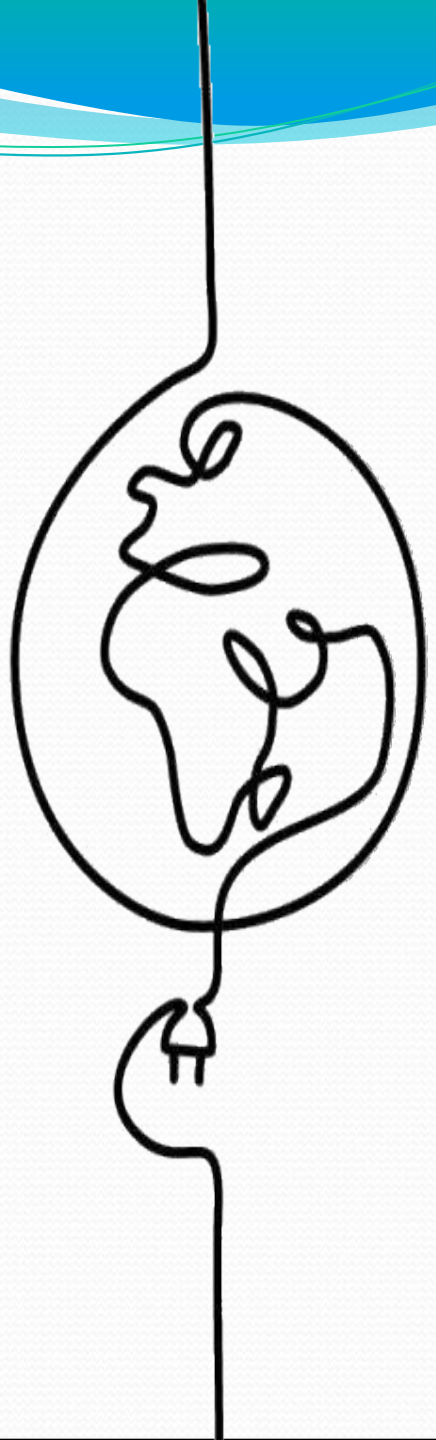
UNIT -3
JAVA SCRIPT

Introduction to Scripting



Introduction

Scripting

Introduction	Web scripting languages are programming languages which support logic building to make web pages dynamic and interactive
	Some web scripting languages : VBScript, JavaScript, Jscript and ECMA Script
	Browser includes scripting interpreter
	Choosing a scripting language for any web application is primarily based on : • Browser compatibility • Programmer familiarity
	Scripts can be executed on client or server (JavaScript can be used with client or server, but mainly used for client side scripting)

Introduction

JavaScript– Core Features

Infosys | Education, Training and Assessment

Core Features	An interpreted scripting language
	Embedded within HTML
	Minimal syntax - Easy to learn (C syntax and java OOC)
	Mainly used for client side scripting because it is supported by all the browsers
	Designed for programming user events
	Platform independent / Architecture neutral
	JavaScript is object based and action-oriented

Introduction

Client Vs. Server Scripting

Client Side Scripting	Server Side Scripting
Runs on the user's computer i.e. Browser interprets the script.	Runs on the Web server and sends the output to the browser in HTML format.
Source code is visible to the user. (Source code is downloaded to the client and executed in browser).	Source code is not visible to the user. Server side source is executed on server.
Used for client side validations and functionality for the user events.	Used for business logic and data access from the database. The pages are created dynamically.
Depends on the browser and version.	Do not depend on the client, any server side technology can be used.

Introduction

Embedding JavaScript into HTML page

- <SCRIPT>.....</SCRIPT> tag
- TYPE - the scripting language used for writing scripts

EMBEDDED JavaScript

```
<SCRIPT TYPE="text/javascript">
        ----
                (JavaScript code goes here)
        ----
</SCRIPT>
```

EXTERNAL JavaScript

```
<SCRIPT TYPE="text/javascript"  src="External.js"></script>
```

- JavaScript can be enabled or disabled in browsers

Introduction

JavaScript – Statements & Comments

Statements & Comments	A semicolon ends a JavaScript statement
	Comments • Supports single line comments using // and multi line comments using /*.....*/
	JavaScript is case sensitive

Introduction

Embedding JavaScript into HTML page - Example

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
      <head>
            <link rel="stylesheet" href="../css/style.css" type="text/css" />
            <script type="text/javascript" src="../js/scripting.js"></script>
            <title>Adding Supplier</title>

            <script type="text/javascript">

                  /* Validating Supplier Details */
                  function supplierDetailsValidation(){
                        var sname = document.supplierForm.supplierName.value;
                        var emailId = document.supplierForm.emailId.value;
                        var contactNo =
                                    document.supplierForm.contactNo.value;

                  //Checking if Supplier Name is empty
                        if(validateEmpty(sname)){
                              alert("Supplier name cannot be blank");
                              return false
                        }
```

External JavaScript

Embedded JavaScript

Introduction

Statements & Comments - Example

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
      <head>
            <link rel="stylesheet" href="../css/style.css" type="text/css" />
            <script type="text/javascript" src="../js/scripting.js"></script>
            <title>Adding Supplier</title>

            <script type="text/javascript">

                  /* Validating Supplier Details */
                  function supplierDetailsValidation(){
                        var sname = document.supplierForm.supplierName.value;
                        var emailId = document.supplierForm.emailId.value;
                        var contactNo =
                                    document.supplierForm.contactNo.value;

                  //Checking if Supplier Name is empty
                        if(validateEmpty(sname)){
                              alert("Supplier name cannot be blank");
                              return false
                        }
```

Multiline Comment

Semicolon acting as the delimiter

Single line Comment

Introduction

- It is a scripting language used to enhance the functionality and appearance of web pages.
- **First Example:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Ist javascript</title>
    <script type = "text/javascript">
    document.writeln("<h1>Welcome to JavaScript Programming!</h1>" );
    </script>

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

Welcome to JavaScript Programming!

Introduction

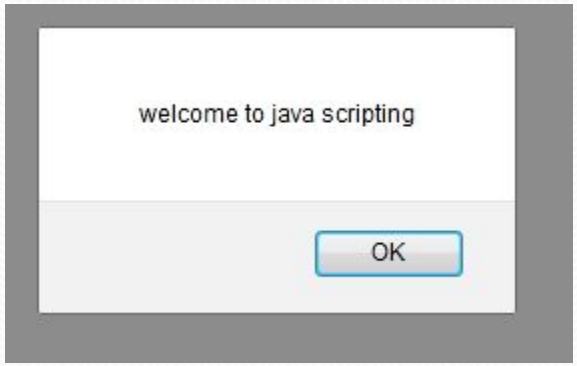
- **<script> :** To indicate to the browser that the text which follows is part of a script
- **document:** Object which represents the HTML5 document the browser is currently displaying
- **writeln:** Method to write a line of HTML5 markup in the HTML5 document
- Every statement in script ends with a semicolon (also known as the **statement terminator**)

Introduction

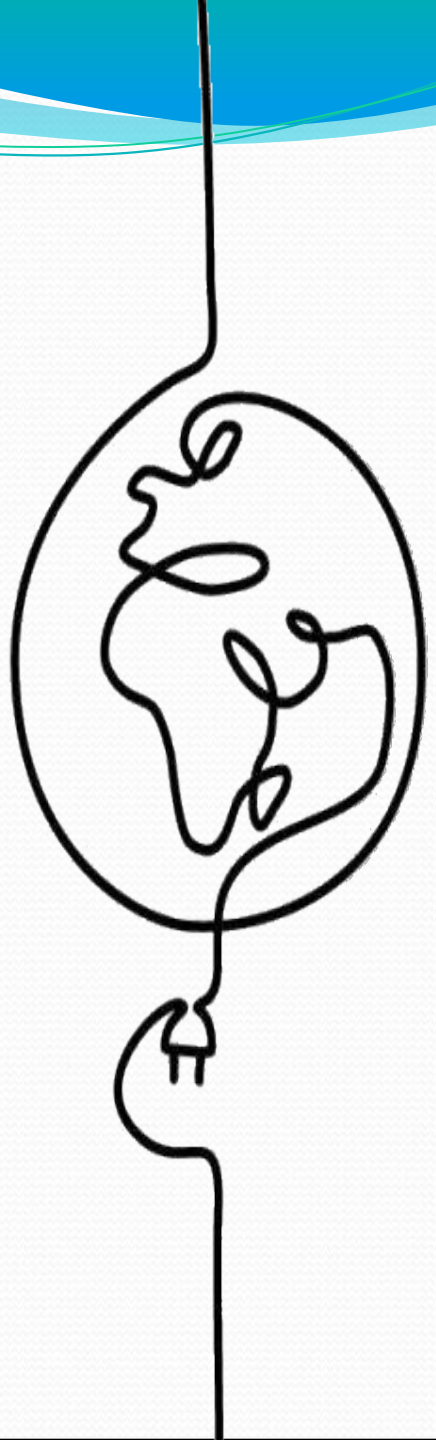
Display using alert box:

```
<script type = "text/javascript">
window.alert("welcome to java scripting");
</script>
```

- window: The script uses the browser's window object to display an alert dialog
- alert: To indicate that the browser is presenting a message to the user



Variables and Data Types



Variables

Variables

- Must start with a letter or an underscore and can have digits.
- Can be defined in 2 ways:
  - Using keyword 'var' and Without using keyword 'var'

Using keyword 'var' :

```
<script type= "text/javascript">
    var num1=10; // Global variable
    function demoOne() {
            var num2 = 20; // Local to the function
    }
    function demoTwo() {
            num1=30; //'num1' is accessible
            num2=40; //'num2' is NOT accessible
    }
```

Variables

- **typeof**
  - Unary operator
  - Indicates the data type of the operand.
    - Eg: `val=123; alert(typeof(val));` // Displays 'number'
      `val="Hello"; alert(typeof(val));` // Displays 'string'
      `val=true ; alert(typeof(val));` // Displays 'boolean'
- **new**
  - Used for instantiation of objects.

    Eg: `val = new Date( ) ; alert(typeof(val));` // Displays 'object'

Variables

- JavaScript variables are containers for storing data values and it is loosely defined
- **Rules for defining variable names:**

  1.Names can contain letters, digits, underscores, and dollar signs.

  2.Names must begin with a letter,$ and _

  3.Names are case sensitive

  4.Keywords cannot be used as variables

Variables

Implicit Data Types

Infosys | Education, Training and Assessment

Implicit Data Types	Variables don't have explicit data types. The data type is automatically decided by the usage and hence implicit.

Implicit data types include:

- number
- string
- boolean
- object

Type Conversion

- Automatically converts between data types
- Eg:

```
val=123 ;          // Here 'val' is of number type
val="Hello";       // Here 'val' is of string type
val=true ;         // Here 'val' is of boolean type
```

Variables

- Prompt: window object's method, which displays the dialog. The argument to prompt specifies a message to user.
- parseInt: Function converts its string to an integer
- getElementById(): Method that identifies the HTML element with specified id
- .innerHTML: place the specified string value into the position denoted by getElementById() method
- OnLoad(): Event that occurs when an object has been loaded.

Variables

- Date(): Create a new object with current date and time.
- getHours(): Method returns the hour (from 0 to 23) of the specified date and time.

JS-Modes

JavaScript - Modes

- SCRIPT tag can be placed in HEAD or BODY tag
- Placing JavaScript in the HEAD tag ensures readability

Immediate	• Script gets executed as the page loads
Deferred	• Script gets executed based on user action

JS-Modes

JavaScript – 3 Ways of Inclusion in HTML(1 of 3)

- 3 ways in which JavaScript can be included in HTML
  - Inline
  - Embedded
  - External file
- Inline JavaScript
  - Scripts are included inside the HTML tag

```
<!DOCTYPE html>
<html>
<body onLoad="alert('document loaded')";>
<h2>Inline mode of java script</h2>
</body>
</html>
```

JS-Modes

JavaScript – 3 Ways of Inclusion in HTML (2 of 3)

- **Embedded**
  - Embedding JavaScript code into an html page is done using <script> tag
  - Embedded code is not accessible from other pages

```
<head>
    <title>Ist javascript</title>
    <script type = "text/javascript">
    document.writeln("<h1>Welcome to JavaScript Programming!</h1>" );
    </script>

</head>
```

JS-Modes

JavaScript – 3 Ways of Inclusion in HTML (3 of 3)

- **External**
  - Can be achieved by using the SRC attribute of the <script> tag.
  - External Javascript file should have an extension .js
  - Should not use <script> tag inside the .js file

external.js

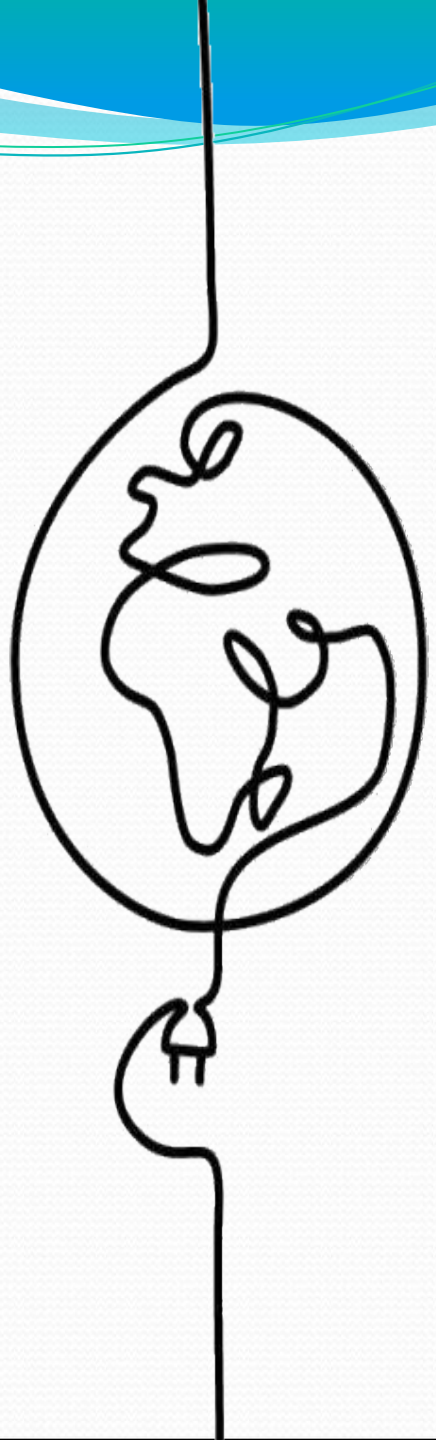
```
document.write("This is External scripting");
```

externaljs.html

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

</head>
```

Operators

...



Operators

Operators

Arithmetic Operators	• Unary : ++, -- • Binary : +, -, *, /, %
Relational Operators	• ==, !=, >, >=, < , <= • ===(Strict equal), !== (Strict not equal)
Logical Operators	• &&, || • !
Assignment Operators	• = • +=, -=, *=, /=, %=

Operators

- **Arithmetic operators:**

  Arithmetic operators are used to perform arithmetic on numbers

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	Increment
--	Decrement

Operators-Arithmetic operators

```
<script type = "text/javascript">
var first;
var second;
var n1,n2;
var sum,sub,mul,div;

first = window.prompt( "Enter 1st number" );
second = window.prompt( "Enter 2nd number" );
n1=parseInt(first);
n2=parseInt(second);
sum = n1+n2;
sub=n1-n2;
mul=n1*n2;
div=n1/n2;
document.write("Sum value is:"+sum);
document.write("Subtraction value is:"+sub);
document.write("Multiplication value is:"+mul);
document.write("Division value is:"+div);
</script>
```

Operators

- **Comparison operators:** Used in logical statements to determine equality or difference between variables

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operators

- **Logical operators:** Logical operators are used to determine the logic between variables or values.

Operator	Description
&&	logical and
||	logical or
!	logical not

- **String Operator:** The + operator can also be used to add (concatenate) strings

Operators

- **Bitwise operators:** Bit operators work on 32 bits numbers

Operator	Description
&	AND
|	OR
~	NOT
^	XOR
<<	Zero fill left shift
>>	Signed right shift
>>>	Zero fill right shift

Operators

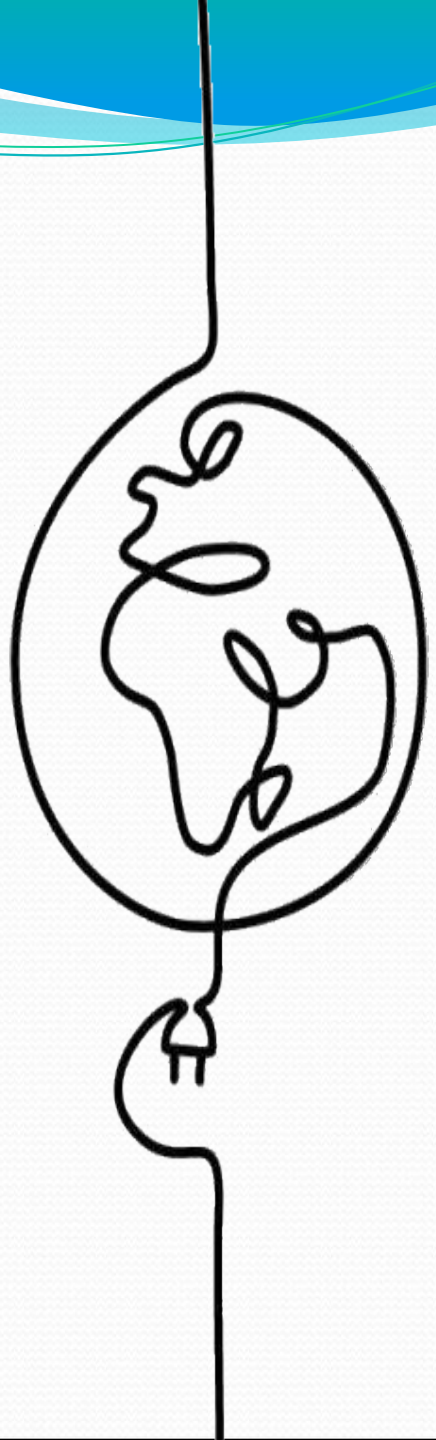
- **Condition (Ternary) operators:** The conditional operator assigns a value to a variable based on a condition
- Syntax:

variablename = (*condition*) ? *value1*:*value2*

- **Example**

voteable = (age < 18) ? "Too young":"Old enough";

Control Statements



Control statements

Control statements - Conditional

- Control structure in JavaScript is as follows:

if	• Is used to conditionally execute a single block of code
if...else	• A block of code is executed if the test condition evaluates to a boolean true; else another block of code is executed
switch	• Switch statement tests an expression against the case options • Executes the statements associated with the first match and break may be used to quit from the switch statement

Control statements

Control statements - Iteration (1 out of 2)

for loop

- Iterate through a block of statement for some particular range of values
- **Syntax :**

```
for(initialization ; condition ; increment/decrement) {
zero or more statements                                }
```

do while loop

- Block of statements is executed first and then condition is checked
- **Syntax :**

```
do {
      zero or more statements
}while ( test condition)
```

Control statements

Control statements - Iteration (2 out of 2)

while loop

- The while statement is used to execute a block of code while a certain condition is true
- **Syntax :**

```
while ( test condition) {
        zero or more statements
}
```

If-else Selection statements

- if

```
if (condition){
    //  block of code to be executed if the condition is true
  }
```

- If-else

```
if (condition) {
    //  block of code to be executed if the condition is true
  } else {
    //  block of code to be executed if the condition is false
  }
```

If-else Selection statements

- If-else-if

```
if (condition1) {
    //  if condition1 is true
  }
else if (condition2) {
    //  if the condition1 is false and condition2 is true
  }
else {
    //  if the condition1 is false and condition2 is false
  }
```

If-else Selection statements

```
<script type="text/javascript">
    var name;
    var now=new Date();
    var hour=now.getHours();
    name=window.prompt("Enter your name");

    document.writeln("Hour is:"+hour);
    if(hour<12)
        document.writeln("<h1>Hai\t"+name+"\tGood morning");
    else if(hour>=12)
    {
        hour=hour-12;
        if(hour<6)
        document.writeln("<h1>Hai\t"+name+"\tGood afternoon");
        else
        document.writeln("<h1>Hai\t"+name+"\tGood evening");
    }
</script>
```

Switch statement

- **Syntax**

```
switch(expression) {
    case x:
      // code block
      break;
    case y:
      // code block
      break;
    default:
      // code block
  }
```

Switch statement

```
<script>
    var d;
    d= window.prompt( "Enter 1 or 2" );
    n1=parseInt(d);
    switch (n1)
    {
    case 1:
    document.write("Entered 1");
    break;

    case 2:
    document.write("Entered 2");
    break;
    }
</script>
```

JS-Looping structure

- **Java script Loop**

Loops can execute a block of code a number of times

- JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

for/of - loops through the values of an iterable object

while - loops through a block of code while a specified condition is true

do/while - also loops through a block of code while a specified condition is true

For Loop

- **Syntax (For Loop)**

for (*statement 1*; *statement 2*; *statement 3*) {
  // code block to be executed
}

```
<script>
    var i;
    for (i = 0; i < 5; i++)
    {
        document.write("\t"+i);
    }
</script>
```

For Loop

0 1 2 3 4

For-in Loop

- JavaScript for/in statement loops through the properties of an object
- **Syntax:** for (*variable* in *object*) {
  // code block to be executed
  }

```
<script>
    var person = {fname:"A", lname:"B", age:25};
    for (x in person)
    {
        document.write("\t"+person[x]);
    }
</script>
```

For-in Loop

A B 25

For-of Loop

- JavaScript for/of statement loops through the values of an iterable objects

Syntax: for (*variable* of *iterable*) {
// code block to be executed
}

```
</script>

<h2>For-of Loop</h2>
<script>
    var txt = 'JavaScript';
    var x;
    for (x of txt)
    {
    document.write(x + "<br>");
    }
</script>
```

For-of Loop

J
a
v
a
S
c
r
i
p
t

While Loop

- The while loop loops through a block of code as long as a specified condition is true
- **Syntax:** while (*condition*) {
  // code block to be executed
  }

```
<script>
    var i=0;
    while(i<6)
    {
        document.write(i+"<br>");
        i++;
    }
</script>
```

While Loop

0
1
2
3
4
5

Do-while Loop

- **Syntax:** do {
  // code block to be executed
  }while (*condition*);

```
</script>

<h2>do-While Loop</h2>

<script>
    var i=0;
    do
    {
        document.write(i+"<br>");
        i++;
    }while(i<6);
</script>
```

do-While Loop

0
1
2
3
4
5

Break and continue

- The break statement "jumps out" of a loop.
- The continue statement "jumps over" one iteration in the loop.

```
<script>
    var i=0;
    while(i<6)
    {
        if(i==3)
        break;
        document.write(i+"<br>");
        i++;
    }
</script>
```

Break

0
1
2

Break and continue

Continue

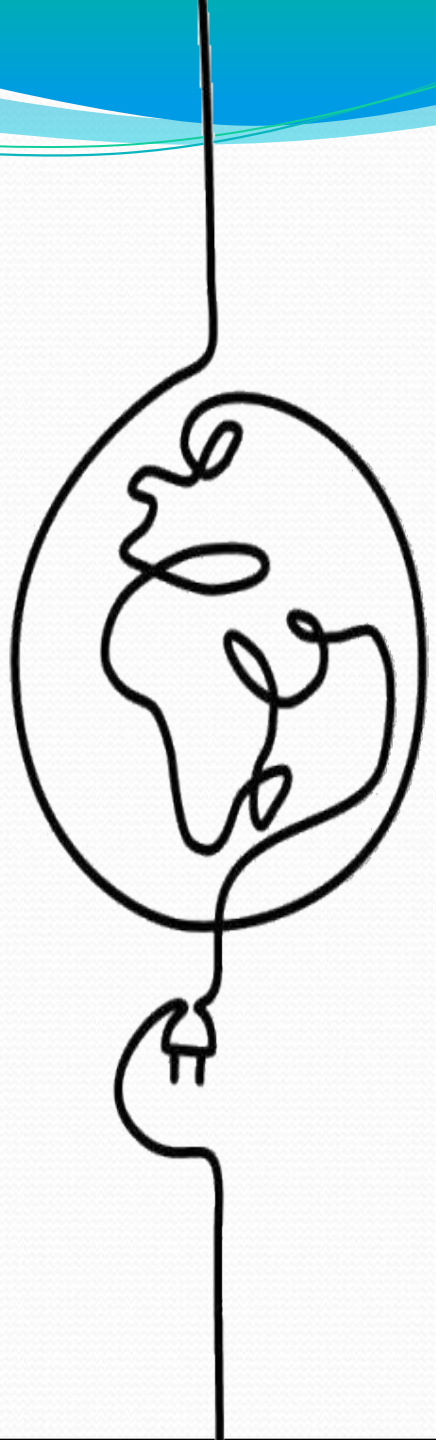
```
<script>

    for(i=0;i<6;i++)
    {
        if(i==3)
        continue;
        document.write(i+"<br>");
    }
</script>
```

do-While Loop

0
1
2
4
5

Functions

...



JS Functions

Functions

- A function is a block of code that has a name. It is a way to organize the code
- JavaScript has in-built functions. User can define his own functions(User defined functions)
- A function has a name, arguments(optional) and function body (statements). Arguments are local to the function

```
function myfunction(argument1,argument2,etc) {
    statements;
}
```

- JavaScript functions are used to link actions on a web page with the JavaScript code

Eg : JavaScript function can be used to link the user action like clicking on the 'Submit' button with the code validating the various fields of the form

Note: Built-in Functions in JavaScript will be revisited later in this course

JS Functions

- A JavaScript function is a block of code designed to perform a particular task.
- **Syntax**

function *name*(*parameter1, parameter2, parameter3*) {
  // code to be executed
}

- **Recursion:**

  A recursive function is a function that calls *itself, either directly, or indirectly through another function*

JS Functions

```
<script>
    function square(y)
    {
        return y*y;
    }
    for(var i=1;i<=5;i++)
    {
        document.writeln(square(i)+"<br>");
    }
</script>
```

Square of number

1
4
9
16
25

JS Functions

```
<script>
    function fact(x)
    {
        var f=1;
        for(var i=1;i<=x;i++)
        {
            f=f*i;
        }
        return f;
    }
    var a=window.prompt("Enter 1st number");
    n1=parseInt(a);
    document.writeln("The factorial is "+fact(n1));
</script>
```

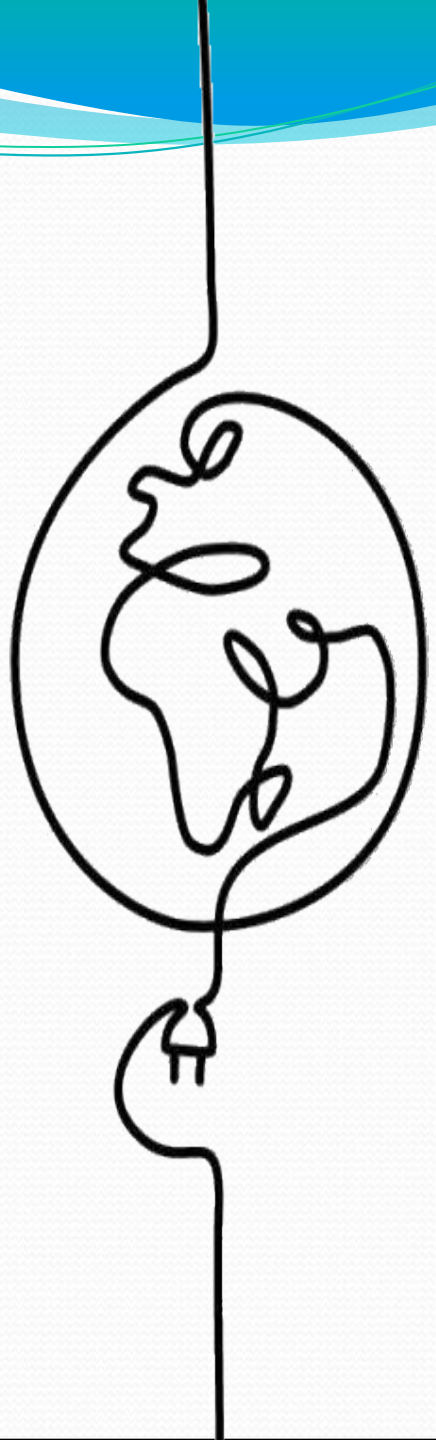
Factorial of number

The factorial is 120

JS Recursive Functions

```
function factorial(x)
{

  if (x === 0)
 {
    return 1;
 }
  return x * factorial(x-1);

}
```

Built-in Functions



Built-in Functions

Built-in functions – parseInt /parseFloat

- **parseInt / parseFloat** : Parses the given string and returns a numeric value if the given string starts with number
  - **Syntax : parseInt(str)**
  - Eg:

```
str= 123;    alert(parseInt(str));          // Displays 123
str= "123";  alert(parseInt(str));          // Displays 123
str= "abc";  alert(parseInt(str));          // Displays NaN
str= "123abc";  alert(parseInt(str));       // Displays 123
str= "abc123abc";  alert(parseInt(str));    // Displays NaN
```

*NaN stands for **N**ot **a** **N**umber

Built-in Functions

Built-in functions - isNaN

- **isNaN:** Evaluates an argument to determine if it is "NaN" (Not a Number)
  - **Syntax : isNaN(str)**
  - Eg:

```
str= "123";  alert(isNaN(str));          // Displays false
str= 123;  alert(isNaN(str));            // Displays false
str= "abc";  alert(isNaN(str));          // Displays true
str= abc;  alert(isNaN(str));            // Displays error
str= "123abc";  alert(isNaN(str));       // Displays true
str= "abc123abc";  alert(isNaN(str));    // Displays true
```

Built-in Functions

Built-in functions - isFinite

- **isFinite:** Evaluates an argument to determine whether it is a finite number
  - **Syntax : isFinite(str)**
  - Eg:

```
str= "123";  alert(isFinite(str));          // Displays true
str= 123;  alert(isFinite(str));            // Displays true
str= "abc";  alert(isFinite(str));          // Displays false
str= abc;  alert(isFinite(str));            // Displays error
str= "123abc";  alert(isFinite(str));       // Displays false
str= "abc123abc";  alert(isFinite(str));    // Displays false
```

Built-in Functions

Built-in functions – Number & String

- **Number:** Functions let you convert an object to number
  - **Syntax : Number(str)**
  - Eg:

```
str= "123";  alert(Number(str));          // Displays 123
str= 123;  alert(Number(str));            // Displays 123
str= abc;  alert(Number(str));            // Displays error
str= "abc";  alert(Number(str));          // Displays NaN
str= "123abc";  alert(isFinite(str));     // Displays NaN
str= "abc123abc";  alert(isFinite(str));  // Displays NaN
```

- **String:** Functions let you convert an object to string
  - **Syntax : String(num)**

Random Number generation

- Var randomValue=Math.random();
- random method generates a floating-point value from 0.0 up to, but not including, 1.0
- Random integers in a certain range can be generated by scaling and shifting the values returned by random, then using Math.floor to convert them to integers (Math.floor(1+Math.random()*6)
- The scaling factor determines the size of the range (i.e. a scaling factor of 6 means produces values from 0.0 to but not including 6.0)
- The shift number is added to the result to determine where the range begins (i.e. shifting the numbers by 1 would give numbers between 1 and 7)

Random Number generation

```
<!DOCTYPE html>

<!-- Fig. 9.4: RandomInt.html -->
<!-- Random integers, shifting and scaling. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Shifted and Scaled Random Integers</title>
      <style type = "text/css">
         p, ol { margin: 0; }
         li    { display: inline; margin-right: 10px; }
      </style>
      <script>

         var value;

         document.writeln( "<p>Random Numbers</p><ol>" );

         for ( var i = 1; i <= 30; ++i )
         {
            value = Math.floor( 1 + Math.random() * 6 );
            document.writeln( "<li>" + value + "</li>" );
         } // end for
```

Random Number generation

```
25            document.writeln( "</ol>" );
26
27         </script>
28      </head><body></body>
29   </html>
```

Displaying Random Images

- Build a random image generator—a script that displays four randomly selected die images every time the user clicks a Roll Dice button on the page.

Displaying Random Images

```
<!DOCTYPE html>

<!-- Fig. 9.5: RollDice.html -->
<!-- Random dice image generation using Math.random. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Random Dice Images</title>
      <style type = "text/css">
         li { display: inline; margin-right: 10px; }
         ul { margin: 0; }
      </style>
      <script>
         // variables used to interact with the i mg elements
         var die1Image;
         var die2Image;
         var die3Image;
         var die4Image;
```

Displaying Random Images

```
// register button listener and get the img elements
function start()
{
   var button = document.getElementById( "rollButton" );
   button.addEventListener( "click", rollDice, false );
   die1Image = document.getElementById( "die1" );
   die2Image = document.getElementById( "die2" );
   die3Image = document.getElementById( "die3" );
   die4Image = document.getElementById( "die4" );
} // end function rollDice

// roll the dice
function rollDice()
{
   setImage( die1Image );
   setImage( die2Image );
   setImage( die3Image );
   setImage( die4Image );
} // end function rollDice
```

Displaying Random Images

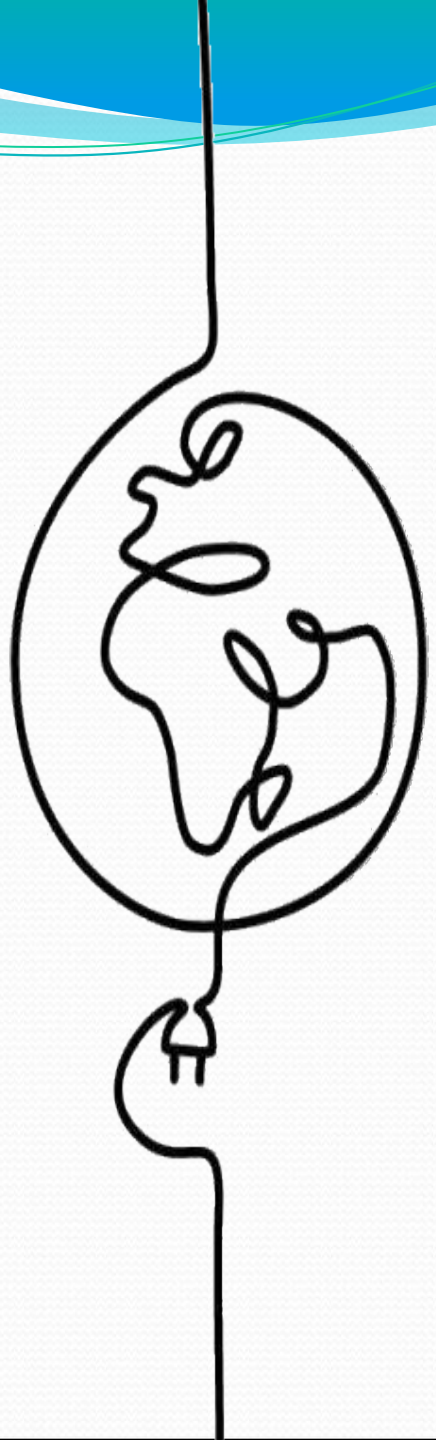
```
      // set image source for a die
      function setImage( dieImg )
      {
         var dieValue = Math.floor( 1 + Math.random() * 6 );
         dieImg.setAttribute( "src", "die" + dieValue + ".png" );
         dieImg.setAttribute( "alt",
            "die image with " + dieValue + " spot(s)" );
      } // end function setImage

      window.addEventListener( "load", start, false );
   </script>
</head>
```

Displaying Random Images

```
52      <body>
53         <form action = "#">
54            <input id = "rollButton" type = "button" value = "Roll Dice">
55         </form>
56         <ol>
57            <li><img id = "die1" src = "blank.png" alt = "die 1 image"></li>
58            <li><img id = "die2" src = "blank.png" alt = "die 2 image"></li>
59            <li><img id = "die3" src = "blank.png" alt = "die 3 image"></li>
60            <li><img id = "die4" src = "blank.png" alt = "die 4 image"></li>
61         </ol>
62      </body>
63   </html>
```

Scope

...



JS-Scope

- Scope determines the accessibility (visibility) of variables
- In JavaScript there are two types of scope

  Local scope

  Global scope
- **Local Scope:**

  Variables declared within a JavaScript function, become **LOCAL** to the function

JS-Scope

- **Example:**

```
<script>
    function sample()
    {
        var i=10;
        document.write("Inside fucntion"+i);
    }
    sample();
    document.write("Outside function"+i);
</script>
```

Local scope

Inside fucntion10

JS-Scope

- **Global Scope:**

A variable declared outside a function, becomes **GLOBAL**

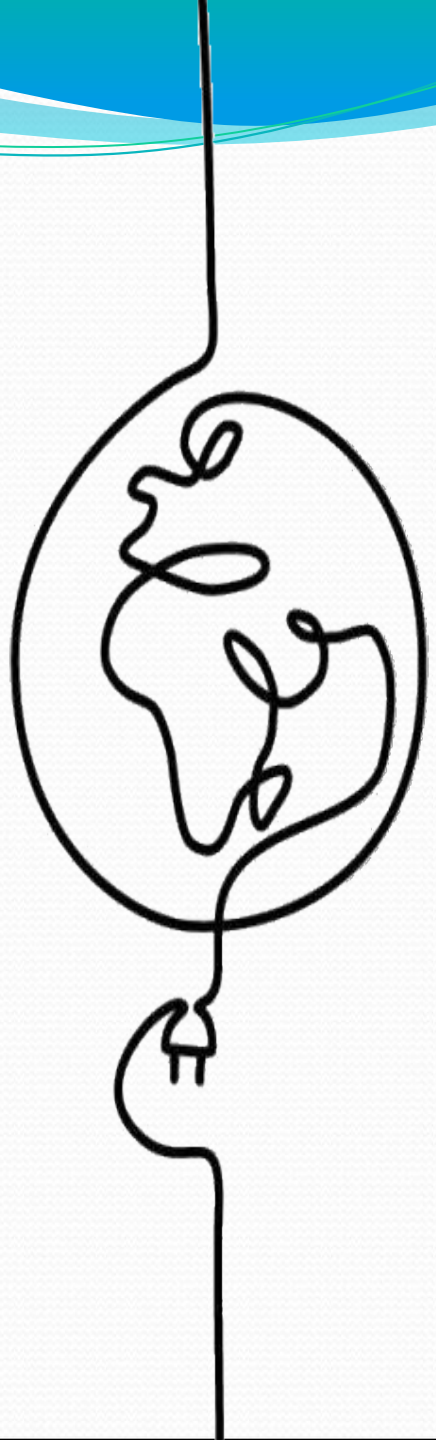
```
<script>
    var i=10;
    function sample()
    {
        document.write("Inside fucntion "+i+"<br>");
    }
    sample();
    document.write("Outside function "+i+"<br>");
</script>
```

Global scope

Inside fucntion 10
Outside function 10

Array

...



Array

- An array is a group of memory locations that all have the same name and normally are of the same type
- **Arrays** are data structures consisting of related data items.
- JavaScript arrays are “dynamic” entities in that they can change size after they’re created.

Declaring and Allocating Arrays

- an array in JavaScript is an **Array object**.
- **new operator** to create an array and to specify the number of elements in an array.
- To allocate 12 elements for integer array c:

```
var c = new Array( 12 );
```

[OR]

```
var c; // declares a variable that will hold the array
c = new Array( 12 ); // allocates the array
```

Different Ways of Initializing a List

- **var** n = [**10**, **20**, **30**, **40**, **50**];
- **var** n = **new** Array(**10**, **20**, **30**, **40**, **50**);
- **var** n = [**10**, **20**, , **40**, **50**];

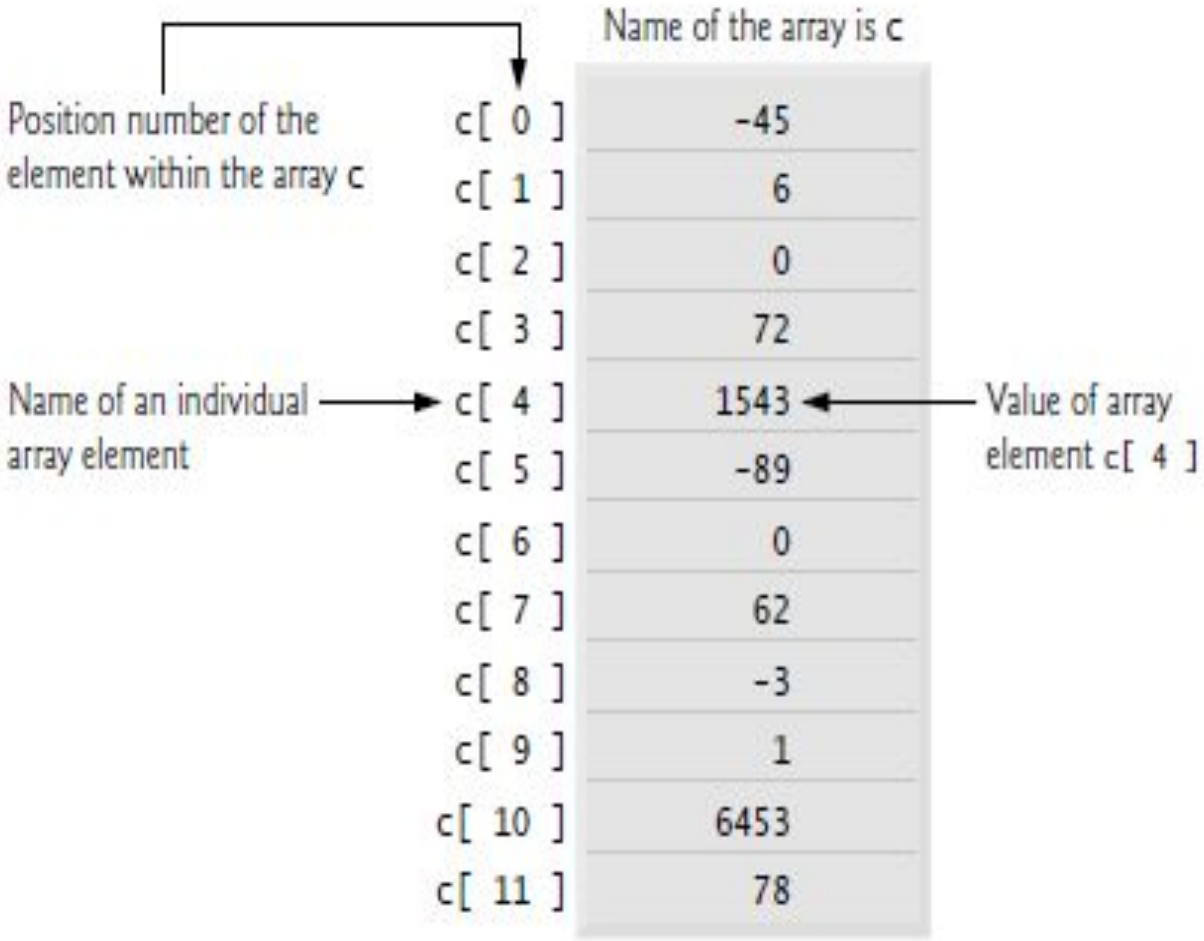


Fig. 10.1 | Array with 12 elements.

Example: Array initialization, Display

```
<html>
<head>
<script src="arrayscript.js">
</script>
</head>
<body onload="start()">
<div id ="outone"> </div>
<div id ="outtwo"> </div>
</body>
</html>
```

```
function start()
{
    var a = new Array(5);
    var b = new Array();
    var len = a.length;
    for ( var i=0;i<len;++i)
    {
        a[i]=i;
        b[i]=i+2;
    }
    dis(a,b);
}
```

Example: Array initialization, Display

```
function dis(a,b)
{
    content1="<h1>A Array</h1> <br>";
    content2="<h1>B Array</h1> <br>";
    for(i=0;i<a.length;i++)
    {
        content1+=a[i]+"<br>";
        content2+=b[i]+"<br>";
    }
    document.getElementById("outone").innerHTML=content1;
    document.getElementById("outtwo").innerHTML=content2;

}
```

A Array

0
1
2
3
4

B Array

2
3
4
5
6

Example: Array modification

```
<script type = "text/javascript">
function start()
{
    var a = [10,20,30,40,50];
    c="Original Array <br>"
    for(i=0;i<a.length;i++)
        c+=a[i]+"\t";
    document.getElementById("original").innerHTML= c;
    modifyarray(a);
    c="Modified Array <br>"
    for(i=0;i<a.length;i++)
        c+=a[i]+"\t";
    document.getElementById("modified").innerHTML= c;

    modifyvalue(a[2]);
    document.getElementById("afterfunc").innerHTML= "a[2] after func:"+a[
}
```

Example: Array modification

```
function modifyvalue(t)
{
    t*=2;
    document.getElementById("insidefunc").innerHTML="a[2] inside func:"+t;
}
</script>
</head>

<body onload="start()">
<div id="original"> </div>
<h2>Effects of passing entire array by reference</h2>
<div id="modified"> </div>
<h2>Effects of passing array element by value</h2>
 <div id = "insidefunc"></div>
 <div id = "afterfunc"></div>
</body>
```

Example: Array modification

Original Array
10 20 30 40 50

Effects of passing entire array by reference

Modified Array
20 40 60 80 100

Effects of passing array element by value

a[2] inside func:120
a[2] after func:60

Note:

Passing array by **reference**, changes value inside function **reflects** in original array

Passing array by **value,** changes value inside function **will not reflect** in original array

Random Image Generation

```
<!DOCTYPE html>

<!-- Fig. 10.11: RandomPicture.html -->
<!-- HTML5 document that displays randomly selected images. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Random Image Generator</title>
      <script src = "RandomPicture.js"></script>
   </head>
   <body>
      <img id = "image" src = "CPE.png" alt = "Common Programming Error">
   </body>
</html>
```

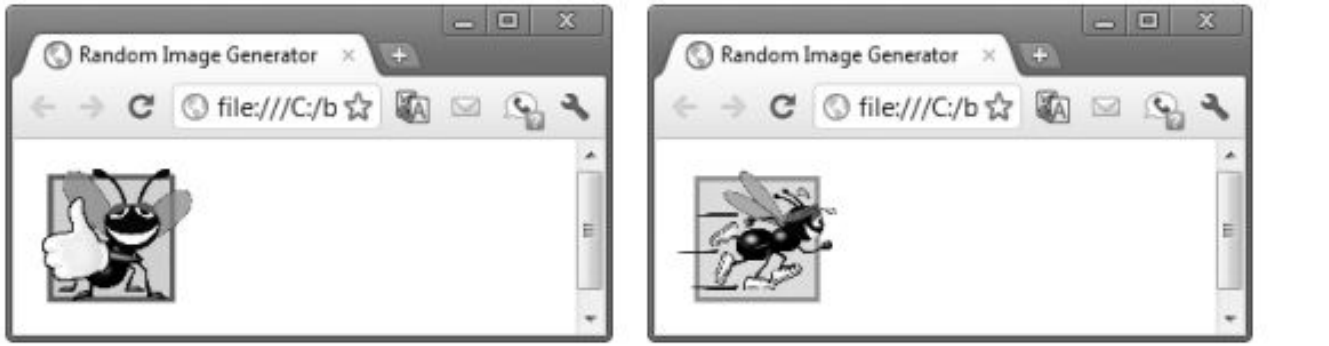
Random Image Generation

```
// Fig. 10.12: RandomPicture2.js
// Random image selection using arrays
var iconImg;
var pictures = [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
```

10.12 | Random image selection using arrays. (Part 1 of 2.)


```
var descriptions = [ "Common Programming Error",
   "Error-Prevention Tip", "Good Programming Practice",
   "Look-and-Feel Observation", "Performance Tip", "Portability Tip",
   "Software Engineering Observation" ];

// pick a random image and corresponding description, then modify
// the img element in the document's body
function pickImage()
{
   var index = Math.floor( Math.random() * 7 );
   iconImg.setAttribute( "src", pictures[ index ] + ".png" );
   iconImg.setAttribute( "alt", descriptions[ index ] );
} // end function pickImage

// registers iconImg's click event handler
function start()
{
   iconImg = document.getElementById( "iconImg" );
   iconImg.addEventListener( "click", pickImage, false );
} // end function start

window.addEventListener( "load", start, false );
```

Random Image Generation



Sorting Arrays with Array Method sort

```
<!DOCTYPE html>

<!-- Fig. 10.15: Sort.html -->
<!-- HTML5 document that displays the results of sorting an array. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Array Method sort</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "Sort.js"></script>
   </head>
   <body>
      <h1>Sorting an Array</h1>
      <p id = "originalArray"></p>
      <p id = "sortedArray"></p>
   </body>
</html>
```

```
// Fig. 10.16: Sort.js
// Sorting an array with sort.
function start()
{
   var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];

   outputArray( "Data items in original order: ", a,
      document.getElementById( "originalArray" ) );
   a.sort( compareIntegers );  // sort the array
   outputArray( "Data items in ascending order: ", a,
      document.getElementById( "sortedArray" ) );
} // end function start

// output the heading followed by the contents of theArray
function outputArray( heading, theArray, output )
{
   output.innerHTML = heading + theArray.join( " " );
} // end function outputArray

// comparison function for use with sort
function compareIntegers( value1, value2 )
{
   return parseInt( value1 ) - parseInt( value2 );
} // end function compareIntegers

window.addEventListener( "load", start, false );
```

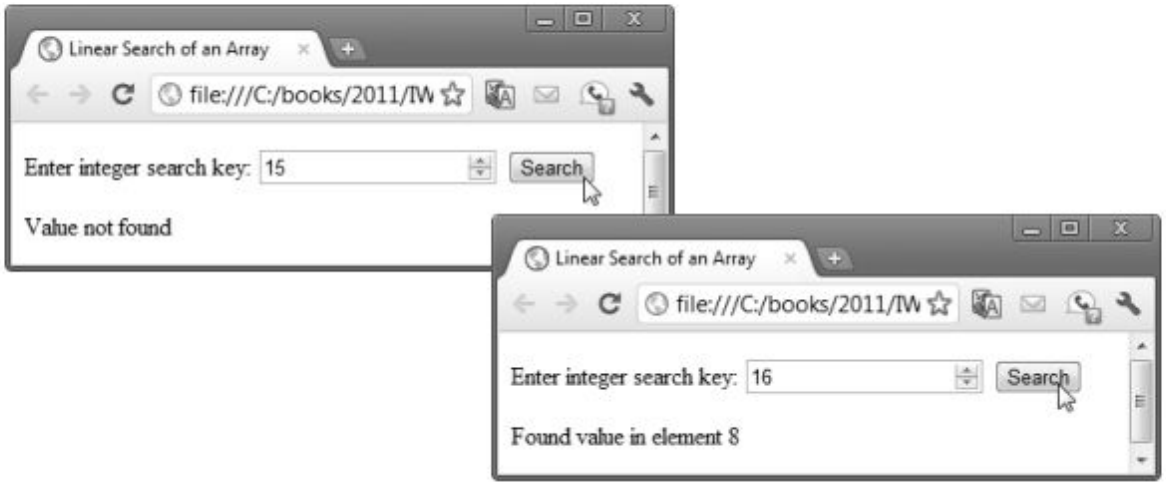

Array Method sort
file:///C:/b
Sorting an Array
Data items in original order: 10 1 9 2 8 3 7 4 6 5
Data items in ascending order: 1 2 3 4 5 6 7 8 9 10

Searching Arrays with Array Method indexOf

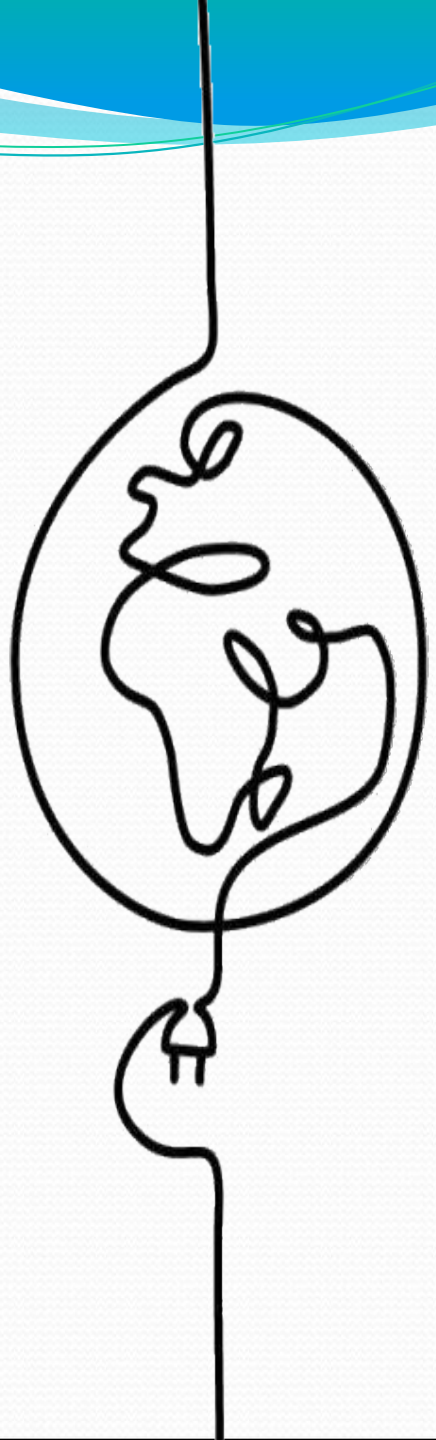
- The Array object in JavaScript has built-in methods indexOf and lastIndexOf for searching arrays.

```
<!DOCTYPE html>

<!-- Fig. 10.17: search.html -->
<!-- HTML5 document for searching an array with indexOf. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search an Array</title>
      <script src = "search.js"></script>
   </head>
   <body>
      <form action  = "#">
         <p><label>Enter integer search key:
            <input id = "inputVal" type = "number"></label>
            <input id = "searchButton" type = "button" value = "Search">
         </p>
         <p id = "result"></p>
      </form>
   </body>
</html>
```

```
// Fig. 10.18: search.js
// Search an array with indexOf.
var a = new Array( 100 );  // create an array

// fill array with even integer values from 0 to 198
for ( var i = 0; i < a.length; ++i )
{
   a[ i ] = 2 * i;
} // end for

// function called when "Search" button is pressed
function buttonPressed()
{
   // get the input text field
   var inputVal = document.getElementById( "inputVal" );

   // get the result paragraph
   var result = document.getElementById( "result" );

   // get the search key from the input text field then perform the search
   var searchKey = parseInt( inputVal.value );
   var element = a.indexOf( searchKey );

   if ( element != -1 )
   {
      result.innerHTML = "Found value in element " + element;
   } // end if
   else
   {
      result.innerHTML = "Value not found";
   } // end else
} // end function buttonPressed

// register searchButton's click event handler
function start()
{
   var searchButton = document.getElementById( "searchButton" );
   searchButton.addEventListener( "click", buttonPressed, false );
} // end function start

window.addEventListener( "load", start, false );
```


Linear Search of an Array
file:///C:/books/2011/IW
Enter integer search key: 15
Search
Value not found
Linear Search of an Array
file:///C:/books/2011/IW
Enter integer search key: 16
Search
Found value in element 8

Objects



Objects

- JavaScript uses objects to perform many tasks
- It is referred to as an object-based programming language
- Objects have attributes and exhibit behaviors
- Objects have the property of information hiding
- Objects may know how to communicate with one another across well-defined interfaces, but normally they are not allowed to know how other objects are implemented
- Web browsers
- Contain a set of objects that encapsulate an XHTML document's elements
- The objects expose to a JavaScript programmer the attributes and behaviors that enable a JavaScript program to interact with (or script) these elements (objects).

Math Object

- Math object methods allow you to perform many common mathematical calculations.
- An object's methods are called by writing the name of the object followed by a dot operator (.) and the name of the method
- In parentheses following the method name is the argument (or a comma-separated list of arguments) to the method
- var result = Math.sqrt(900);
- Answer:30

Method	Description	Examples
`abs( x )`	Absolute value of x.	`abs( 7.2 )` is 7.2 `abs( 0 )` is 0 `abs( -5.6 )` is 5.6
`ceil( x )`	Rounds x to the smallest integer not less than x.	`ceil( 9.2 )` is 10 `ceil( -9.8 )` is -9.0
`cos( x )`	Trigonometric cosine of x (x in radians).	`cos( 0 )` is 1
`exp( x )`	Exponential method e^x.	`exp( 1 )` is 2.71828 `exp( 2 )` is 7.38906
`floor( x )`	Rounds x to the largest integer not greater than x.	`floor( 9.2 )` is 9 `floor( -9.8 )` is -10.0
`log( x )`	Natural logarithm of x (base e).	`log( 2.718282 )` is 1 `log( 7.389056 )` is 2
`max( x, y )`	Larger value of x and y.	`max( 2.3, 12.7 )` is 12.7 `max( -2.3, -12.7 )` is -2.3
`min( x, y )`	Smaller value of x and y.	`min( 2.3, 12.7 )` is 2.3 `min( -2.3, -12.7 )` is -12.7
`pow( x, y )`	x raised to power y (x^y).	`pow( 2, 7 )` is 128 `pow( 9, .5 )` is 3.0
`round( x )`	Rounds x to the closest integer.	`round( 9.75 )` is 10 `round( 9.25 )` is 9
`sin( x )`	Trigonometric sine of x (x in radians).	`sin( 0 )` is 0
`sqrt( x )`	Square root of x.	`sqrt( 900 )` is 30 `sqrt( 9 )` is 3
`tan( x )`	Trigonometric tangent of x (x in radians).	`tan( 0 )` is 0

Math Constants

Constant	Description	Value
`Math.E`	Base of a natural logarithm (*e*).	Approximately 2.718
`Math.LN2`	Natural logarithm of 2.	Approximately 0.693
`Math.LN10`	Natural logarithm of 10.	Approximately 2.302
`Math.LOG2E`	Base 2 logarithm of *e*.	Approximately 1.442
`Math.LOG10E`	Base 10 logarithm of *e*.	Approximately 0.434
`Math.PI`	π—the ratio of a circle's circumference to its diameter.	Approximately 3.1415926535897 93
`Math.SQRT1_2`	Square root of 0.5.	Approximately 0.707
`Math.SQRT2`	Square root of 2.0.	Approximately 1.414

String Object

- Characters are the fundamental building blocks of JavaScript programs
- Every program is composed of a sequence of characters grouped together meaningfully that is interpreted by the computer as a series of instructions used to accomplish a task
- A string is a series of characters treated as a single unit
- A string may include letters, digits and various special characters, such as +, -, *, /, and $
- JavaScript supports Unicode, which represents a large portion of the world's languages
- String literals or string constants (often called anonymous String objects) are written as a sequence of characters in double quotation marks or single quotation marks

String object-Examples

"John Q. Doe" (a name)
'9999 Main Street' (a street address)
"Waltham, Massachusetts" (a city and state)
'(201) 555-1212' (a telephone number)

A String may be assigned to a variable in a declaration.

```
var color = "blue";
```

Methods of the String Object

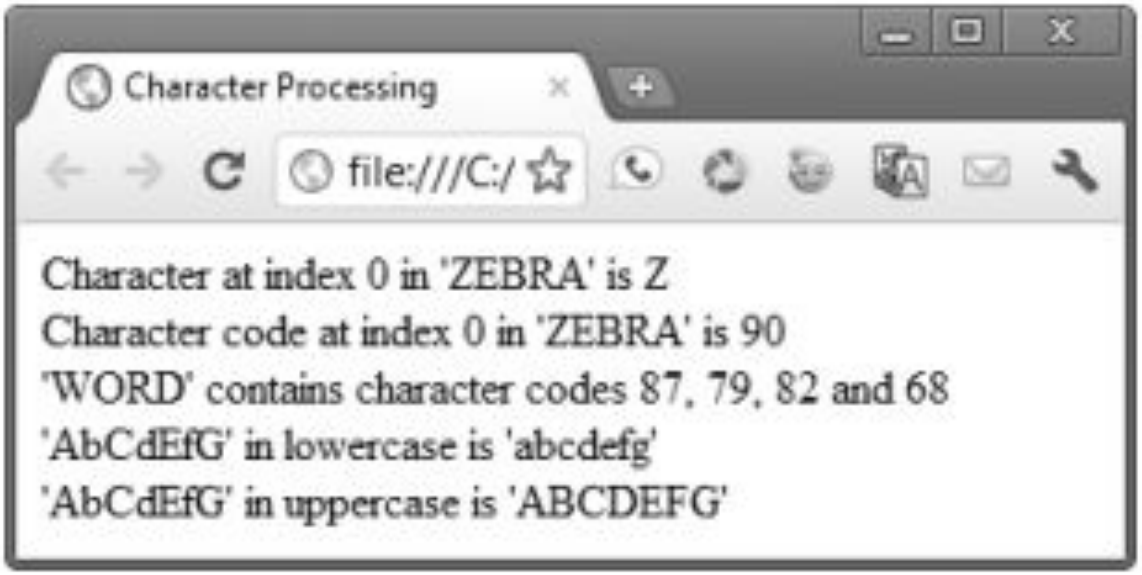
Method	Description
`charCodeAt( index )`	Returns the Unicode value of the character at the specified *index*, or `NaN` (not a number) if there's no character at that *index*.
`concat( string )`	Concatenates its argument to the end of the string on which the method is invoked. The original string is not modified; instead a new `String` is returned. This method is the same as adding two strings with the string-concatenation operator + (e.g., `s1.concat(s2)` is the same as `s1 + s2`).
`fromCharCode( value1, value2, ...)`	Converts a list of Unicode values into a string containing the corresponding characters.
`indexOf( substring, index )`	Searches for the *first* occurrence of *substring* starting from position *index* in the string that invokes the method. The method returns the starting index of *substring* in the source string or −1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from index 0 in the source string.
`lastIndexOf( substring, index )`	Searches for the *last* occurrence of *substring* starting from position *index* and searching toward the beginning of the string that invokes the method. The method returns the starting index of *substring* in the source string or −1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from the *end* of the source string.
`replace( searchString, replaceString )`	Searches for the substring *searchString*, replaces the first occurrence with *replaceString* and returns the modified string, or returns the original string if no replacement was made.
`slice( start, end )`	Returns a string containing the portion of the string from index *start* through index *end*. If the *end* index is not specified, the method returns a string from the *start* index to the end of the source string. A negative *end* index specifies an offset from the end of the string, starting from a position one past the end of the last character (so −1 indicates the last character position in the string).
`split( string )`	Splits the source string into an array of strings (tokens), where its *string* argument specifies the delimiter (i.e., the characters that indicate the end of each token in the source string).

Character-Processing Methods

- charAt—returns the character at a specific position
- charCodeAt—returns the Unicode value of the character at a specific position
- fromCharCode—returns a string created from a series of Unicode values
- toLowerCase—returns the lowercase version of a string
- toUpperCase—returns the uppercase version of a string

```
<!DOCTYPE html>
 <!-- HTML5 document to demonstrate String methods charAt, charCodeAt,fromCharCode, toLowercase and toUpperCase. -->
<html>
 <head>
<meta charset = "utf-8">
<title>Character Processing</title>
<link rel = "stylesheet" type = "text/css" href ="style.css">
<script src = "CharacterProcessing.js"></script>
</head>
<body>
<div id = "results"></div>
 </body>
```

```
// Fig. 11.5: CharacterProcessing.js
// String methods charAt, charCodeAt, fromCharCode,
// toLowercase and toUpperCase.
function start()
{
   var s = "ZEBRA";
   var s2 = "AbCdEfG";
   var result = "";

   result = "<p>Character at index 0 in '" + s + "' is " +
      s.charAt( 0 ) + "</p>";
   result += "<p>Character code at index 0 in '" + s + "' is " +
      s.charCodeAt( 0 ) + "</p>";

   result += "<p>'" + String.fromCharCode( 87, 79, 82, 68 ) +
      "' contains character codes 87, 79, 82 and 68</p>";

   result += "<p>'" + s2 + "' in lowercase is '" +
      s2.toLowerCase() + "'</p>";
   result += "<p>'" + s2 + "' in uppercase is '" +
      s2.toUpperCase() + "'</p>";

   document.getElementById( "results" ).innerHTML = result;
} // end function start

window.addEventListener( "load", start, false );
```


Character Processing
file:///C:/
Character at index 0 in 'ZEBRA' is Z
Character code at index 0 in 'ZEBRA' is 90
'WORD' contains character codes 87, 79, 82 and 68
'AbCdEfG' in lowercase is 'abcdefg'
'AbCdEfG' in uppercase is 'ABCDEFG'

telerik

The Date Object

- The Date object provides date / calendar functions

dates.html

```
var now = new Date();
var result = "It is now " + now;
document.getElementById("timeField")
  .innerText = result;
...
<p id="timeField"></p>
```

telerik

Timers: setTimeout()

- Make something happen (once) after a fixed delay

```
var timer = setTimeout('bang()', 5000);
```

5 seconds after this statement executes, this function is called

```
clearTimeout(timer);
```

Cancels the timer

telerik

Timers: setInterval()

- Make something happen repeatedly at fixed intervals

```
var timer = setInterval('clock()', 1000);
```

This function is called continuously per 1 second.

```
clearInterval(timer);
```

Stop the timer.

telerik

Timer – Example

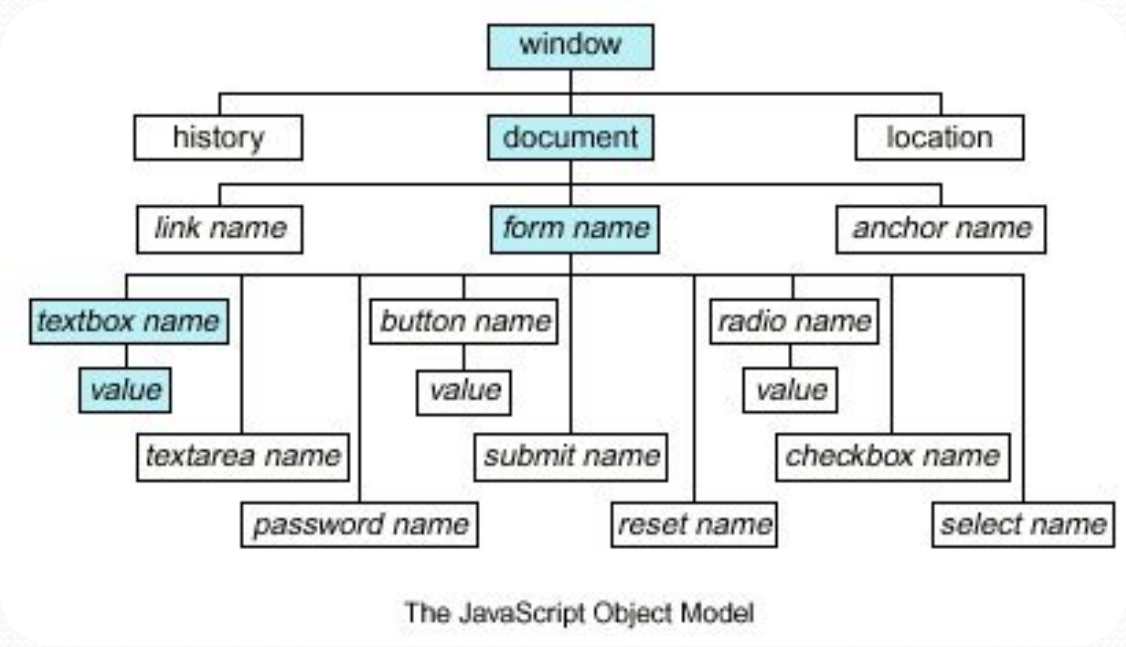
timer-demo.html

```
<script type="text/javascript">
  function timerFunc() {
    var now = new Date();
    var hour = now.getHours();
    var min = now.getMinutes();
    var sec = now.getSeconds();
    document.getElementById("clock").value =
      "" + hour + ":" + min + ":" + sec;
  }
  setInterval('timerFunc()', 1000);
</script>
<input type="text" id="clock" />
```

Boolean and Number Objects

- JavaScript provides the Boolean and Number objects as object wrappers for boolean true/ false values and numbers, respectively.
- var b = new Boolean(booleanValue);
- toString() Returns the string "true" if the value of the Boolean object is true; otherwise, returns the string "false".
- valueOf() Returns the value true if the Boolean object is true; otherwise, returns false.

Document Object Model (DOM)



The JavaScript Object Model

Document Object Model (DOM)

- Every HTML element is accessible via the JavaScript DOM API
- Most DOM objects can be manipulated by the programmer
- The event model lets a document to react when the user does something on the page
- Advantages
  - Create interactive pages
  - Updates the objects of a page without reloading it

Accessing Elements

- Access elements via their ID attribute

```
var elem = document.getElementById("some_id")
```

- Via the name attribute

```
var arr = document.getElementsByName("some_name")
```

- Via tag name
  - Returns array of descendant `<img>` elements of the element "`el`"

```
var imgTags = el.getElementsByTagName("img")
```

DOM Manipulation

- Once we access an element, we can read and write its attributes

DOM-manipulation.html

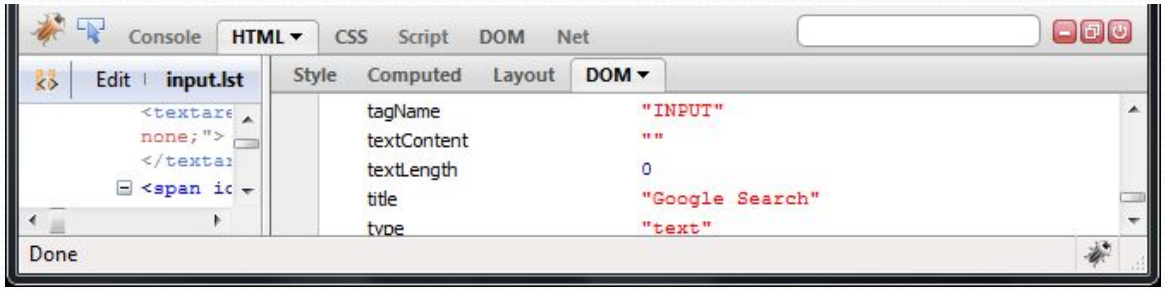
```
function change(state) {
  var lampImg = document.getElementById("lamp");
  lampImg.src = "lamp_" + state + ".png";
  var statusDiv =
    document.getElementById("statusDiv");
  statusDiv.innerHTML = "The lamp is " + state";
}
…
<img src="test_on.gif" onmouseover="change('off')"
  onmouseout="change('on')" />
```

Common Element Properties

- Most of the properties are derived from the HTML attributes of the tag
  - E.g. `id`, `name`, `href`, `alt`, `title`, `src`, etc…
- `style` property – allows modifying the CSS styles of the element
  - Corresponds to the inline style of the element
    - Not the properties derived from embedded or external CSS rules
  - Example: `style.width`, `style.marginTop`, `style.backgroundImage`

Common Element Properties (2)

- `className` – the `class` attribute of the tag
- `innerHTML` – holds all the entire HTML code inside the element
- Read-only properties with information for the current element and its state
  - `tagName`, `offsetWidth`, `offsetHeight`, `scrollHeight`, `scrollTop`, `nodeType`, etc…



Accessing Elements through the DOM Tree Structure

- We can access elements in the DOM through some tree manipulation properties:
  - `element.childNodes`
  - `element.parentNode`
  - `element.nextSibling`
  - `element.previousSibling`
  - `element.firstChild`
  - `element.lastChild`

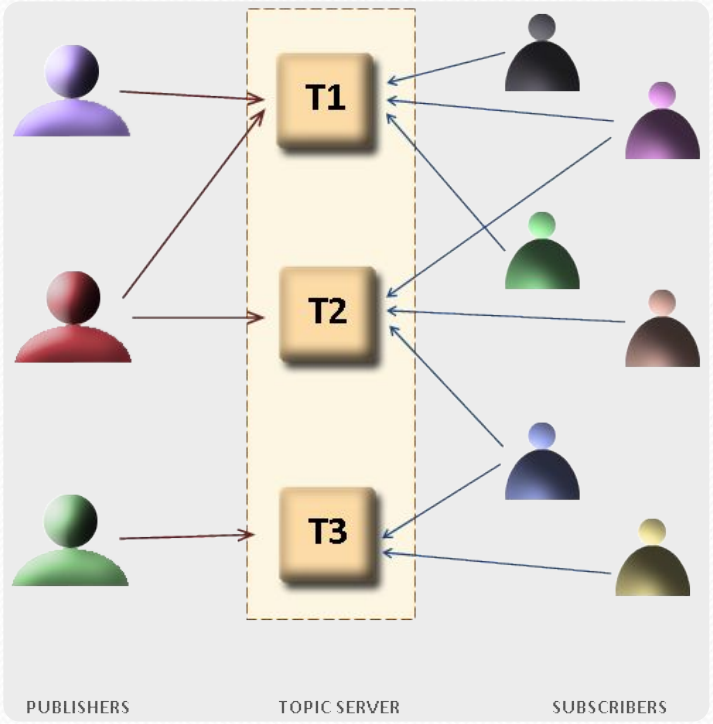
Accessing Elements through the DOM Tree – Example

accessing-elements-demo.html

```
var el = document.getElementById('div_tag');
alert (el.childNodes[0].value);
alert (el.childNodes[1].
  getElementsByTagName('span').id);
…
<div id="div_tag">
  <input type="text" value="test text" />
  <div>
    <span id="test">test span</span>
  </div>
</div>
```

- Warning: may not return what you expected due to Browser differences

The HTML DOM Event Model



The HTML DOM Event Model

- JavaScript can register event handlers
  - Events are fired by the Browser and are sent to the specified JavaScript event handler function
  - Can be set with HTML attributes:

```
<img src="test.gif" onclick="imageClicked()" />
```

  - Can be accessed through the DOM:

```
var img = document.getElementById("myImage");
img.onclick = imageClicked;
```

The HTML DOM Event Model (2)

- All event handlers receive one parameter
  - It brings information about the event
  - Contains the type of the event (mouse click, key press, etc.)
  - Data about the location where the event has been fired (e.g. mouse coordinates)
  - Holds a reference to the event sender
    - E.g. the button that was clicked

The HTML DOM Event Model (3)

- Holds information about the state of [Alt], [Ctrl] and [Shift] keys
- Some browsers do not send this object, but place it in the `document.event`
- Some of the names of the event's object properties are browser-specific

Common DOM Events

- Mouse events:
  - `onclick`, `onmousedown`, `onmouseup`
  - `onmouseover`, `onmouseout`, `onmousemove`
- Key events:
  - `onkeypress`, `onkeydown`, `onkeyup`
  - Only for input fields
- Interface events:
  - `onblur`, `onfocus`
  - `onscroll`

Common DOM Events (2)

- Form events
  - `onchange` – for input fields
  - `onsubmit`
    - Allows you to cancel a form submission
    - Useful for form validation
- Miscellaneous events
  - `onload`, `onunload`
    - Allowed only for the `<body>` element
    - Fires when all content on the page was loaded / unloaded

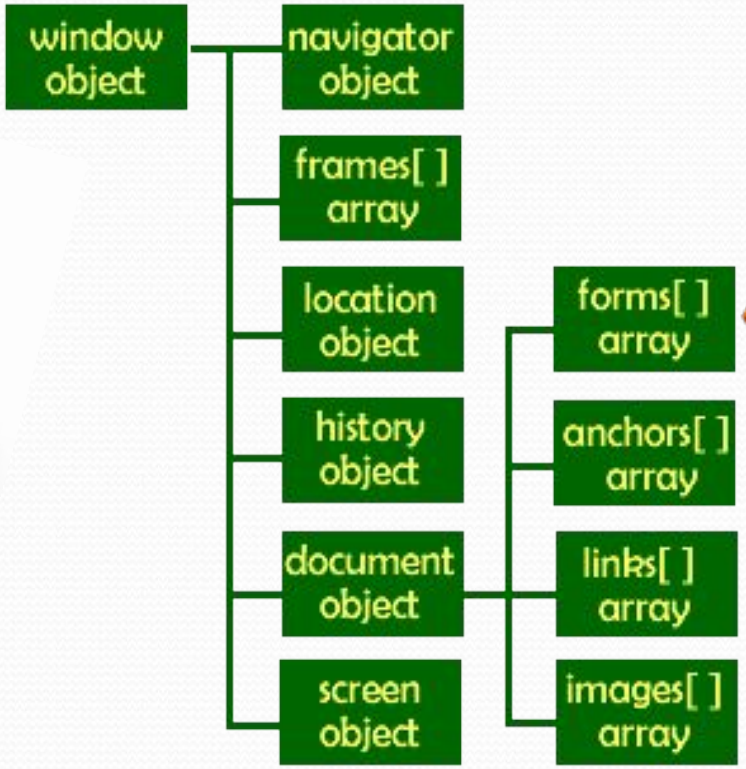
onload Event – Example

onload.html

- onload event

```
<html>

<head>
  <script type="text/javascript">
    function greet() {
      alert("Loaded.");
    }
  </script>
</head>

<body onload="greet()" >
</body>

</html>
```

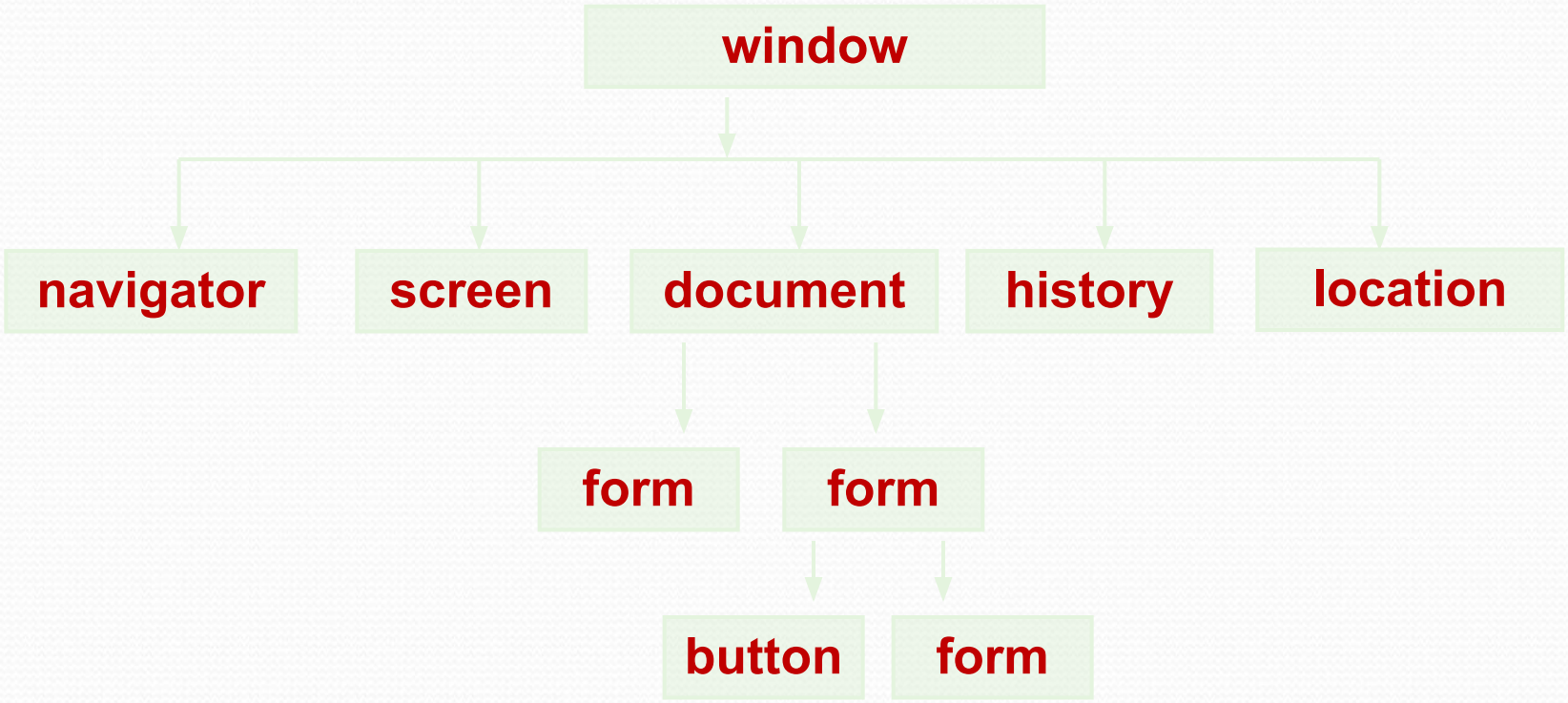
The Built-In Browser Objects



Built-in Browser Objects

- The browser provides some read-only data via:
  - `window`
    - The top node of the DOM tree
    - Represents the browser's window
  - `document`
    - holds information the current loaded document
  - `screen`
    - Holds the user's display properties
  - `browser`
    - Holds information about the browser

DOM Hierarchy – Example

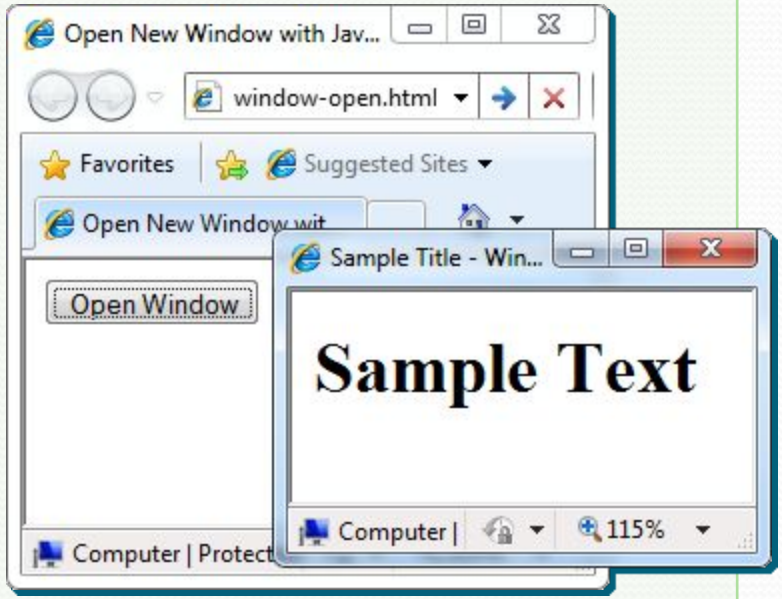


Opening New Window – Example

window-open.html

- window.open()

```
var newWindow = window.open("", "sampleWindow",
  "width=300, height=100, menubar=yes,
  status=yes, resizable=yes");

newWindow.document.write(
  "<html><head><title>
  Sample Title</title>
  </head><body><h1>Sample
  Text</h1></body>");
newWindow.status =
  "Hello folks";
```



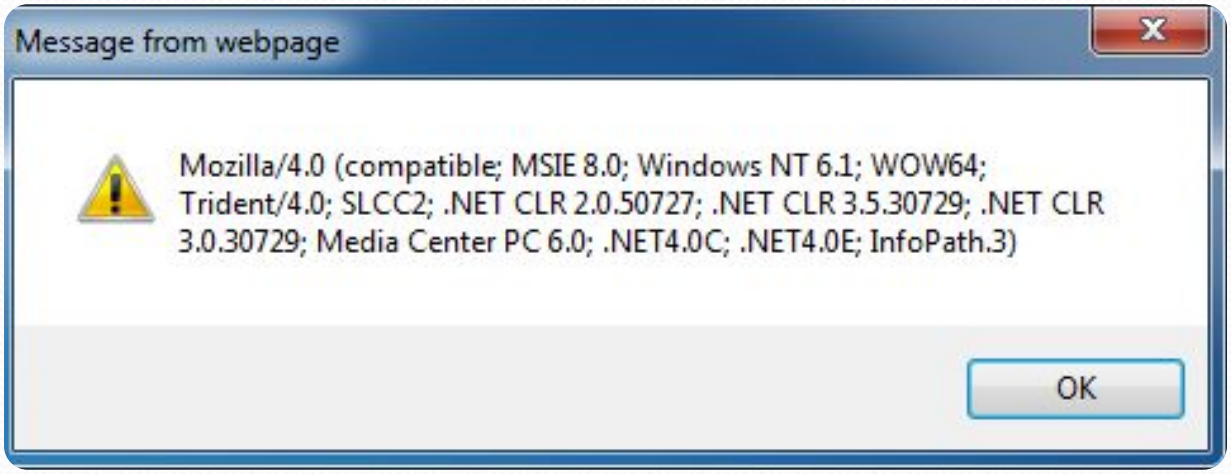
The Navigator Object

```
alert(window.navigator.userAgent);
```

The browser window

The navigator in the browser window

The userAgent (browser ID)



The Screen Object

- The `screen` object contains information about the display

```
window.moveTo(0, 0);
x = screen.availWidth;
y = screen.availHeight;
window.resizeTo(x, y);
```

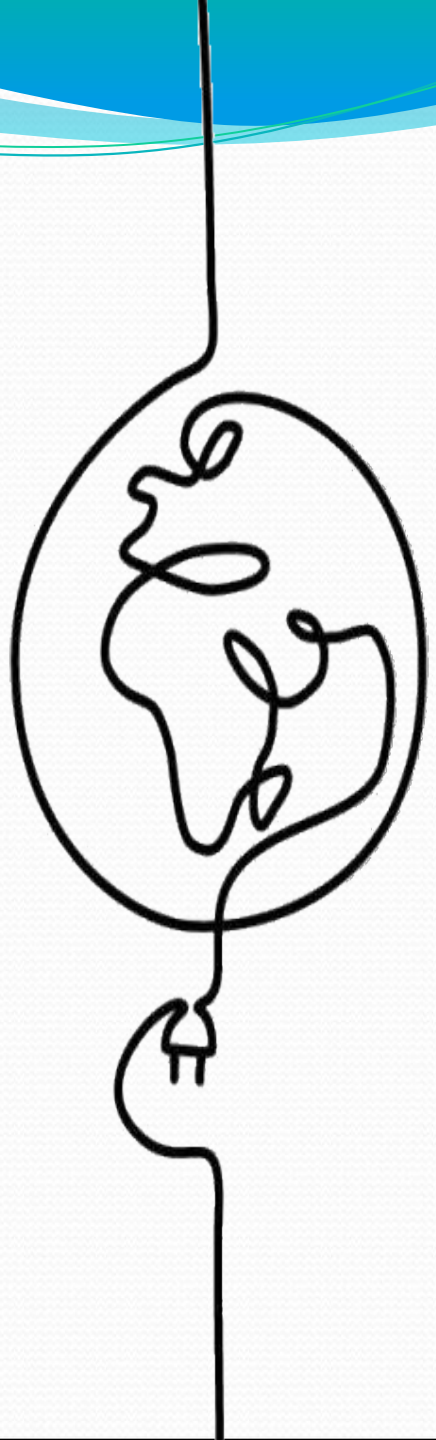
telerik

Form Validation – Example

form-validation.html

```
function checkForm()
{
  var valid = true;
  if (document.mainForm.firstName.value == "") {
    alert("Please type in your first name!");
    document.getElementById("firstNameError").
      style.display = "inline";
    valid = false;
  }
  return valid;
}
…
<form name="mainForm" onsubmit="return checkForm()">
  <input type="text" name="firstName" />
  …
</form>
```

Events



Event Handling

- **JavaScript events**, which allow scripts to respond to user interactions and modify the page accordingly.
- Events and event handling help make web applications more dynamic and interactive.

Registering an Event Handler

- An **event handler** is a function that responds to an event.
- Assigning an event handler to an event for a DOM node is called **registering an event handler**.

addEvent Listener

- Method **addEventListener** is available for every DOM node.

The method takes three arguments:

- The first is the name of the event for which we're registering a handler.
- The second is the function that will be called to handle the event.
- The last argument is typically false.[optional]

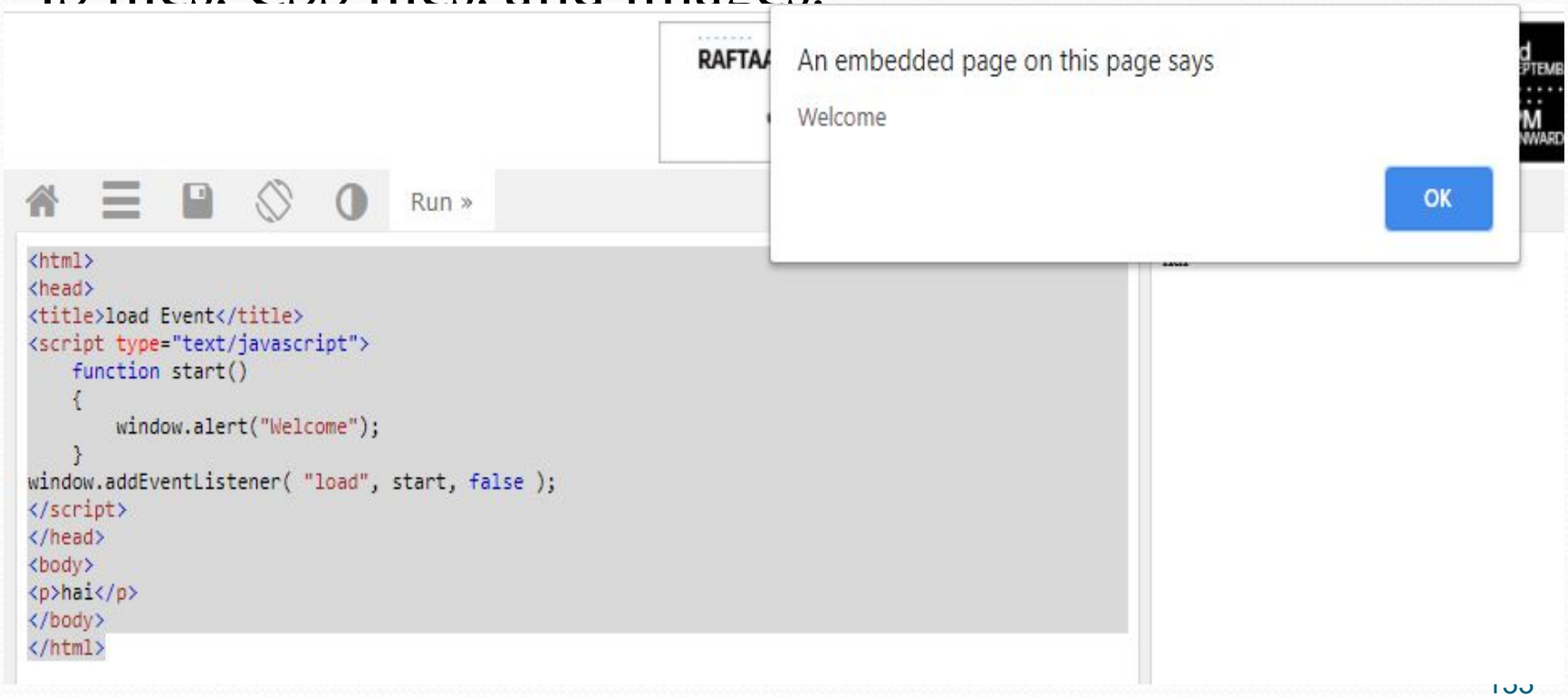
Syntax: addEventListener("Event",Function,usecapture);

Example:

window.addEventListener("**load**", startTimer, **false**);

Load Event

- The load event occurs when the document has been completely loaded, including dependent resources like JS files, CSS files, and images.



Mouse Events

- **mousemove**-The event occurs when the pointer is moving while it is over an element
- **mouseover** - The event occurs when the pointer is moved onto an element
- **mouseout**- The event occurs when a user moves the mouse pointer out of an element

Example 1: mousemove

```
<!DOCTYPE html>
 <html>
 <head>
    <title>Ist javascript with alert</title>

    <script type = "text/javascript">
    function moveone()
    {
        document.getElementById("move").innerHTML ="Kongu Engineering College";
    }

    </script>

</head>
<body>
    <p  id ="move" onmousemove="moveone()"> KEC </p>
</body>
 </html>
```

Example 2: mousemove and mouseout

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

    function start()
    {
    document.getElementById("move").addEventListener("mousemove",
moveone, false );
    document.getElementById("move").addEventListener("mouseout",
leaveone, false );
    }

    function moveone()
    {
    document.getElementById("move").innerHTML ="Kongu Engineering
College";
    }

    function leaveone()
    {
    document.getElementById("move").innerHTML ="KEC";
    }
    window.addEventListener( "load", start, false );
</script>
</head>
<body>
<p  id ="move"> KEC </p>
```

Example 3: mouseover and mouseout

```
 <title>Mouse_Event_Listener</title>
<script type = "text/javascript">
    window.addEventListener( "load", start, false );

    function start()
    {
    document.getElementById("picture").addEventListener("mouseover", over, false );
    document.getElementById("picture").addEventListener("mouseout", out, false );
    }

    function over(e)
    {
     document.getElementById("picture").setAttribute( "src", "grass.jpeg");
    }

    function out(e)
    {
    e.target.setAttribute( "src", "flower.jpg");
    }

</script>
</head>
<body>

    <img src = "flower.jpg" id = "picture" alt = "Heading Image">
```

Form Events

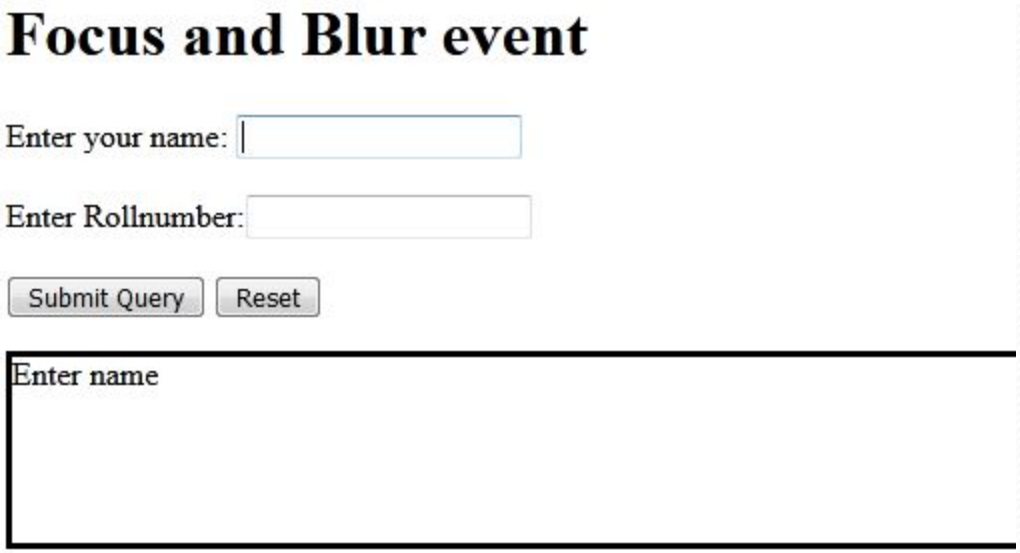
- The **focus** and **blur** events can be useful when dealing with form elements that allow user input.
- focus event fires when an element gains the focus (i.e., when the user clicks a form field or uses the *Tab* key to move between form elements)
- blur fires when an element loses the focus, which occurs when another control gains the focus.
- submit and reset events fire when a form is submitted or reset

Form Events

```
<script>
window.addEventListener( "load", start, false );
function start()
{
document.getElementById("fname").addEventListener("focus",focusone, false );
document.getElementById("fname").addEventListener("blur", blurone, false );
document.getElementById("myform").addEventListener("submit",subfun, false );
document.getElementById("myform").addEventListener("reset",resetfun, false );
}

function focusone() {document.getElementById("help").innerHTML="Enter name" ;}
function blurone() { document.getElementById("help").innerHTML="" ;}
function subfun()
{ window.alert("Are you Sure to submit");
  window.alert("Thank you");
}
function resetfun() {window.alert("Are you Sure to reset")}
</script>

</head>
```

Form Events

```
<body>
<h1>Focus and Blur event</h1>
<form id="myform">
Enter your name: <input type="text" id="fname"><br><br>
Enter Rollnumber:<input type="text" id="roll"><br><br>
<input type="submit"  id="sub">
<input type="reset"  id="reset">
</form>
<p id="help" style="border:solid; height:70pt;"></p>
</body>
```



Event bubbling

- **Event bubbling** is the process by which events fired on *child* elements "bubble" up to their *parent* elements.
- When an event is fired on an element, it's first delivered to the element's event handler (if any), then to the parent element's event handler (if any).
- cancelBubble=True: Stops bubbling and event will handle only in child element

Example-Event Bubble

```
<script>
    function documentClick()
    {
        alert( "You clicked in the document." );
    }
    function bubble( e )
    {
        alert( "This will bubble." );
        e.cancelBubble = false;
    }
    function noBubble( e )
    {
        alert( "This will not bubble." );
        e.cancelBubble = true;
     }

    function registerEvents()
    {
    document.addEventListener( "click", documentClick, false );
    document.getElementById( "bubble" ).addEventListener("click", bubble, false );
    document.getElementById( "noBubble" ).addEventListener("click", noBubble, false );
    }
    window.addEventListener( "load", registerEvents, false );
</script>
```

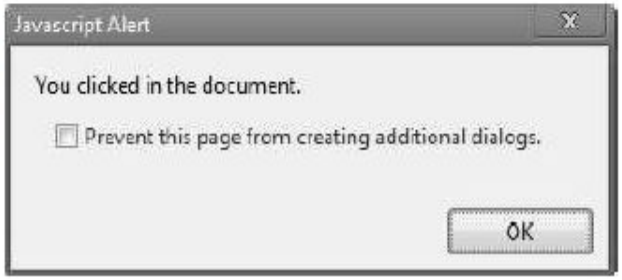
```
<body>
    <p id = "bubble">Bubbling enabled.</p>
    <p id = "noBubble">Bubbling disabled.</p>
</body>
```

Output

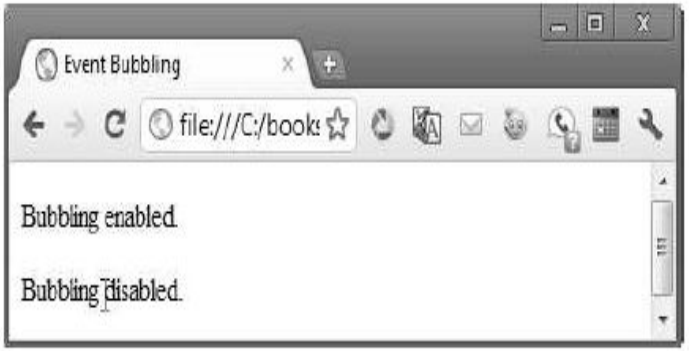
a) User clicks the first paragraph, for which bubbling is enabled.

b) Paragraph's event handler causes an alert.

c) Document's event handler causes another alert, because the event bubbles up to the document.



d) User clicks the second paragraph, for which bubbling is disabled.



e) Paragraph's event handler causes an alert. The document's event handler is not called.

Event	Description
`abort`	Fires when image transfer has been interrupted by user.
`change`	Fires when a new choice is made in a `select` element, or when a text input is changed and the element loses focus.
`click`	Fires when the user clicks the mouse.
`dblclick`	Fires when the user double clicks the mouse.
`focus`	Fires when a form element gets the focus.
`keydown`	Fires when the user pushes down a key.
`keypress`	Fires when the user presses then releases a key.
`keyup`	Fires when the user releases a key.
`load`	Fires when an element and all its children have loaded.
`mousedown`	Fires when a mouse button is pressed.
`mousemove`	Fires when the mouse moves.
`mouseout`	Fires when the mouse leaves an element.
`mouseover`	Fires when the mouse enters an element.
`mouseup`	Fires when a mouse button is released.
`reset`	Fires when a form resets (i.e., the user clicks a reset button).
`resize`	Fires when the size of an object changes (i.e., the user resizes a window or frame).
`select`	Fires when a text selection begins (applies to `input` or `textarea`).
`submit`	Fires when a form is submitted.
`unload`	Fires when a page is about to unload.

Fig. 13.13 | Common events.

Thank you