecting various problems with script.
- In most cases script errors cause the browser not to display the document, without producing any error messages.
- Error message may be: "Disable script debugger " "A Runtime error has occurred" etc.

Example

```
<!DOCTYPE html>
<html>
<head>
        <title>Debugging help</title>
</head>
<body>
<script type="text/javascript">
        var row;
        row = 4;
        while (row==4 {         //here ")" is missing in while statement
                document.write("row is", row, "<br/>");
                row++;
        }
</script>
</body>
</html>
```

JavaScript and HTML Document

Introduction

- Client-side JS does not have any language constructs that are not core JavaScript. Instead, it defines the collection of objects, methods, and properties that allow scripts to interact with HTML documents on the client side.
- Here, this topic give a brief overview of the Document Object Model and the techniques for accessing HTML documents is JavaScript.
- The concept of event handling are also introduced.
- Application of event handling are introduced through three examples.
  - ✓ The first illustrates the use of load event
  - ✓ Second illustrates the use of click event
  - ✓ Third illustrates the use of change event to check the format of the textual input in a form.

The Document Object Model

- DOM is a model that defines interfaces between HTML documents and applications programs.
- It is an abstract model because it must apply to a variety of application programming languages.
- The actual DOM specification consists of a collection of interface, including one for each document tree node type.
- The interfaces are similar to Java interfaces and C++ abstract classes. They define the objects, methods and properties that are associated with their respective node types.
- With DOM, users can write code in programming languages to create documents, move around in their structures and change, add, or delete elements and their content.
- Document in DOM have a treelike structure, but there can be more than one tree in a document.
- Because DOM is an abstract interface, it does not dictate that documents must be implemented as trees or collections of trees. So, the relationships among the elements of a document can be represented in any number of different ways.
- The DOM is not a data model, it is an object-oriented model. Therefore, the elements of a document using the DOM model are objects, with both data and operations.
- The data are called properties and the operations are called methods.

Example

```
<html>

<head><tiltle>A simple document</title>
</head>
<body>
<table>
        <tr>
                <th> Breakfast </th>
                <td> 0  </td>
                <td> 1 </td>
        </tr>
        <tr>
                <th> Lunch </th>
                <td> 1 </td>
                <td> 0 </td>
        </tr>
</table>
<body>
</html>
```

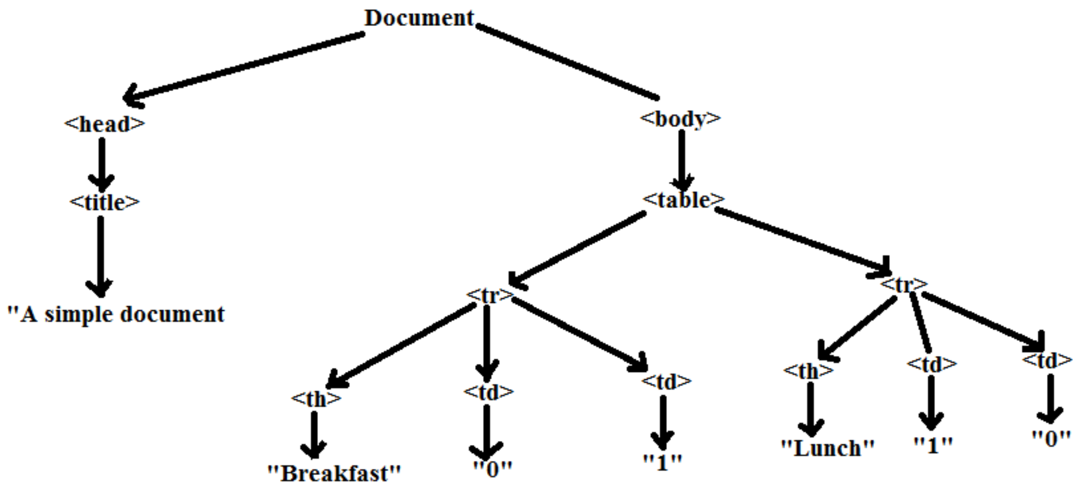


Fig: The DOM structure for a simple document

- A language that is designed to support the DOM must have a binding to the DOM constructs.
- In JS, binding to the DOM, HTML element are represented as objects and element attributes are represented as properties.
  For example, the following element would be represented as an object with two properties, type and name, with the values "text" and "address", respectively.

```
< input type = "text" name = "address" >
```

<u>Element Access in JavaScript</u>

- There are several ways an HTML form element can be addressed in JavaScript. The original (DOM 0) way is to use the forms and elements arrays.
  Example

```
<html>
<head><title>Access to form elements.</title>
</head>
<body>
<form action = "">
```

```
        <input type = "button" name = "turnItOn">
</form>
</body>
</html>
```

- We refer to JS object that is associated with an HTML element as its DOM address. The DOM address of the button in this example, using the forms and elements array, is

  ```
  document.forms[ 0 ].element[ 0 ]
  ```

  The problem with this approach is that the DOM address is defined by address elements that could change. For example, if a new button were added before the turnItOn button, the DOM address just shown would be wrong
- Another approach to DOM addressing is to use element names. For this, the element and its enclosing elements, up to but not including the body element, must include name attributes.

  ```
  <html>
  <head><title>Access to form elements</title>
  </head>
  <body>
  <form name = "myForm" action = "">
          <input type = "button" name = "tutnItOn">
  </form>
  </body>
  </html>
  ```

  Using the name attribute, the button's DOM address is :

  ```
  document.myForm.turnItOn
  ```

  The one drawback of this approach is that the W3C XHTML strict standard does not allow the name attribute in the form element, even though the attribute is now legal for form elements.
- The third approach to element addressing is to use the JS method "getElementById", which is defined in DOM 1. If an element's identifier (id) is unique in the document, this approach works, regardless of how deeply the element is nested in other elements in the documents. For example, if the id attribute of our button is set to "turnItOn", the following could be used to get the DOM address of that button element:

  ```
  document.getElementById("turnItOn")
  ```

<u>**Events and Event Handling**</u>

- The HTML 4.0 standard provided the first specification of an event model for documents.
- This is sometimes referred to as the DOM 0 event model. It is supported by all browsers that include JS.
- A complete and comprehensive event model was specified by DOM2.

Basic Concept of Event Handling

- An event is a notification that something specific has occurred, either with the browser, such as the completion of the loading of a page, or because of a browser user action, such as a mouse click on a button. Strictly, an event is an object that is implicitly created by the browser and the JS system in response to something happening.
- An event handler is a script that is implicitly executed in response to the appearance of an event. Event handler enables a web document to be responsive to browser and user activities.
- One of the most common example of event handlers is to check for a simple errors and omission in the elements of a form, either when they are changed or when the form is submitted. This saves the time of sending the form data to the server, where its correctness then must be checked by a server-resident program of script before it can be processed.
- This is similar to exception handling in C++. Both events and exceptions occur at unpredictable times, and both often require some specific program actions.
- Event handlers are scripts whose execution can be triggered by the appearance of an of an event. The process of connecting an event handler to an event is called registration.
- The "write " method of document should never be used in an event handler. Because events usually occur after the whole document is displayed. If write appears in an event handler, the content generated by it might be placed over the top of the existing document.

Events, Attributes and Tags

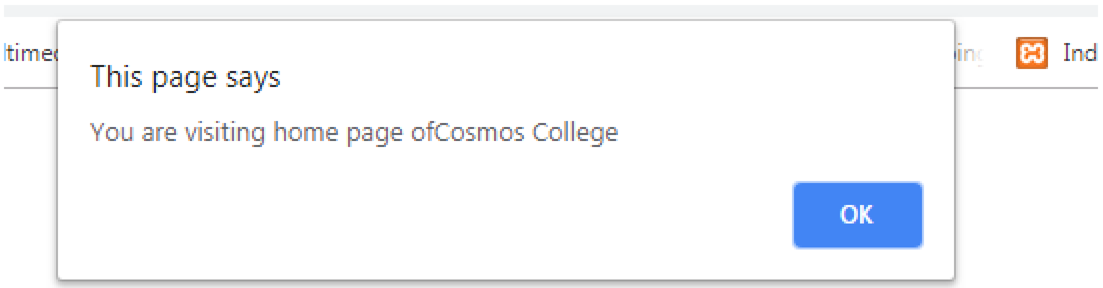
Event	Tag Attributes
Abort	Onabort
Blur	Onblur
Change	Onchange
Click	Onclick
Error	Onerror
Focus	Onfocus
Load	Onload
Mouseout	onmouseOut
Mouseover	onmouseOver
Reset	Onreset
Resize	Onresize
Select	Onselect
Submit	Onsubmit
Unload	Onunload

- Some additional events-namely dragdrop, keydown, keyup, mouse-down, mousemove, mouseup and move- are handled in more complicated ways.

Example : Using load event

```
<!DOCTYPE html>
<html>

<head>
        <title>The onLoad event handler</title>
        <script type="text/javascript">
        function load_greeting(){
                alert("You are visiting home page of" +"Cosmos College");
                        }
        </script>
</head>
<body onload = "load_greeting()">
</body>
</html>
```

Output



```
This page says
You are visiting home page ofCosmos College
```

Event Handlers for Button Events

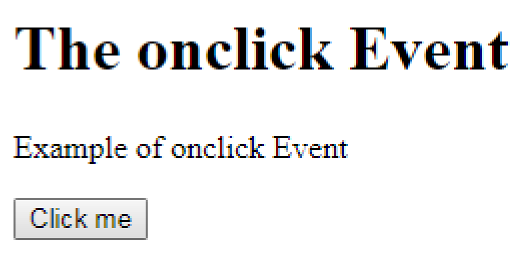
- Buttons in a web document provide a simple and effective way to collect multiple-choice input from the browser user.
- Example, a set of radio buttons that enables the user to choose information about a specific airplane.
- Radio buttons must be handled somewhat differently from other form elements, because all radio buttons in a group have the same name. That disallows using their names to get their DOM addresses.
- Every radio buttons has its own position in the element array. So, rather than checking the value of one DOM address, we must test all of the positions in the elements array that are associated with the radio buttons in the group to determine which one is set.
- The checked property of the radio button object is set to true if the button is checked. Its associated element of the elements array is also set to true.
- If the only elements of a form are in a group of radio buttons, we search the form's whole elements array.

Example 1

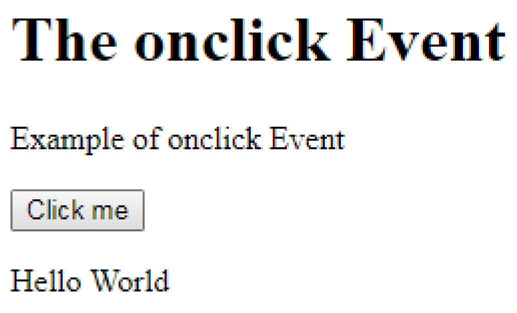
```
<!DOCTYPE html>
```

```
<html>
<body>
<h1>The onclick Event</h1>
<p>Example of onclick Event</p>
<button onclick="myFunction()">Click me</button>
<p id="demo"></p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```

Output



```
The onclick Event

Example of onclick Event

Click me
```

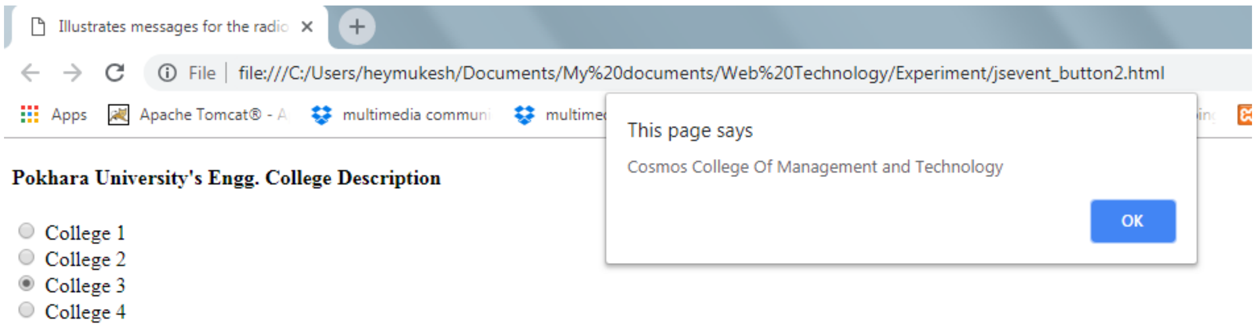


```
The onclick Event

Example of onclick Event

Click me

Hello World
```

<u>Example 2</u>

```
<!DOCTYPE html>
<html>
<head>
        <title>Illustrates messages for the radio buttons </title>
<script type="text/javascript">
        //Event handler collection for radio button
        function collegeChoice(){
                var college;
                //put DOM address of element array in a local variable
                var radioElement = document.getElementById("myForm").elements;
                //Determine which button was pressed
                for(var index = 0; index < radioElement.length; index++){
                        if(radioElement[index].checked){
                                college = radioElement[index].value;
                                break;
                        }
                }
//produce an alert message about the chosen college
switch(college){
        case "1":
        alert("Nepal College Of Information Technology");
        break;
        case "2":
        alert("Nepal Engineering College");
        break;
        case "3":
        alert("Cosmos College Of Management and Technology");
        break;
        case "4":
        alert("Everest Engineering College");
        break;
        default:
        alert("Error in choice!!!");
        break;
}
        }
</script>
</head>
<body>
<h4>Pokhara University's Engg. College Description</h4>
<form id = "myForm" action = "">
        <p>
                <input type="radio" name="CollegeButton" value="1"
                        onclick = "collegeChoice()"/>
                        College 1
                        <br/>
                <input type="radio" name="CollegeButton" value="2"
                        onclick="collegeChoice()"/>
                        College 2
                        <br/>
                <input type="radio" name="CollegeButton" value="3"
                        onclick="collegeChoice()"/>
                        College 3
                        <br/>
                <input type="radio" name="CollegeButton" value = "4"
                        onclick = "collegeChoice()"/>
                        College 4
        </p>
</form>
</body>
</html>
```

Output


Illustrates messages for the radio
File
file:///C:/Users/heymukesh/Documents/My%20documents/Web%20Technology/Experiment/jsevent_button2.html
Apps
Apache Tomcat® - A
multimedia communi
multimed
OK

This page says
Cosmos College Of Management and Technology

Pokhara University's Engg. College Description

- ○ College 1
- ○ College 2
- ● College 3
- ○ College 4

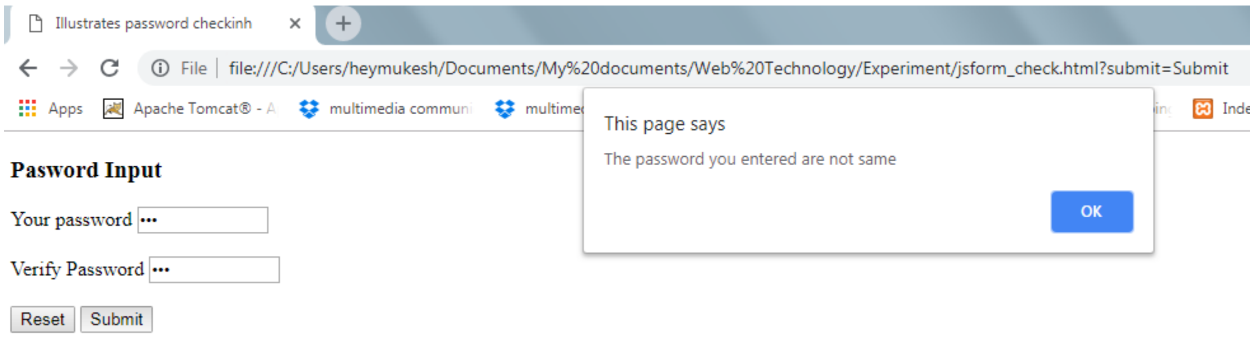
Checking Form Input

- When a user fills in a form input element incorrectly and a JavaScript event-handler function detects the error, the function should do several things.
- First, it should produce an alert message indicating the error to the user and specifying the correct format of the input.
- Next, it should cause the input element to be put in focus, which positions the cursor in the element. This is done with the "focus" function.
  Example:
  document.getElementById("phone").focus( );
  This puts the cursor in the phone text box. Finally, the function should select the element, which highlights the text in the element. This is done with the "select" function, as in following example:
  document.getElementById("phone").select( );
- If an event handler returns "false", that tells the browser not to perform any default actions of the event. For example, if the event is a click on the Submit button, the default action is to submit the form data to the server for processing.
- If user input is being validated in an event handler that is called when the submit event occurs and some of the input is incorrect, the handler should return "false" to avoid sending the bad data to the server.
- We use the convention that event handlers that check form data always return false if they detect an error, and true otherwise.
- When a form requests a password from the user and that password will be used in future sessions, the user is often asked to enter the password a second time for validation. A JavaScript function can be used to check that the two entered passwords are the same.
  Example

```
<!DOCTYPE html>
<html>
<head>
        <title>Illustrates password checkinh</title>
        <script type="text/javascript">
                //The event handler function for password checking

                function chkPasswords(){
```

```
                var init = document.getElementById("initial");
                var sec = document.getElementById("second");
                if(init.value == ""){
                    alert("You did not enter a pasword.Please enter");
                    init.focus();
                    return false;
                }
                if(init.value != sec.value){
                    alert("The password you entered are not same");
                    init.focus();
                    init.select();
                    return false;
                }else
                return true;
            }
    </script>
</head>
<body>
<h3>Pasword Input</h3>
<form action = "">
    <p>
        Your password
        <input type = "password" id = "initial" size = "10"/>
        <br/><br/>

        Verify Password
        <input type="password" id = "second" size = "10"/>
        <br/><br/>

        <input type="reset" name = "reset"/>
        <input type="submit" name = "submit"/>
    </p>
</form>
<script type="text/javascript">
    //Set submit button onsubmit property to the event handler
    document.forms[0].onsubmit = chkPasswords;
</script>
</body>
</html>
```

Output


Illustrates password checkinh
File
file:///C:/Users/heymukesh/Documents/My%20documents/Web%20Technology/Experiment/jsform_check.html?submit=Submit
Apps
Apache Tomcat® - A
multimedia communi
multimed
Inde
Reset
Submit
OK

Pasword Input

Your password •••

Verify Password •••

This page says

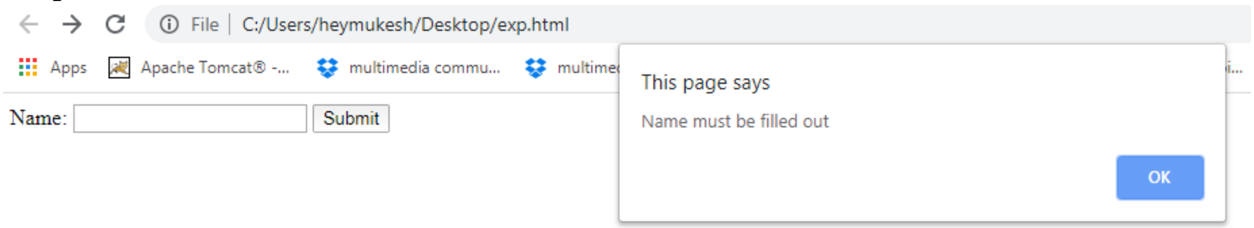
The password you entered are not same

Checking validity of the form values

Example 1 Checking form name spacing

```
<!DOCTYPE html>
<html>
<head>
  <title>Checking Form Name spacing </title>
<script>
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<form name="myForm" action="" onsubmit="return validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Output



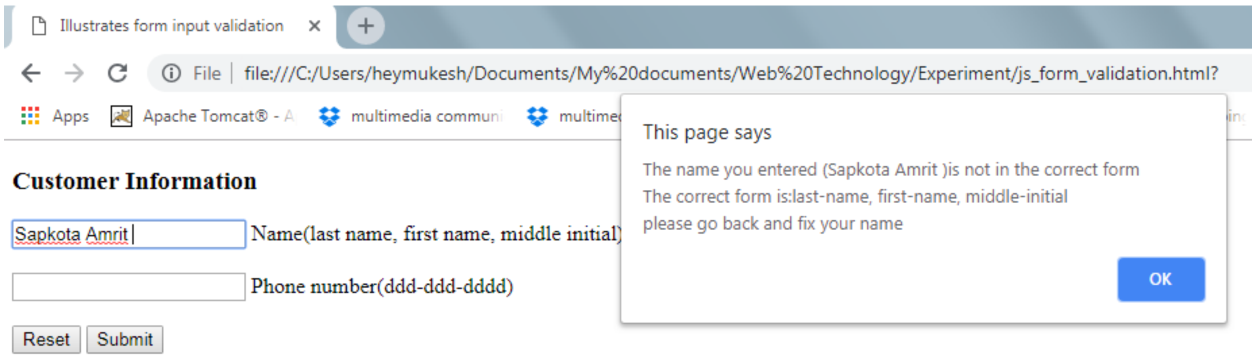
Name:

This page says

Name must be filled out

Example 2 form input validation

```
<!DOCTYPE html>
<html>
<head>
        <title>Illustrates form input validation</title>
        <script type="text/javascript">
                //The event handler function for the name test box
                function chkName(){
                        var myName = document.getElementById("custName");
                        //Test the format of the input name
                        //Allow the spaces after the commas to be optional
                        //Allow the period after the initial to be optional

                        var pos = myName.value.search(/^\w+, ?\w+, ?\w.?/);

                        if(pos!=0){
                                alert("The name you entered ("+ myName.value +")
is not in the correct form\n" +
                                         "The correct form is:" +
                                         "last_name, first_name, middle_initial\n" +
                                        "please go back and fix your name");
                                myName.focus();
                                myName.select();
                                return false;
                                }else
                                return true;
                        }
                        //The Event handler function for the phone number text box

                        function chkPhone(){
                                var myPhone = document.getElementById("phone");
                                //Test the format of the input phone number
                                var pos = myPhone.value.search(/^\d{3}-\d{3}-\d{4}$/);

                                if(pos!=0){
                                        alert("The phone number you entered ("+ myPhone.value
+")is not in the correct form.\n" +
                                                "The correct form is: ddd-ddd-dddd\n" +
                                                "please go back and fix your Phone number");
                                        myPhone.focus();
                                        myPhone.select();
                                        return false;
                                }else

                                return true;
                        }
                        function chkEmail()
{
        var emailAddr = document.getElementById("email").value;
 if (/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(emailAddr))
  {
   return (true);
  }
   alert("You have entered an invalid email address! format is a@mail.com");
   return (false);
}

        </script>
</head>
<body>
<h3>Customer Information</h3>
<form action="">
        <p>
                <input type="text" id="custName"/>
                Name(last name, first name, middle initial)
                <br/><br/>

                <input type="text" id="phone"/>
                Phone number(ddd-ddd-dddd)
                <br/><br/>

                <input type="text" id="email"/>
                E-Mail(...@...)
                <br/><br/>

                <input type="reset" id="reset"/>

                <input type="submit" id="submit"/>
        </p>
</form>
<script type="text/javascript">
        //set form element object properties to their
        //corresponding event handle functions
        document.getElementById("custName").onchange = chkName;
        document.getElementById("phone").onchange = chkPhone;
        document.getElementById("email").onchange = chkEmail;
</script>
</body>
</html>
```



Illustrates form input validation

file:///C:/Users/heymukesh/Documents/My%20documents/Web%20Technology/Experiment/js_form_validation.html?

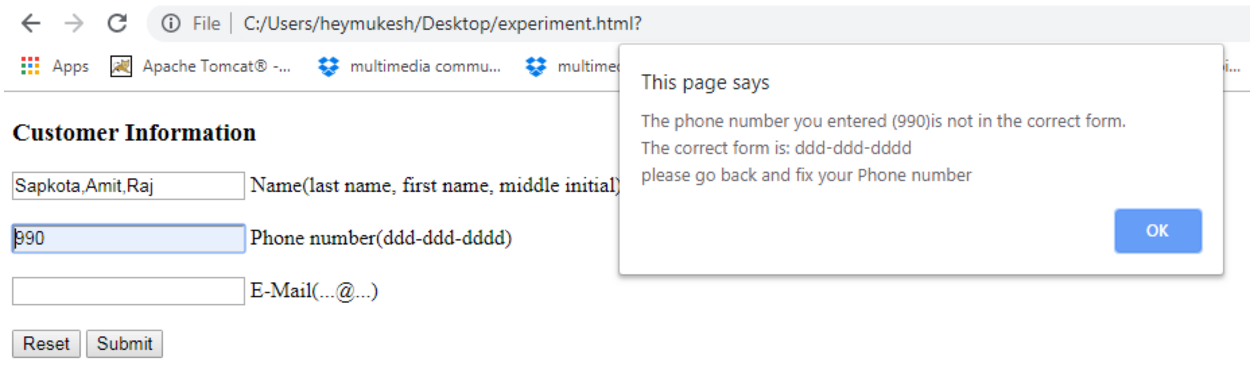
Customer Information

Sapkota Amrit	Name(last name, first name, middle initial)
	Phone number(ddd-ddd-dddd)

Reset Submit

This page says
The name you entered (Sapkota Amrit)is not in the correct form
The correct form is:last-name, first-name, middle-initial
please go back and fix your name

OK



C:/Users/heymukesh/Desktop/experiment.html?

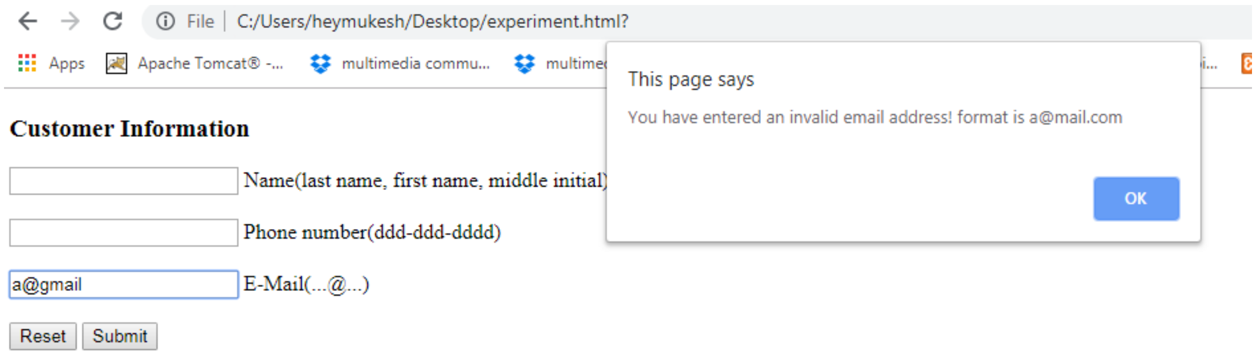
Customer Information

Sapkota,Amit,Raj	Name(last name, first name, middle initial)
990	Phone number(ddd-ddd-dddd)
	E-Mail(...@...)

Reset Submit

This page says
The phone number you entered (990)is not in the correct form.
The correct form is: ddd-ddd-dddd
please go back and fix your Phone number

OK



C:/Users/heymukesh/Desktop/experiment.html?

Customer Information

	Name(last name, first name, middle initial)
	Phone number(ddd-ddd-dddd)
a@gmail	E-Mail(...@...)

Reset Submit

This page says
You have entered an invalid email address! format is a@mail.com

OK

<u>Event Handler Registration</u>

- DOM 0 event model uses two different ways of registering event handlers.
- First, the handler code can be assigned as a string literal to the event's associated attribute in the element. Second, the name of the handler function can be assigned to the property associated with the event.
- The method "addEventListener", takes three parameters, the first of which is the name of the event as a string literal. For example, "mouseup" and "submit" would be legitimate first parameter.

- The second parameter is the handler function. This could be specified as the function code itself or as the name of a function that is defined elsewhere.
- The third is a Boolean value that specifies whether the handler is enabled for calling during the capturing phase. If the value is "false", the handler can be called either at the target node or on any node reached during bubbling.

  Example of DOM 2 Event Model

```
<!DOCTYPE html>
<html>
<head>
        <title>Illustrates form input validation with DOM 2</title>
        <script type="text/javascript">
                //The event handler function for the name test box
                function chkName(event){
                        //Get the target node of the event
                        var myName = event.currentTarget;
                        var pos = myName.value.search(/\w+, ?\w+, ?\w.?/);

                        if(pos!=0){
                                alert("The name you entered ("+ myName.value +")
                                        is not in the correct form\n" +
                                         "The correct form is:" +
                                         "last_name, first_name, middle_initial\n" +
                                        "please go back and fix your name");
                                myName.focus();
                                myName.select();
                                }
                        }
                        //The Event handler function for the phone number text box

                        function chkPhone(event){
                                //Get the target node of the event
                                var myPhone = event.currentTarget;
                                //Test the format of the input phone number
                                var pos = myPhone.value.search(/^\d{3}-\d{3}-\d{4}$/);

                                if(pos!=0){
                                        alert("The phone number you entered ("+ myPhone.value
                                                +")is not in the correct form.\n" +
                                                "The correct form is: ddd-ddd-dddd\n" +
                                                "please go back and fix your Phone number");
                                        myPhone.focus();
                                        myPhone.select();
                                }

                        }
```

```
	</script>
</head>
<body>
<h3>Customer Information</h3>
<form action="">
	<p>
		<input type="text" id="custName"/>
		Name(last name, first name, middle initial)
		<br/><br/>

		<input type="text" id="phone"/>
		Phone number(ddd-ddd-dddd)
		<br/><br/>

		<input type="reset" id="reset"/>

		<input type="submit" id="submit"/>
	</p>
</form>
<script type="text/javascript">
	//Get the DOM address of the elements and register the event handlers
	var customerNode = document.getElementById("custName");
	var phoneNode = document.getElementById("phone");
	customerNode.addEventListener("change",chkName,false);
	phoneNode.addEventListener("change", chkPhone,false);

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

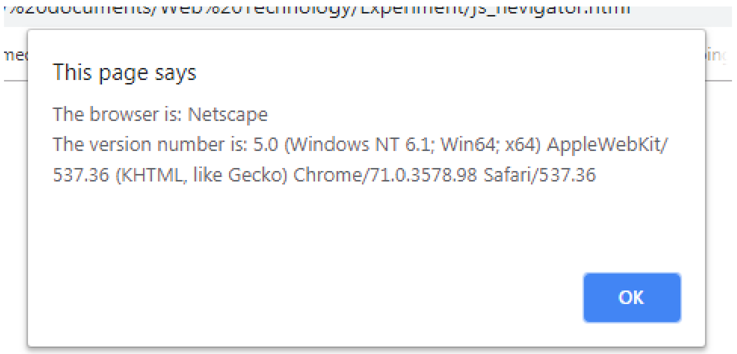
Output
(same as before)

The Navigator Object

- The navigator object indicates which browser is being used to view the HTML document.
  example

```
<!DOCTYPE html>
<html>
<head>
	<title>Using Navigator</title>
	<script type="text/javascript">
		function navProperties(){
			alert("The browser is: " + navigator.appName + "\n" +
				"The version number is: " + navigator.appVersion+"\n");
		}

	</script>
```

```
</head>
<body onload= "navProperties()">

</body>
</html>
```



```
This page says
The browser is: Netscape
The version number is: 5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
```

Dynamic Documents with JavaScript

- Through dynamic HTML, the browser user can change the document currently being displayed.
- Client-side JavaScript can be used to implement dynamic HTML documents.
- HTML elements can be initially positioned at any given location on the display. If they are positioned in the proper way, elements can be dynamically moved to new positions on the display.
- Elements can be made to disappear and reappear through user interactions
- The colors of the background and the foreground of a document can be changed.
- The font, font size, and font style can be changed. Also, the content of an element can be changed.
- Overlapping element in a document can be positioned in a specific top-to-bottom stacking order and their stacking order can be dynamically changed.
- The position of the mouse cursor on the browser display can be determined when the mouse is clicked
- Elements can be defined to allow the user to drag and drop them around the display window.

Element Positioning

- Cascading Style Sheets-Positioning(CSS-P) provides a meant not only to position any element anywhere in the document, but also to dynamically move an element to a new position in the document, using JavaScript to change the positioning style properties of the element.
- These style properties, which are appropriately named "left" and "top", dictate from the left and top of some reference point to where the element is to appear.
- Another style property, "position", interacts left and top to provide a higher level of control of placement and movement of elements.
- The position property has three possible values: absolute, relative, and static

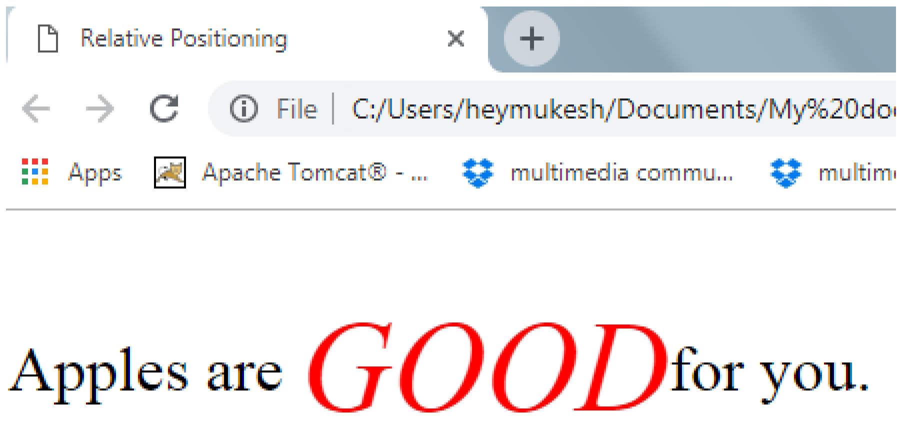
Absolute Positioning

- The absolute value is specified for position when the element is to be placed at a specific place in the document without regard to the positions of other elements.
  Example:
  ```
  <p style = "position: absolute; left: 100px; top: 200px">
          -----text-----
  </p>
  ```

<u>Relative Positioning</u>

- An element that has the position property set to relative but does not specify top and left property values is placed in the document as if the position attribute were not set at all. However, such an element can be moved later.
- If he top and left properties are given values, they displace the element by the specified amount from the position where it would have been placed.
- For example, suppose that two buttons are placed in a document and the position attribute has its default value, which is static. They appear next to each other in a row, assuming the current row had sufficient horizontal space for them. If position has been set to relative and the second button had its left property set to 50px, the effect would be to move it 50 pixels farther to the right than it otherwise would have appeared.
- Relative positioning can be used for a variety of special effects in element placement.
- It can be used to create superscripts and subscripts by placing the values to be raised or lowered in <span> tags and displacing them from their regular positions.

```
<!DOCTYPE html>
<html>
<head>
        <title>Relative Positioning</title>
</head>
<body style="font-family: Times; font-size: 24pt;">
        <p>
                Apples are <span style = "position: relative;top: 10px;font-family: Times;font-
size: 48pt;font-style: italic;color: red;">
        GOOD</span>for you.
        </p>
</body>
</html>
```



Apples are GOOD for you.

Static Positioning

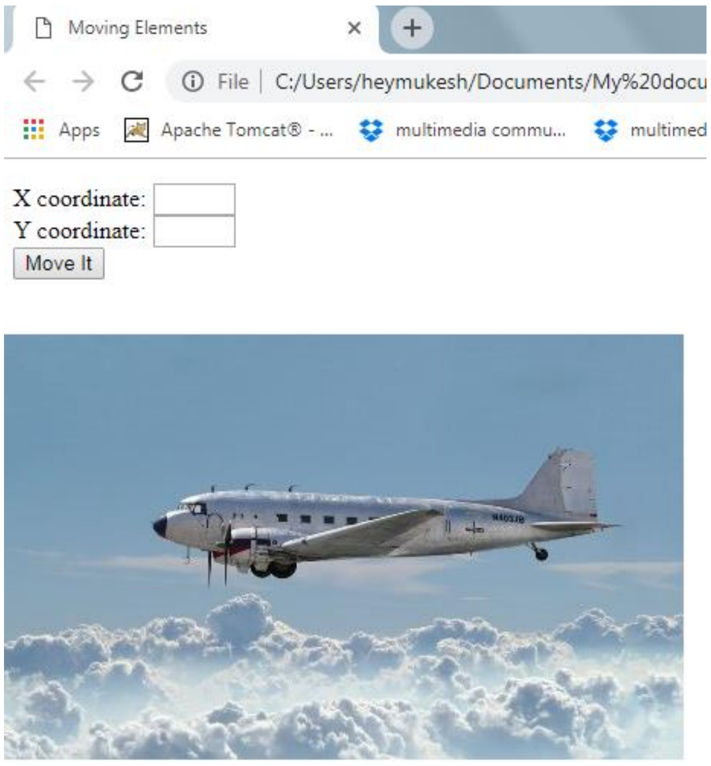
- The default value for the position property is static.
- A statically positioned element is placed in the document as if it had the position value of relative.
- The difference is that a statically positioned element cannot have its top or left properties initially set or changed later.
- Therefore, a statically placed element cannot be displaced from its normal position and cannot be moved from that position later.

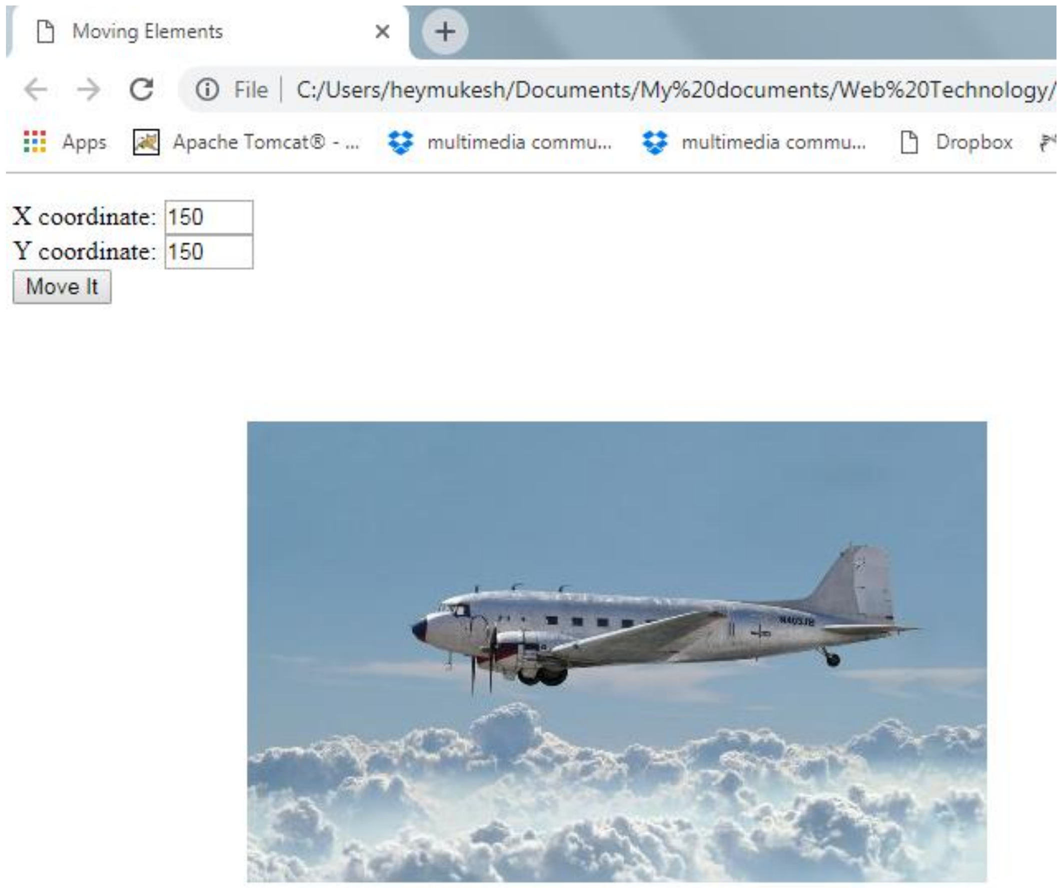
Moving Element

- If the position of the of an HTML element is set to either "absolute" or "relative", it can be moved.
- Moving an element is simple: changing the top or left property values causes the element to move within the display.

```
<!DOCTYPE html>
<html>
<head>
        <title>Moving Elements</title>
        <script type="text/javascript">
                function moveIt(movee, newTop, newLeft){
                        dom = document.getElementById(movee).style;

                        /*Change the top and left properties to perform the move. Note the
asddition of units to the input values */
                        dom.top = newTop + "px";
                        dom.left = newLeft + "px";
                }
        </script>
</head>
```

```
<body>
        <form action = "">
                <p>
                        X coordinate: <input type="text" id="leftCoord" size="3" />
                        <br/>
                        Y coordinate: <input type="text" id="topCoord" size="3"/>
                        <br/>
                                <input type="button" value="Move It"
                                onclick =
                                "moveIt('airplane',
                                document.getElementById('topCoord').value,
                                document.getElementById('leftCoord').value)" />
                                </p>
</form>
<div id = "airplane" style = "position: absolute; top: 115px; left: 0;">
        <img src="C:\Users\heymukesh\Desktop\aeroplane.jpg"
        alt = "(Picture of airplane)" />
</div>
</body>
</html>
```



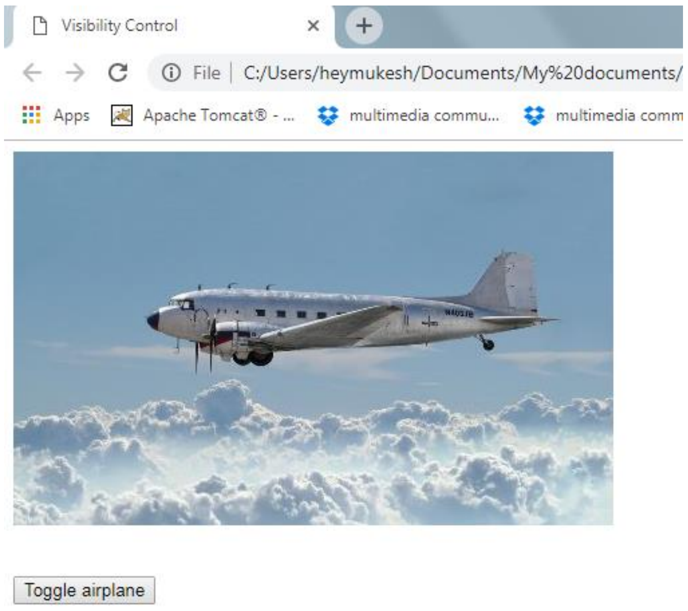


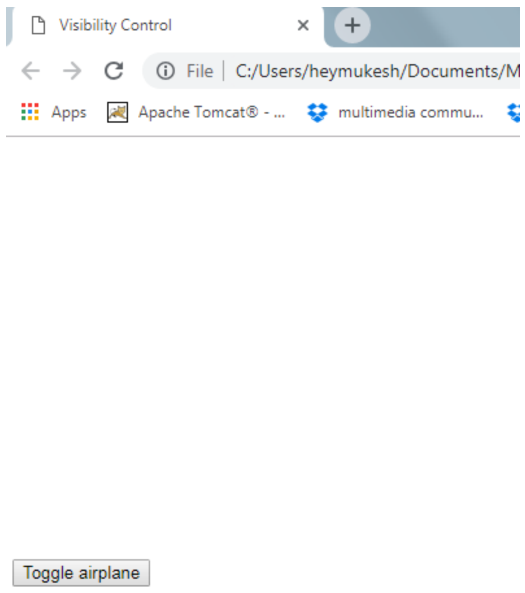
<u>Element Visibility</u>

- Document element can be made to be visible or hidden, depending upon the value of their visibility property.
- "visible" and "hidden" are used for appearance or disappearance of an element.

```
<!DOCTYPE html>
<html>
<head>
        <title>Visibility Control</title>
        <script type="text/javascript">
                function flipImag(){
                        dom = document.getElementById("airplane").style;

                        /*The Event handler function to toggle the visibility of the image plane */
                if(dom.visibility == "visible")
                        dom.visibility = "hidden";
                else
                        dom.visibility = "visible";

                }
        </script>
</head>
```

```
<body>
        <form action = "">

<div id = "airplane" style = "position: relative;
                visibility: visible;">
        <img src="C:/Users/heymukesh/Desktop/aeroplane.jpg"
        alt = "(Picture of airplane)" />
</div>
<p>
        <br/>
        <input type="button" value="Toggle airplane" onclick = "flipImag()"/>
</p>
</form>
</body>
</html>
```

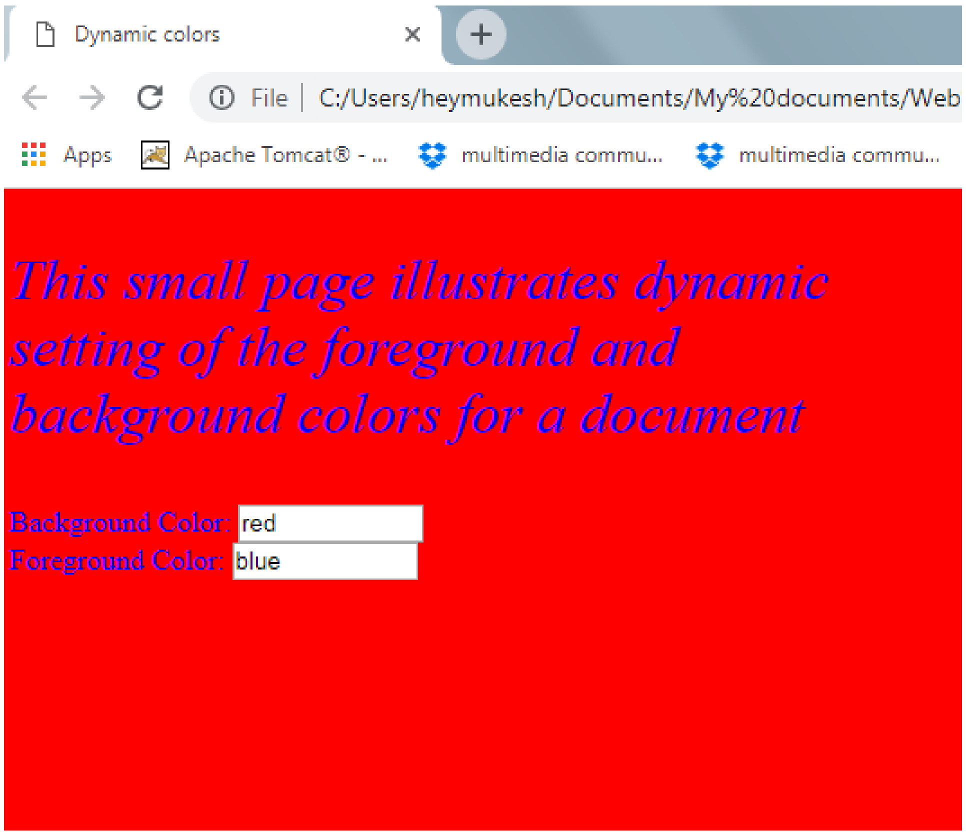




Changing Colors and Fonts

```
<!DOCTYPE html>
<html>
<head>
        <title>Dynamic colors</title>
        <script type="text/javascript">
                function setColor(where, newColor){
                        if(where == "background")
                                document.body.style.backgroundColor = newColor;
                        else
                                document.body.style.color = newColor;
                }
        </script>
</head>
<body>
<p style = "font-family: Times; font-style: italic; font-size: 24pt">
        This small page illustrates dynamic<br/>
        setting of the foreground and<br/>
        background      colors for a document
</p>
<form action="">
        <p>
                Background Color:
                <input type="text" name="background" size="10"
                 onchange = "setColor('background',this.value)"/>
<br/>
Foreground Color:
<input type="text" name="foreground" size = "10"
onchange ="setColor('foreground', this.value)"/>
```

```
<br/>
</p>
</form>
</body>
</html>
```

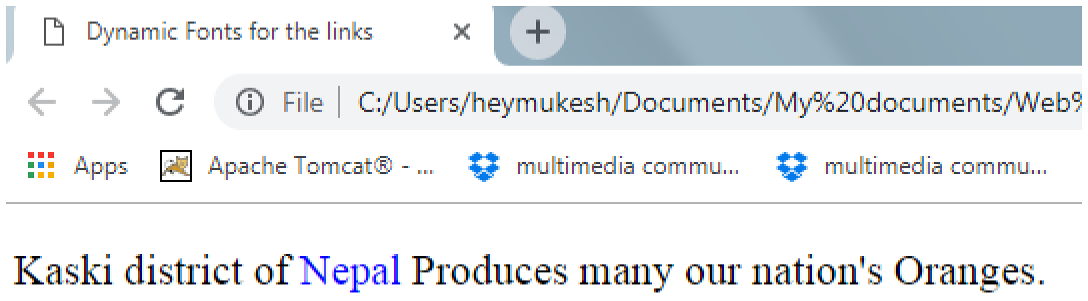


This small page illustrates dynamic setting of the foreground and background colors for a document

Background Color: red
Foreground Color: blue

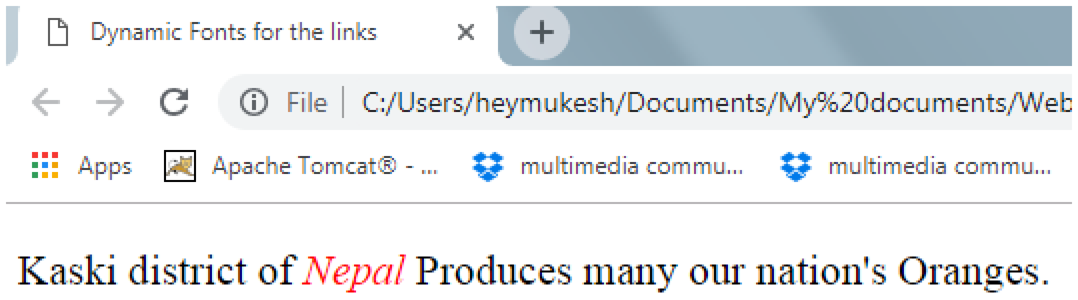
Changing Fonts

```
<!DOCTYPE html>
<html>
<head>
        <title>Dynamic Fonts for the links</title>
        <style type="text/css">
                .regText {font: Times; font-size: 16pt;}

        </style>
</head>
<body>
        <p class="regText">
                Kaski district of
        <a style="color: blue;"
        onmouseover="this.style.color ='red';
                                this.style.font = 'italic 16pt Times';"
        onmouseout = "this.style.color = 'blue';
                                this.style.font  = 'normal 16pt Times';">
        Nepal</a>
        Produces many our nation's Oranges.
        </p>
</body>
</html>
```


Dynamic Fonts for the links
File
C:/Users/heymukesh/Documents/My%20documents/Web9
Apps
Apache Tomcat® - ...
multimedia commu...
multimedia commu...

Kaski district of Nepal Produces many our nation's Oranges.

Output with the mouse cursor over the link (Nepal)


Dynamic Fonts for the links
File
C:/Users/heymukesh/Documents/My%20documents/Web
Apps
Apache Tomcat® - ...
multimedia commu...
multimedia commu...

Kaski district of *Nepal* Produces many our nation's Oranges.

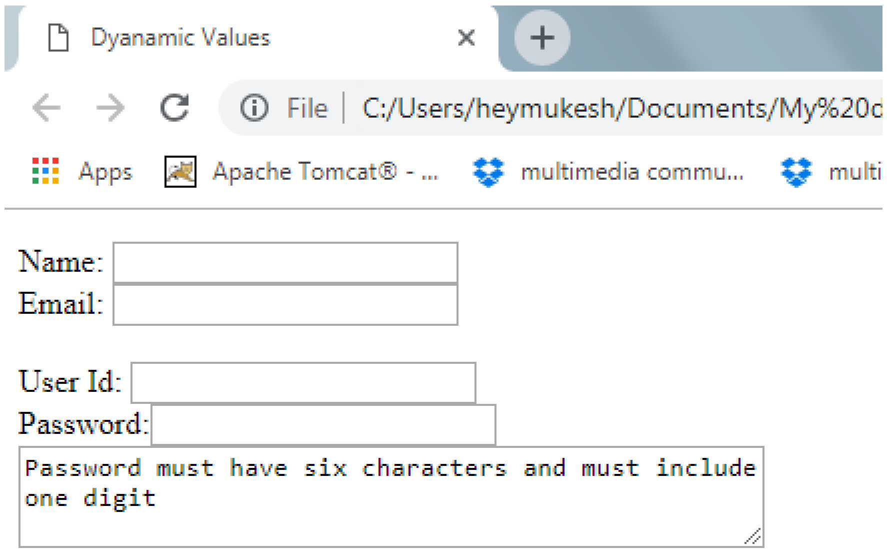
<u>Dynamic Content</u>

- This describes changing the content of HTML elements.

- The content of an element is accessed through the value property of its associated JavaScript object. So, changing the content of an element is not essentially different from changing other properties of an element.
- Following example describes the changing the content of a collection of text fields.

```
<!DOCTYPE html>
<html>
<head>
	<title>Dyanamic Values</title>
	<script type="text/javascript">
		var helpers  = ["Your name must be in the form:\n, first name, middle name, last
name",
						"Your mail id in the form user@domain",
						"Mail Id should have at least six charactrs",
						"Password must have six characters and must
include one digit"]
function message(adviceNumber) {
	document.getElementById("adviceBox").value = helpers[adviceNumber];
}
</script>
</head>
<body>
<form action = "">
	<p>
		Name: <input type="text" onmouseover="message(0)"
onmouseout="message(4)" />
		<br/>
		Email: <input type="text" onmouseover="message(1)"
onmouseout="message(4)" />
		<br/><br/>
		User Id: <input type="text" onmouseover="message(2)"
onmouseout="message(4)" />
		<br/>
		Password:<input type="text" onmouseover="message(3)"
onmouseout="message(4)" />
		<br/>
		<textarea id="adviceBox" rows="3" cols="50" style="position: absolute;left: 250;
top = 0">

		</textarea>>
	</p>

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

Output



Name:
Email:

User Id:
Password:

```
Password must have six characters and must include
one digit
```

Fig: output when mouse cursor is over the page password

Stacking Elements

- The placement of elements in the in the third dimension (i.e. other than top and left) dimension is controlled by the z-index.
- The JavaScript style property associated with the z-index attribute is zIndex.

```
<!DOCTYPE html>

  <head>

    <title> Dynamic stacking of images </title>

     <script type = "text/javascript">

      var topp = "airplane3";

// The event handler function to move the given element

//  to the top of the display stack

function toTop(newTop) {

// Set the two dom addresses, one for the old top

//  element and one for the new top element

  var domTop = document.getElementById(topp).style;

  var domNew = document.getElementById(newTop).style;
// Set the zIndex properties of the two elements, and
//  reset top to the new top
  domTop.zIndex = "0";
  domNew.zIndex = "10";
  topp = newTop;
}
    </script>
    <style type = "text/css">
      .plane1 {position: absolute; top: 0; left: 0;
               z-index: 0;}
      .plane2 {position: absolute; top: 50px; left: 110px;
               z-index: 0;}
      .plane3 {position: absolute; top: 100px; left: 220px;
               z-index: 0;}
    </style>
  </head>
  <body>
    <p>
      <img class = "plane1"  id = "airplane1"
           src = "airplane1.jpg"
           alt = "(Picture of an airplane)"
           onclick = "toTop('airplane1')" />
      <img class = "plane2"  id = "airplane2"
           src = "airplane2.jpg"
           alt = "(Picture of an airplane)"
           onclick = "toTop('airplane2')" />
```

```
    <img class = "plane3"  id = "airplane3"

       src = "airplane3.jpg"

       alt = "(Picture of an airplane)"

       onclick = "toTop('airplane3')" />

   </p>

  </body>

</html>
```

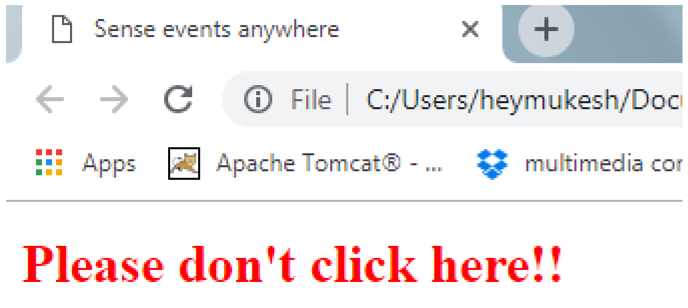
Reacting to a mouse click

- "mousedown" and "mouseup" events are used to show and hide.

```
<!DOCTYPE html>
<html>
<head>
        <title>Sense events anywhere</title>
        <script type="text/javascript">
                function displayIt(){
                        document.getElementById("message").style.visibility = "visible";
                }
                function hideIt(){
                        document.getElementById("message").style.visibility = "hidden";

                }

        </script>
</head>
<body>
        <p>
                <span id = "message"
                        style="color:red; visibility:hidden;
                                                font-size: 20pt;font-size: italic;
                                                font-weight: bold;">
                        Please don't click here!!
                        </span>
                </p>
<script type="text/javascript">
        document.onmousedown = displayIt;
        document.onmouseup  = hideIt;
</script>
</body>
</html>
```

Output

When mouse clicked



Please don't click here!!

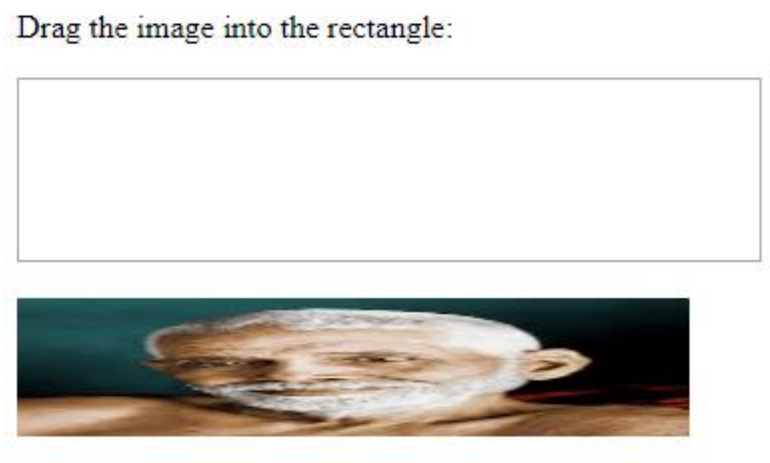
Drag And Drop

- Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

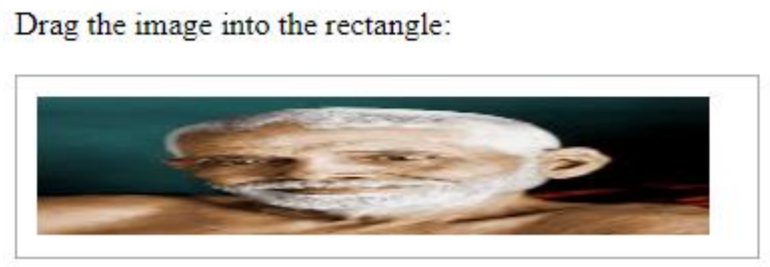
```
<!DOCTYPE HTML>
<html>
<head>
<style>
#div1 {
  width: 350px;
  height: 70px;
  padding: 10px;
  border: 1px solid #aaaaaa;
}
</style>
<script>
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<p>Drag the image into the rectangle:</p>
```

```
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
<br>
<img id="drag1" src="C:\Users\heymukesh\Desktop\a.png" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

Output



Drag the image into the rectangle:



Drag the image into the rectangle: