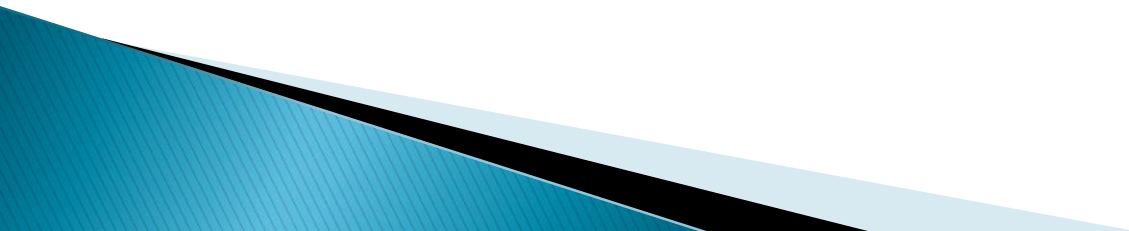
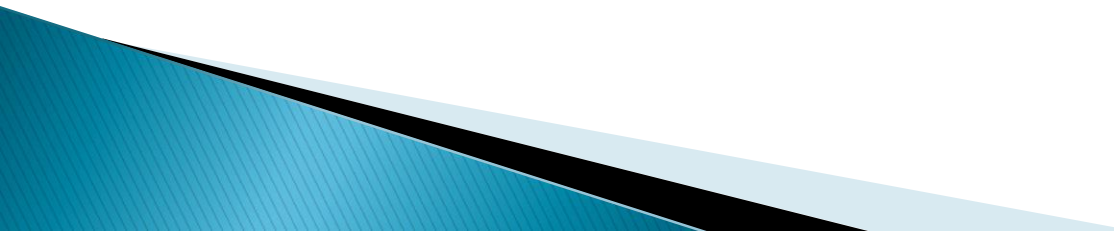


# JavaScript




# Javascript introduction

- ▶ JavaScript originated from a language called LiveScript.
  - ▶ JavaScript is a simple language used for simple tasks.
  - ▶ JavaScript is platform independent and can run anywhere.
  - ▶ It mainly used to manipulate pieces of Document Object Model (DOM).
  - ▶ Document Object Model (DOM) is an interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.
- 


- ▶ The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- ▶ The HTML DOM is a standard object model and programming interface for HTML. It defines:
  - The HTML elements as objects
  - The properties of all HTML elements
  - The methods to access all HTML elements
  - The events for all HTML elements

# Advantages of JavaScript

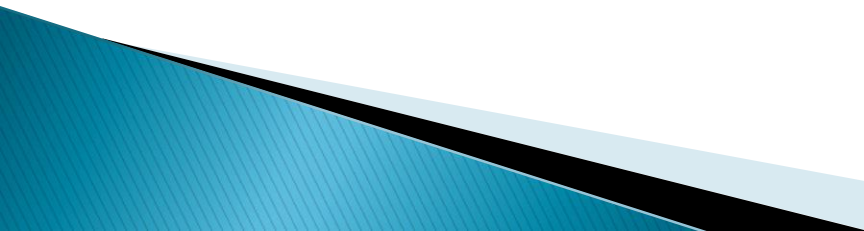
- ▶ Widely supported in web browsers
  - ▶ Can access and manipulate document objects easily
  - ▶ Can give interesting animations in less time
  - ▶ Secure
  - ▶ Web surfers do not need a special plug-in to use java scripts.
  - ▶ Embedded within HTML
  - ▶ Improved performance
  - ▶ Minimal syntax
  - ▶ An interpreted language
  - ▶ Good programming constructs & designed for programming events(event handling)
- 

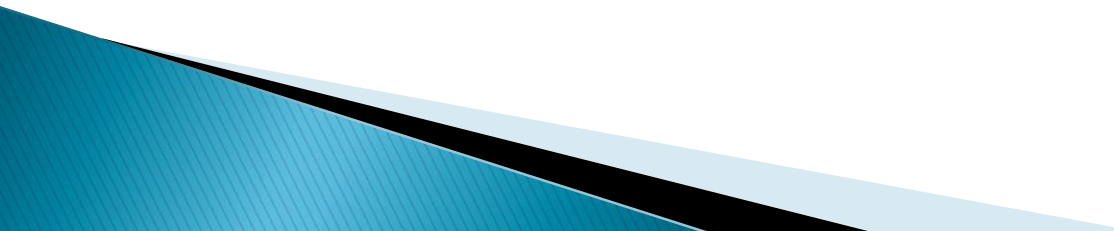
- ▶ Easy debugging
- ▶ Platform independence
- ▶ Programming tool for HTML designers
- ▶ Flexibility in putting dynamic text in an HTML page
- ▶ Powerful enough to validate data
- ▶ Determine visitors browser, create cookies

## **Disadvantages of JavaScript**

- ▶ Access to objects differ from browser to browser
  - ▶ Web page is useless if the script does not work
  - ▶ Scripts run slowly & complex scripts takes time
- 

# Javascript basics

- ▶ Javascript is client side scripting language
  - ▶ In some case javascript code resembles C.
  - ▶ Javascript code is interpreted and executed by the web browser itself
  - ▶ Browser can debug and display errors
  - ▶ Javascript is mainly written in the head tag of an HTML page.
  - ▶ Js is case sensitive
- 

- ▶ Each line is terminated by a semicolon
  - ▶ Blocks of code must be surrounded by a pair of curly brackets
  - ▶ Functions have parameters which are passed inside parenthesis
  - ▶ Variables are declared using keyword 'var'
  - ▶ Execution of a code starts with the first line of code and runs until there is no more code
  - ▶ Single line can be commented using // and multiple lines using /\*...\*/
  - ▶ Concatenation character is +
- 

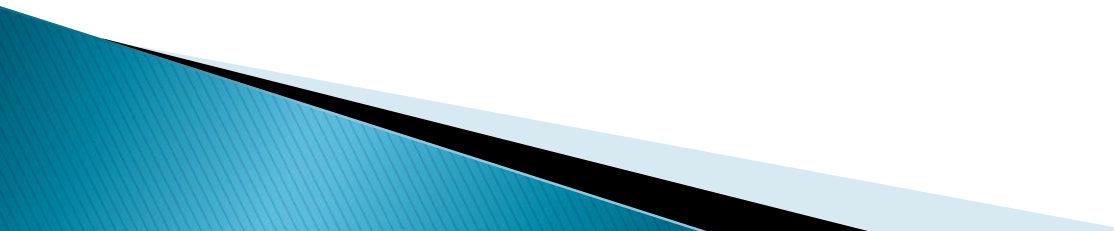
- ▶ HTML tags can be inserted in javascript by putting them in quotes.
  - ▶ document.write – to write in the webpage using js
- Eg : document.write("<h1>test</h1>");

## Syntax :

```
<html>
  <head>
    <script language="javascript" type = "text/javascript">
      <!--
          // javascript code here
      // -->
    </script>
  </head>
<body>
</html>
```

Put the scripts in html comments to avoid displaying the script as part of web page if it is disabled in your browser



- ▶ Language – This attribute specifies what scripting language you are using. Typically, its value will be javascript.
  - ▶ Type – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".
- 

# JavaScript Output

- ▶ JavaScript can "display" data in different ways:
  - 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()**.
- ▶ To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

Example1 :

```
<html>
```

```
  <body>
```

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

```
      <!--
```

```
        document.write("Hello World!")
```

```
      //-->
```

```
    </script>
```

```
  </body>
```

```
</html>
```



# Use of external javascript file

- ▶ Instead writing javascript code inside the html page, we can include an external javascript file to an HTML page.
- ▶ I.e. Write all the js codes in a file with extension .js and call that file in the head section of an HTML page

## Syntax :

```
<html>  
  <head>  
    <script language="javascript" src="sam.js"></script>  
  </head>  
<body>  
</html>
```

# variables

- ▶ Variables are containers for storing information which has a name.
- ▶ Variables names are created using keyword 'var'
- ▶ Rules in creating variables
  - Variables must begin with a letter, digit or underscore
  - Cannot use spaces
  - Variable names are case sensitive
  - Cannot use reserved word as a variable name

Eg : var x;

x=5;

Or

Var x=5;

# Datatypes

- ▶ In js the same variable can hold different types of data.
- ▶ It is not necessary to declare or specify the data type of a variable before using it.
- ▶ Data type of a variable is determined by the type of value the variable holds.
- ▶ Js has only four types of data
  - **Numeric** – these are basic numbers. They can be integers like 234 or floating point numbers like 23.43, -43.45 and 2E45 etc

- **Strings** – it is a collection of letters, digits, spaces and other special characters. The value of a string variable should be in double or single quotes.

eg : x="glen" or x=" 45.2" or x='4aa'

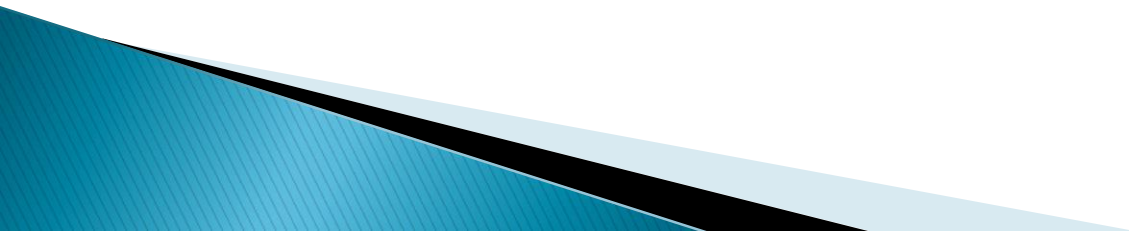
x="hello 'world' hi"

x="hello \"world\" hi"

- **Boolean** – boolean variables holds values true and false.

eg : var x= true;

- **Null** – it consists of a single null value.  
Usually this is used to initialize a variable.





# Converting data types

- ▶ Js variables are loosely typed

```
Var x= "12";
```

```
var y = parseInt(x);
```

# JavaScript Operators – Arithmetic Operators

Operator	Description	Example	Result
+	If the arguments are numbers they are added and if they are strings they are concatenated	x=2 y=2 x+y	4
-	Subtracts one operand from the other. If only one operand then it reverses the sign	x=5 y=2 x-y	3
*	Multiplies two numbers together	x=5 y=4 x*y	20
/	Divides the first no by the second no	15/5 5/2	3 2,5
%	Modulus returns the integer division remainder	5%2 10%8 10%2	1 2 0
++	Increment by one	x=5 x++	x=6
--	Decrement by one	x=5 x--	x=4

# Assignment Operators

Operator	Description	Is The Same As
<code>=</code>	Assigns a value to a variable	<code>x=y</code> <code>=</code> <code>x=y</code>
<code>+=</code>	Adds 2 nos and result is assigned to the variable on the left	<code>x+=y</code> <code>=</code> <code>x=x+y</code>
<code>-=</code>	Subtracts the right term from the left & result is assigned to the left var	<code>x-=y</code> <code>=</code> <code>x=x-y</code>
<code>*=</code>	multiplies 2 nos and result is assigned to the variable on the left	<code>x*=y</code> <code>=</code> <code>x=x*y</code>
<code>/=</code>	divides right term by left term & result is assigned to the left variable	<code>x/=y</code> <code>=</code> <code>x=x/y</code>
<code>%=</code>	Result of modulus division is assigned to the left variable	<code>x%=y</code> <code>=</code> <code>x=x%y</code>

# Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5"  x==y returns true  x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

# Logical Operators

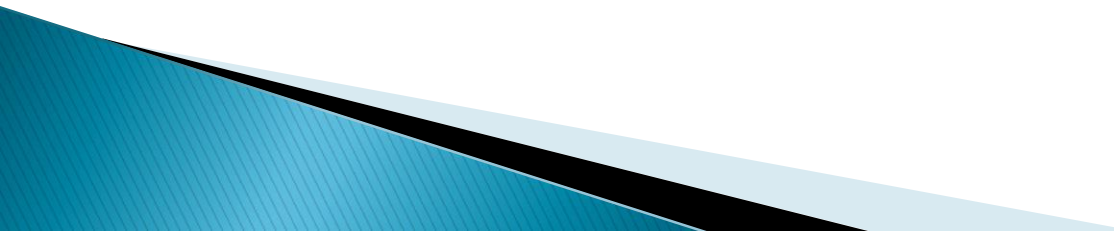
Operator	Description	Example
&& (AND)	Returns true if both operands are true, otherwise false	x=6 y=3  (x < 10 && y > 1) returns true
(OR)	Returns true if one or both operands are true, otherwise false	x=6 y=3  (x==5    y==5) returns false
!(NOT)	Returns false if operand evaluates to true, otherwise false	x=6 y=3  !(x==y) returns true

- ▶ String Operators
  - String concatenation operator +

- ▶ Special operators
  - Delete
    - Is used to delete a property of an object or an element but an array index.
    - Delete myArray[5] – deletes 6<sup>th</sup> element from myArray.
  - New
    - Is used to create an instance of an object type.
    - myArray = new Array()

# statements

## Conditional Statements

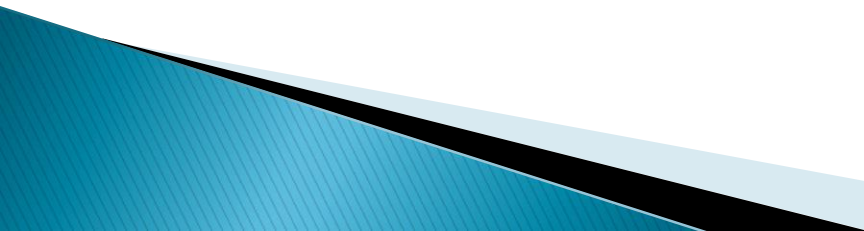
- ▶ **if statement** – use this statement to execute some code only if a specified condition is true
  - ▶ **if...else statement** – use this statement to execute some code if the condition is true and another code if the condition is false
  - ▶ **if...else if....else statement** – use this statement to select one of many blocks of code to be executed
  - ▶ **switch statement** – use this statement to select one of many blocks of code to be executed
- 

# Conditional Statements – syntax

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

---

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






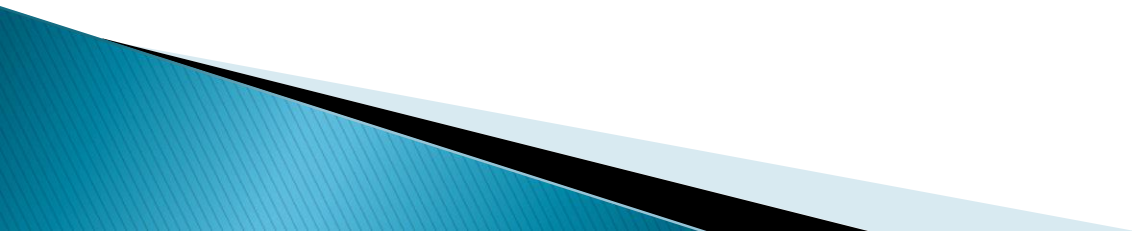
```
if (condition)
{
code to be executed if condition is true
}
else if (condition2)
{
code to be executed if condition1 is false & condition2 is true
}
else
{
code to be executed if both conditions are false
}
```

---

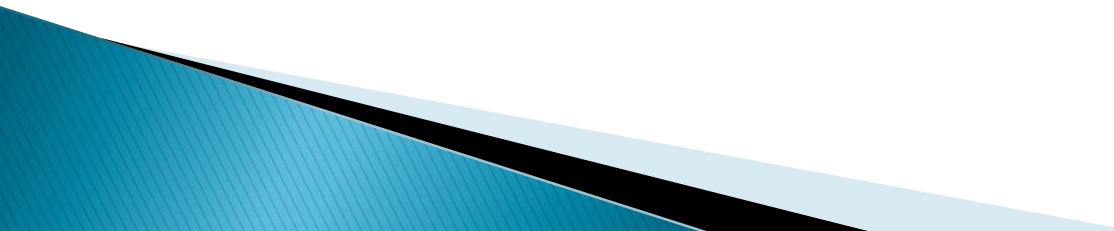
```
if (condition1)
{
    if(condition2)
    {
code to be executed if condition1 & condition2 is true
    }
}
else
{
code to be executed if condition1 is false
}
```



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



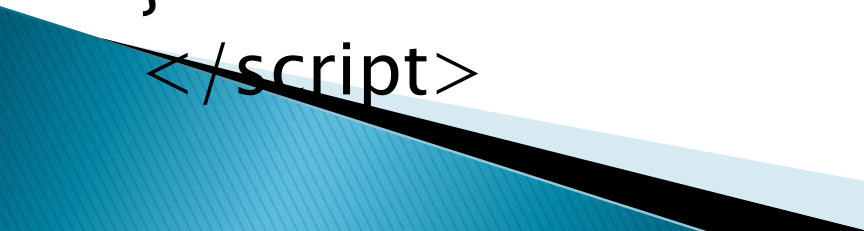
# Switch

- ▶ The single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each ***case*** in the structure.
  - ▶ If there is a match, the block of code associated with that case is executed. Use ***break*** to prevent the code from running into the next case automatically.
  - ▶ The ***default*** keyword specifies what to do if there is no match.
- 

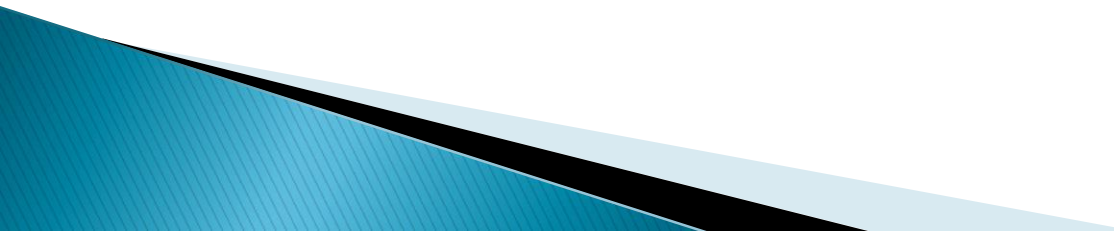
# Conditional Statements Examples

```
<script>  
  x=3;  
  if(x<0)  
  {  
    alert ("negative");  
  }  
  else  
  {  
    alert ("positive");  
  }  
</script>
```

```
<script>
var x;
var d=new Date().getDay();
switch (d)
{
case 0: alert("Today is Sunday");
break;
case 1: alert(" Today is Monday");
break;
:
case 6: alert(" Today is Saturday");
break;
default: alert(" Invalid Value");
}
</script>
```



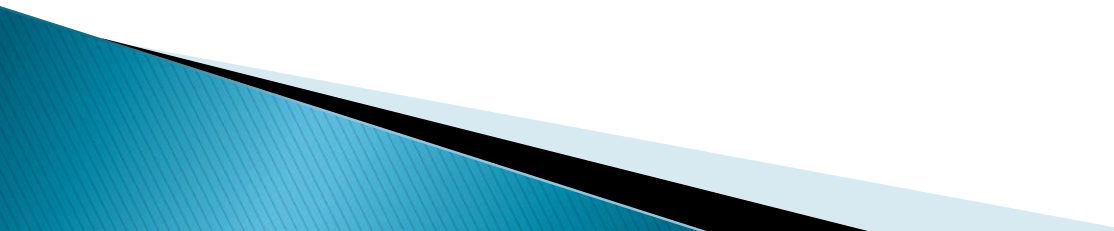
# Looping statements

- ▶ **for** – loops through a block of code a number of times
  - ▶ **while** – loops through a block of code while a specified condition is true
  - ▶ **do/while** – also loops through a block of code while a specified condition is true
- 

# The For Loop

*Syntax:*

```
for (initialization; condition; increment)  
{  
    set of statements  
}
```

- ▶ ***Initialization*** is executed before the loop starts.
  - ▶ ***Condition*** defines the condition for running the loop.
  - ▶ ***Increment*** is executed each time after the loop has been executed.
- 

## The While Loop

- ▶ The while loop loops through a block of code as long as a specified condition is true.

Syntax: **while** (*condition*)

```
{  
    code block  
}
```

## The Do/While Loop

- ▶ The do/while loop is a variant of the while loop. This will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax : **do**

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

- ▶ The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:



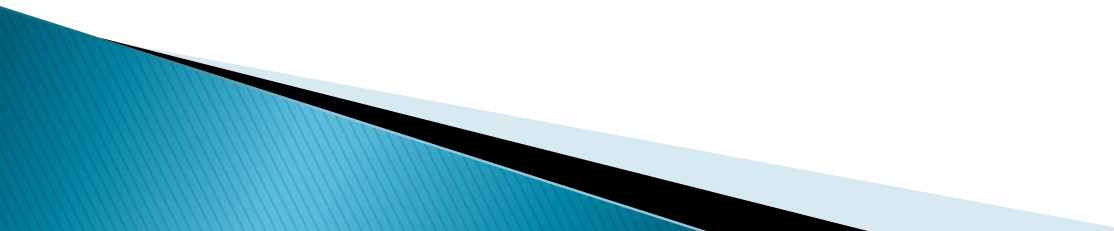
## The Break Statement

- ▶ It is used to "jump out" of a switch() statement and a loop.
- ▶ The **break** statement breaks the loop and continues executing the code after the loop (if any).

## The Continue Statement

- ▶ It breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

# Arrays

- ▶ An array is an ordered set of data elements which all has same data type and can be accessed through a single variable.
  - ▶ Array elements are accessed using their index
  - ▶ Index is the position of the element in the array and it starts from 0 to arraylength-1.
- 

There are 3 ways to create an array. They are :

1. Simply declare a variable and pass elements to it. Here array elements are surrounded by square brackets

```
var days=["Mon","Tue","Wed","Thur"];
```

2. Create an array object using keyword 'new'. Here array elements are surrounded by parenthesis.

```
var days = new Array("Mon","Tue","Wed","Thur");
```

3. An array object which has space for a number of elements can be created.

```
var days = new Array(4);
```

- ▶ Js arrays can hold mixed data types
- ▶ Elements can be added to an array as

```
var days[4]="Fri";
```



# Object based array functions

**Syntax : arrayname.function(param1,param2)**

- ▶ **Array(item0, .....itemN)** – create an array with the specified elements
- ▶ **concat(array2,...,arrayN)** – concatenates one more arrays to an array.

`arr1.concat(arr2,arr3)`

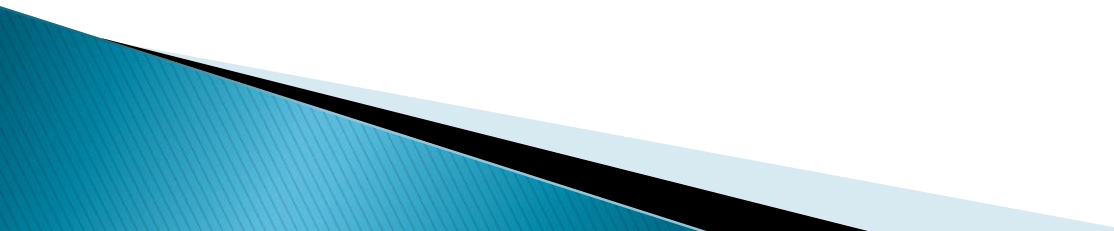
- ▶ **slice(start [,finish])** – to extract range of elements from an array. In start and finish, index values are specified. If finish index is not specified then extracts till the end of the array.

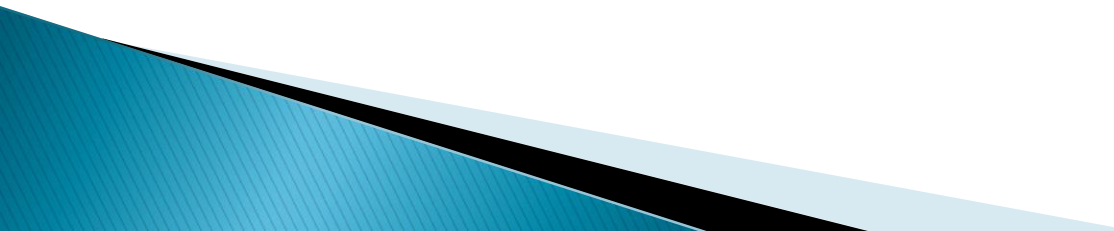
Eg 1 : `arr1.slice(2)`

Eg 2 : `arr1.slice(2,3)`

- ▶ **Splice(index,number[,element1,element2,elementN])**
  - used to alter an array by removing or adding new ones. First two parameters index and number is compulsory where index denotes the position in the array at which new elements will start. Number parameters indicates how many elements will be deleted from the original array. set the number field to 0 if no elements are to be deleted. This method changes the original array

***splice(2,0,"red","blue");***

- ▶ **sort()** – used to sort an array. This method changes the original array.
  - ▶ **unshift(item1,item2,...)** – The unshift() method adds new items to the beginning of an array, and returns the new length. This method changes the length of an array.
  - ▶ **push(item1,item2,...)** – To add new items at the end of an array and returns the new length.
- 

- ▶ `Pop()` – this function removes last element from the array and reduces the number of elements by one.
  - ▶ `Reverse()` – reverses the array.
  - ▶ `Shift()` – this removes the first element of the array and reduces its length by one.
  - ▶ `Join(string)` – all elements of an array joined together to form a string.
- 

# String manipulation

## String object Methods

### ▶ Length

Returns the no of characters in the string

```
Var x="hello";
```

```
Var len_x=x.length.    - 5
```

### ▶ charAt(index)

Returns the character at the specified index

```
x.charAt(2)
```

### ▶ charCodeAt(index)

Returns the Unicode of the character at the specified index



## ▶ concat()

Joins two or more strings, and returns a copy of the joined strings.

**Concat("string1 "[, "string2"[, ..... "stringn"]])**

```
var x="smitha ";  
var y="xavier";  
Var z=" p";  
alert(x.concat(y,z));
```

## ▶ indexOf("search"[, offset])

Gets the first index of a String within a String after an offset. i.e. Returns the position of the first found occurrence of a specified value in a string. If the search is successful, index of the start of the target string is returned. If the search is unsuccessful returns -1. By default the indexOf() starts at index 0. to change the starting index use offset value. Offset is optional.

Syntax :

**indexOf("search string" [, offset])**

Eg : `y.indexOf("e",5)`

## ▶ lastIndexOf()

Gets the last index of a character within a String after an offset. ie. Returns the position of the last found occurrence of a specified value in a string. Here the use of offset is same as in indexOf, but the default value is `string.length-1`. (offset value is counted from last of string)

**`lastIndexOf("search string" [, offset])`**

Eg : `y.lastIndexOf("e",5)`

- ▶ replace()

Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring

`x.replace("search_string", "replace_string")`

- ▶ search()

Searches for a match between a regular expression and a string, and returns the position of the match

`x.search("search_string")`



## ▶ split()

Splits a string into an array of substrings. Limit is used to specify how many pieces are to be stored in the array.

**Split(separator[,limit]);**

```
p="depaul college union";  
res=p.split(" ");
```

0/p : depaul, college, union

```
res=p.split(" ",2); – depaul, college
```

## ▶ substr()

Extracts the characters from a string, beginning at a specified start position, and through the specified number of characters indicated by the length. If length is not specified, substring is retrieved till the end of the string

*substr(index[, length])*

p.substr(5,7) – starting from 5<sup>th</sup> letter, 7 letters are retrieved

## ▶ substring()

Extracts the characters from a string, between two specified indices(i.e range).

*substring(index1[, index2])*

p.substring(5,7) – retrieves letters between 5 to 7 positions

- ▶ toLowerCase()

Converts a string to lowercase letters  
**x.toLowerCase();**

- ▶ toUpperCase()

Converts a string to uppercase letters  
**x.toUpperCase();**

# Mathematical functions

- ▶ Mathematical functions and values are a part of built-in JS object called Math.
- ▶ Math object provides methods and properties used in complex mathematics offered by arithmetic operators.
- ▶ Syntax for using properties / methods of Math.  
    `var x = Math.PI;`  
    `var y = Math.Sqrt(16);`
- ▶ Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.



- ▶ Another method of using *Math object* to .

with (Math)

```
{  
  var area = pi *(r*r);  
  var next = ceil(area);  
}
```

NaN – this is a value representing Not a Number

eval() – this is a js built-in function. String versions of mathematical expressions can be passed into the function where they are evaluated and the result is returned as an integer.

## Global math methods

- ▶ **isNaN(parameter1)** – this function returns true if the parameter value is not a number and false if it is numeric.
- ▶ **parseInt(string[,radix])** – string is parsed and its value as an integer returned. radix is used to mention the base of given number. By default radix is 10. during parsing the number is converted into decimal.

**parseInt(123sd) – 123**

- ▶ **parseFloat(string)** – the string is returned as a floating point number.

# Math Object Methods

- ▶ `abs(x)` – Returns absolute value of  $x$
- ▶ `acos(x)` – Returns the arccosine of  $x$ , in radians
- ▶ `asin(x)` – Returns the arcsine of  $x$ , in radians
- ▶ `atan(x)` – Returns the arctangent of  $x$ , in radian
- ▶ `cos(x)` Returns the cosine of  $x$ , in radians.
- ▶ `exp(x)` Returns the value of  $E^x$ .
- ▶ `ceil(x)` Returns  $x$ , rounded upwards to the nearest integer.

- ▶ `floor(x)` – Returns  $x$ , rounded downwards to nearest integer.
- ▶ `log(x)` – Returns the natural logarithm (base  $E$ ) of  $x$ .
- ▶ `max(x,y,z,...n)` – Returns the max value.
- ▶ `min(x,y,z,...n)` – Returns the min value.
- ▶ `pow(x,y)` – Returns the result of  $x^y$
- ▶ `random()` – Returns a pseudorandom number between 0 & 1
- ▶ `round(x)` – Returns  $x$ , rounded to the nearest integer
- ▶ `sqrt(x)` – Returns square root of  $x$

# JavaScript Popup Boxes

## ▶ Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.

```
<script>
```

```
alert("Hello World!")
```

```
</script>
```



## ► Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
var x=confirm("do you want to continue")
if(x)
{
  Alert("you can continue");
}
else
{
  alert("cant continue");
}
```

## ► Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK", the box returns the input value. If the user clicks "Cancel", the box returns null.

**Prompt("message", "default value")**

This message will come on top of prompt box and the default value will come inside the textbox when the prompt box appears. For numerical default value, no need of double quotes

# Prompt Box Example

```
<script>
```

```
x=prompt ("Enter a number", 1)
```

```
document.write("number is : <br>",+x)
```

```
</script>
```

```
<script>
```

```
x=prompt ("Enter the name", "No Name")
```

```
document.write("Name is : <br>",+x)
```

```
</script>
```






# Functions

- ▶ A function is a piece of code that performs a specific task and often returns a value.
- ▶ Every function is made up of a number of statements.
- ▶ Using functions same piece of code can be used repeatedly(i.e reusable code block).
- ▶ The functions created are used by *calling* the function.
- ▶ Functions can be developed without affecting the rest of the program.
- ▶ Parameters can be passed to a function and a function can return zero or more parameters

There are two types of functions

- ▶ **Built-in functions** – some of the built-in(already defined) functions in java script are eval(), parseInt(), parseFloat() etc. the name of these function can't be changed
  - ▶ **User-defined functions** – these type of function need to be declared and called. Here the user can define a function by giving any name and set of statements as block inside a function and can be called whenever we want.
- 

# User-defined functions – Defining Functions

- Functions are defined using the keyword function.
- Function name can be a combination of letters, digits and underscore and cannot use space(same rules for variable naming).
- Name of the function is followed by parenthesis which can contain values that you want to pass(if any)
- Function statements are surrounded by curly brackets
- Functions are called using function name and specify the parameter names if any.
- Syntax :

```
function function_name(parameters)
{
    //statements
}
```

# Parameter passing

- Every function does not have parameters.
- When a function receives a value as a parameter, that value is given a name and can be accessed using that name.
- The names of parameters are applied in the order in which parameters are passed.

```
<script>  
    function fn_name(emp_name,age)  
    {  
        document.write("Name: " +emp_name);  
        document.write("Age: " +age);  
    }  
    fn_name("Smitha",25);  
</script>
```

# Scope of variables

- ▶ If a variable is declared inside a function are local to that function and its value is not available outside the function unless we pass it as parameter. Such variables which have scope within the function are called **local variables**.
- ▶ Global scoping means that a variable is available to all parts of the program. A variable declared outside the function can be accessed by all the functions in that page . These variables are called **global variables** and usually they hold static data which does not alter .

# Returning values

- ▶ If a function needs to return a value, then a return statement must be used.
- ▶ The value will be returned to the statement from where the function has been invoked.

```
function myFunction()  
{  
  var x=5;  
  return x;  
}
```

```
<html>
<head>
<script>
function fn_add()
{
    var x=prompt("Enter first no",0);
    var y=prompt("Enter second no",0);
    var total = parseFloat(x) + parseFloat(y);
    return total;
}
</script>
</head>
<body>
<h2 id="head1"
onclick="document.getElementById('head1').innerHTML
=fn_add();">
    click to add 4 and 5</h2>
</body>
</html>
```

# JS Events

- ▶ HTML events are "**things**" that happen to HTML elements.
- ▶ When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.
- ▶ An HTML event can be something the browser does, or something a user does.
- ▶ Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- ▶ Often, when events happen, you may want to do something.
- ▶ JavaScript lets you execute code when events are detected.
- ▶ HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.



- ▶ With single quotes:  
<element event='some JavaScript'>
- ▶ With double quotes:  
<element event="some JavaScript">

<!DOCTYPE html>

<html>

<body>

<button  
onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>

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

</body>

</html>

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<button onclick="this.innerHTML=Date()">The  
time is?</button>
```

```
</body>
```

```
</html>
```



# DHTML events

- ▶ DHTML is not a programming language. It is a group of techniques (JavaScript, HTML, CSS and DOM) which is becoming standard. However, it is powerful tool for creating interactive web content.
- ▶ DHTML is a TERM describing the art of making dynamic and interactive web pages.
- ▶ DHTML is a new tool for designers to create Web pages with special effects and animation such as flying headlines and news tickers.
- ▶ Designers can attach a simple script to any object on a page and the user can change the appearance of a Web page "on the fly."

- ▶ Events can be defined as user actions like clicking a button, pressing a key, moving a window or moving a mouse, page loading etc.
- ▶ HTML events can trigger actions in a browser.
- ▶ Each and every element on an HTML page has events which can trigger a JavaScript.
- ▶ For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button.
- ▶ We define the events in the HTML tags.
- ▶ DHTML programmer can write code to run in response to the event.
- ▶ This particular segment of code is generally known as an event handler.

**Table 4–7 DHTML Event Attributes<sup>a</sup>**

<b>Attribute</b>	<b>Description</b>
onblur (16)	Element loses focus
onchange (11)	Element's value changes
onclick (17)	Mouse button is clicked over the element
ondblclick (21)	Mouse button is double-clicked over the element
onfocus (16)	Element receives focus
onkeydown (21)	Key is pressed
onkeypress (21)	Key is pressed and subsequently released
onkeyup (21)	Key is released
onload (1)	Page is loaded
onmousedown (21)	Mouse button is pressed over the element
onmousemove (21)	Mouse moves over the element
onmouseout (21)	Mouse leaves the element's area
onmouseover (21)	Mouse moves onto an element
onmouseup (21)	Mouse button is released
onreset (1)	Form is reset

```
<html>
<head>
<script>
    function fn1()
    {
        alert("empty function");
    }

    function fn_name(emp_name,age)
    {
        alert("name : " + emp_name + " age : " + age);
    }
</script>
</head>
<body onload="alert('body loaded');">

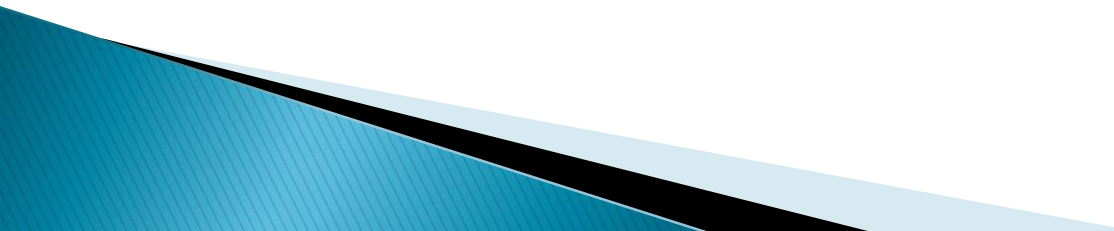
<a href="#" onclick="alert('hello');"> click </a><br>

<a href="#" onclick="fn1 ();"> click_empty function </a><br>

<a href="#" onmousemove="fn_name('maya',25);"> with parameters </a>
</body></html>
```

## DOM example

```
<html>
<head>
<script>
    function changetext(txt)
    {
        txt.innerHTML="It Worked !";
    }
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text</h1>
</body>
</html>
```



```
<html>
<head>

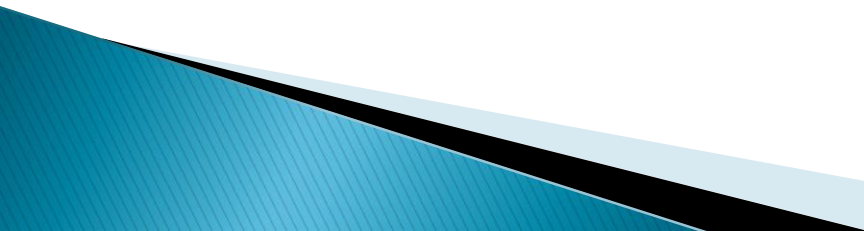
</head>
<body>
<h1 onclick=" this.innerHTML='It Worked !'; ">Click on
this text</h1>
</body>
</html>
```



```
<html>
<head>
<script>
  function fn1()
  {
    document.getElementById("head1").style.visibility="hidden";
  }
  function fn2()
  {
    document.getElementById("img1").src="../../images/img_html.jpeg";
  }
</script>
</head>
<body>
<h1 id="head1" onclick="fn1();" >Click Here</h1>

</body>
</html>
```

# Objects in JavaScript

- ▶ "Everything" in JavaScript is an Object: a String, a Number, an Array, a Date....
  - ▶ In JavaScript, an object is data, with properties and methods.
  - ▶ **Properties** are **values** associated with an object.
  - ▶ **Methods** are **actions** that can be performed on objects.
  - ▶ There are user-defined and built-in objects
- 

- ▶ The keyword *new* is used to create objects. It allocates memory and storage for them and sets all variables that can be set.
- ▶ *New* calls a function which has same name as the type of object that is being created. This function is called constructor
- ▶ Objects can have functions as well as variables
- ▶ *this* keyword is used refer the current object. To refer a property, method or a variable, a dot(.) is placed between the object name and the property, method or variable.

*this.var\_name*



- ▶ You create a JavaScript String object when you declare a string variable like this:

*var txt = new String("Hello World");*

- ▶ String objects have built-in properties and methods:

Object – "Hello World"

Property – txt.length

Method – txt.indexOf("World")

<script>

```
var person=new Object();
```

```
person.firstname="John";
```

```
person.lastname="Doe";
```

```
person.age=50;
```

```
person.eyecolor="blue";
```

```
document.write(person.firstname + " is " +  
person.age + " years old.");
```

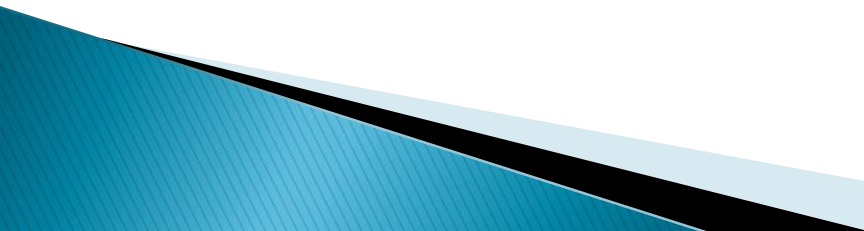
</script>



# Regular Expression


- ▶ Regular Expression is a pattern which describes a set of characters that may be present in a string.
- ▶ Javascript versions after 1.1 include a set of routines to manipulate strings and search patterns. These are wrapped up as regular expression objects
- ▶ Regular expressions is a form of pattern matching that you can apply on textual content.
- ▶ Take for example the DOS wildcards ? and \* which you can use when you're searching for a file. That is a kind of very limited subset of RegExp.

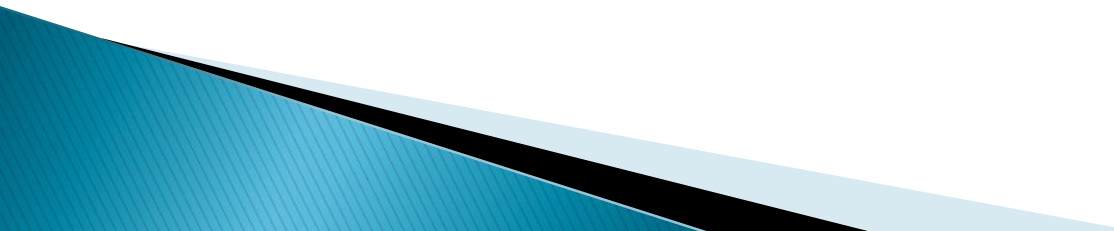
- ▶ A regular expression is a javascript object. It can be created statically or dynamically.
- ▶ Static RE is created when the script is parsed as:  
***regex=/fish | fowl/;***
- ▶ Dynamic patterns are created using ***new*** keyword to create an instance of **RegExp** class  
***regex = new RegEXP("fish/fowl");***
- ▶ Regular expressions are, in short, a way to effectively handle data, search and replace strings, and provide extended string handling.

- ▶ RE can be expressed as a set of symbols with grammar to replace complex statements.
  - ▶ Some of the characters in the JavaScript grammar are preceded by a backslash character. This is an escape character which tells the browser that the character indicates an operation not a letter.
  - ▶ in JavaScript forward slash (/) is used to declare RegExp literals.
  - ▶ The RegExp object holds the result of its operations in an array which it returns to the calling script.
- 



# Javascript RE grammar

- ▶ **^** – match at the start of the input string
  - ▶ **\$** – match at the end of the input string
  - ▶ **\*** – match 0 or more times
  - ▶ **+** – match 1 or more times
  - ▶ **?** – match 0 or 1 time
  - ▶ **a|b** – match a or b
  - ▶ **{n}** – match the string n times
  - ▶ **\d** – match the digit
  - ▶ **\D** – match anything except for digits
  - ▶ **\w** – match any alphanumeric character or underscore
- 

- ▶ `\W` – match anything except alphanumeric character or underscore
  - ▶ `\s` – match a whitespace character
  - ▶ `\S` – match anything except whitespace characters
  - ▶ `[...]` – creates a set of characters, one of which must match. Range can be specified as `[0–9]` or `[D–G]`
  - ▶ `[^...]` – creates a set of characters which must not match. Here also range can be specified.
- 

- ▶ RE can be manipulated using functions of RegExp or String classes

## String Class functions

- ▶ *match(pattern)* – searches for a match pattern. Returns an array holding the results, or null if no match found

```
var str = "Rain in SPAIN stays mainly in the plain";  
var res = str.match(/ain/g);  
document.getElementById("demo").innerHTML=res;
```

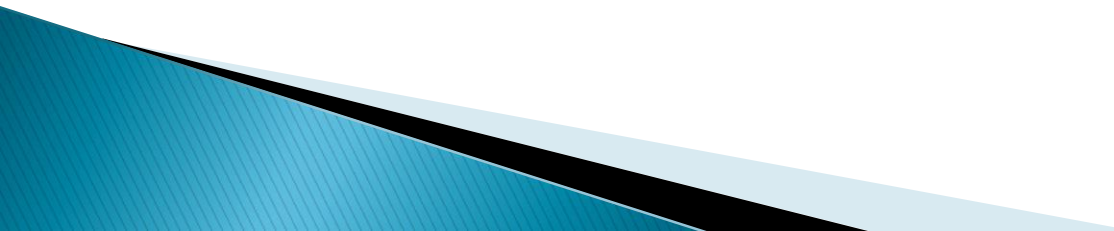
- *replace(pattern1,pattern2)* – searches for pattern1. if search is successful pattern1 is replaced with pattern2

```
var res = str.replace("Microsoft","Wipro");
```

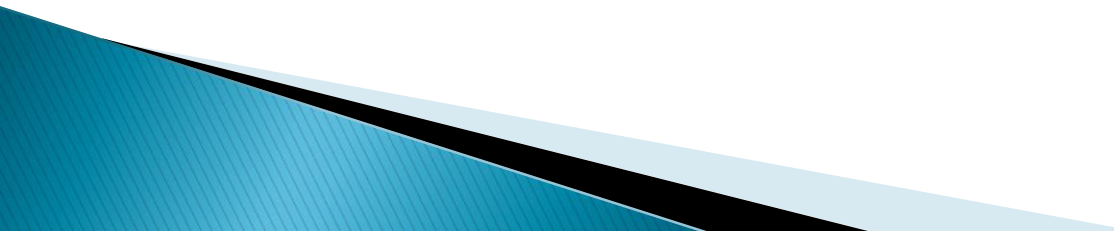
- *search(pattern)* – searches for a pattern in the string. If match is successful, the index, offset of the start of the match is returned and if search fails, returns -1

```
var n = str.search("the");
```



- ▶ *split(pattern)* –split the string into parts based on the pattern or RE
  - ▶ *escape(string)*
  - ▶ *unescape(string)* – *escape()* takes a string as parameter and returns a new string with characters converted to escape sequences. *unescape()* does the reverse i.e. converts escape sequences into characters.
- 

## Class RegExp functions

- *exec(string)* – executes a search for a matching pattern. It returns an array holding the results of the operation.
  - *test(string)* – searches for a match in its parameter string. It returns true if a match is found, otherwise returns false.
- 

Three *flags* for RegExp objects are :

- ▶ **i** – performs search ignoring the case
- ▶ **m** – allows searching data which spans several input lines
- ▶ **g** – forces global matching without stopping when a match is found

- ▶ Flags can be applied directly into RE

```
Regex=/fish|fowl/ig
```

- ▶ Flags can be applied as an additional parameter to the object constructor

```
regex=new RegExp("fish|fowl","ig");
```

- ▶ `RE_name=new RegExp("^[A-Z][a-zA-Z_.']+$", "g");`

`//can end with a-zA-Z_.'`

- ▶ `RE_age=new RegExp("^[0-9]+[0-9]$");`



# Exception Handling

- ▶ A mechanism for handling the run time errors with general classes of error is called Exception Handling.
- ▶ A user is aware of the different choices or behaviors of a program at any time. So we can write code to cope with these situations.
- ▶ An exception on object-based programming is an object, created dynamically at runtime, which encapsulates an error and some information about it.
- ▶ The great thing about exceptions is that you can define your own exception classes to include exactly what you need in order to handle the problem successfully
- ▶ Incorrect inputs are described by `UserInputException` objects

## ▶ **Throw**

- ▶ An exception is an object. Created using new keyword.
- ▶ Syntax:
- ▶ Do something
- ▶     if an error happens {
- ▶         create a new exception object
- ▶         throw the exception
- ▶     }

## ► try ..... Catch

```
try {  
    statement1  
    statement2  
    statement3  
} catch exception {  
    handle the exception  
}
```

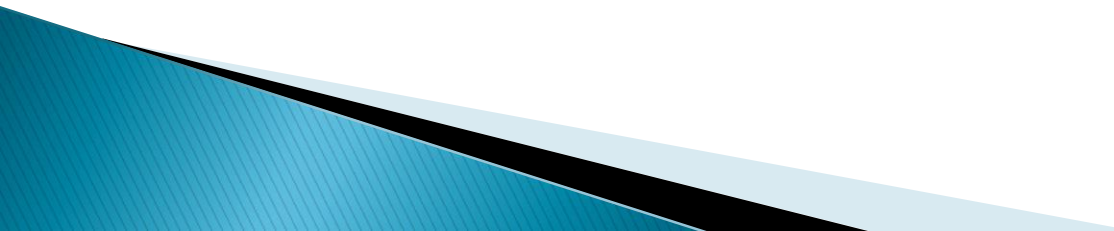
- ▶ `<html><body>`
- ▶ `<div id="example"></div>`
- ▶ `<script>`
- ▶ `try{`
- ▶ `fn1();`
- ▶ `alert('hello');`
- ▶ `}`
- ▶ `catch(e){`
- ▶ `alert('An error has occurred: '+e.message)`
- ▶ `}`
- ▶ `finally{`
- ▶ `alert('I am alerted regardless of the outcome`  
`above') // it will execute for every try whether`  
`error occurred or not`
- ▶ `}`
- ▶ `</script>`
- ▶ `</body></html>`

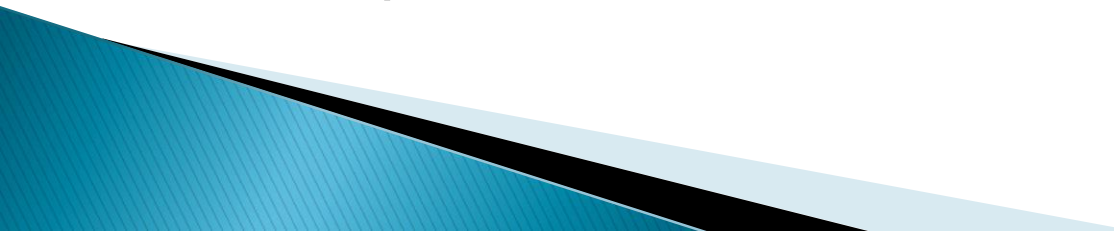
```
> <script>
> var empcode = prompt("Input the Employee code : (Between 3 to 8 characters):","");
> try
> {
>     if(empcode.length>8)
>     {
>         throw "error1";
>     }
>     else if(empcode.length<3)
>     {
>         throw "error2";
>     }
> }
> catch(err)
> {
>     if(err=="error1")
>     {
>         alert("The Employee code length exceed 8 characters.");
>     }
>     if(err=="error2")
>     {
>         alert("The Employee code length is less than 3 characters");
>     }
> }
> </script>
```

# Browser object

- ▶ The navigator object contains information about the browser.
- ▶ `txt = "<p>Browser CodeName(initial name): " + navigator.appCodeName + "</p>";`
- ▶ `txt+= "<p>Browser Name: " + navigator.appName + "</p>";`
- ▶ `txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";`
- ▶ `txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";`
- ▶ `txt+= "<p>Browser Language: " + navigator.language + "</p>";`
- ▶ `txt+= "<p>Browser Online: " + navigator.onLine + "</p>";`
- ▶ `txt+= "<p>Platform: " + navigator.platform + "</p>";`
- ▶ `txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";`
- ▶ `txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>";`

# Date Object

- ▶ `new Date()` // current date and time – construct an empty date object.
  - ▶ `new Date(milliseconds)` // milliseconds since 1970/01/01
  - ▶ `new Date(String)` – construct a date object based upon the contents of a text string.
  - ▶ `new Date(year, month, day, hours, minutes, seconds, milliseconds)`
- 

- ▶ `<script>`
  - ▶ `var d=new Date();`
  - ▶ `document.write(d);`
  - ▶ `var d5 = new Date(34354545)`
  - ▶ `var d1 = new Date("October 13, 1975 11:13:00")`
  - ▶ `var d2 = new Date(79,5,24)`
  - ▶ `var d3 = new Date(79,5,24,11,33,0)`
  - ▶ `document.write("<br>" + d1);`
  - ▶ `document.write("<br>" + d2);`
  - ▶ `document.write("<br>" + d3);`
  - ▶ `document.write("<br>" + d5);`
  - ▶ `</script>`
- 



- ▶ `x.setFullYear(2100,0,14);`
- ▶ `var d=new Date();`
- ▶ `var d=new Date();` – Mon Jan 20 2014 15:56:52 GMT-0800 (Pacific Standard Time)
- ▶ `d.getFullYear()` – 2014
- ▶ `d.getHours();` – 15
- ▶ `d.getMilliseconds();` – 483
- ▶ `d.getMinutes();` – 59
- ▶ `d.getMonth();` – 0 [0=January, 1=February etc.]
- ▶ `d.getTime();` – 1390262429129
- ▶ `d.setHours(21)` – Mon Jan 20 2014 21:01:54 GMT-0800 (Pacific Standard Time)

# The window object

- ▶ The window object represents an open window in a browser.
- ▶ If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

# Cookies

- ▶ Cookies are data, stored in small text files, on your computer.
- ▶ When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- ▶ Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his/her name.
- ▶ Cookies are saved in name–value pairs like:
  - username = John Doe

- ▶ When a browser requests a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

# Create a Cookie with JavaScript

- ▶ JavaScript can create, read, and delete cookies with the `document.cookie` property.
- ▶ With JavaScript, a cookie can be created like this:
  - `document.cookie = "username=John Doe";`
- ▶ You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:
  - `document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";`
- ▶ With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.
  - `document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";`

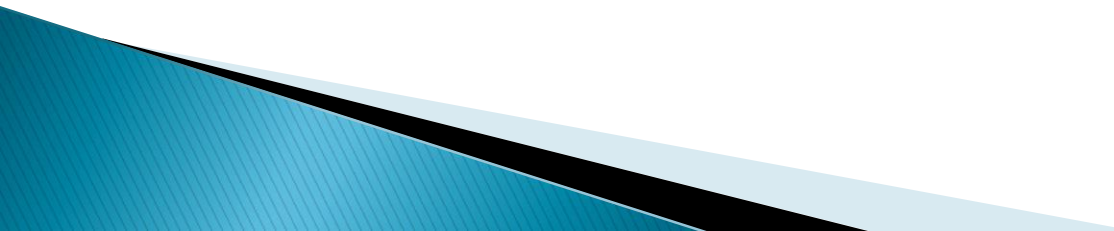
# Read a Cookie with JavaScript

- ▶ With JavaScript, cookies can be read like this:
- ▶ `var x = document.cookie;`
- ▶ `document.cookie` will return all cookies in one string much like: `cookie1=value;`  
`cookie2=value; cookie3=value;`

# Change a Cookie with JavaScript

- ▶ With JavaScript, you can change a cookie the same way as you create it:
- ▶ `document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";`
- ▶ The old cookie is overwritten.

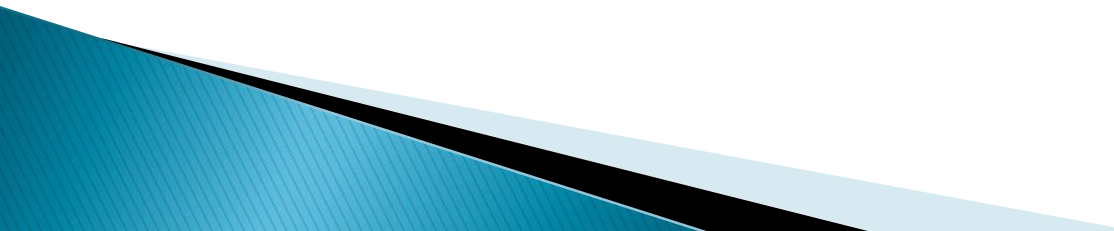
# Delete a Cookie with JavaScript

- ▶ Deleting a cookie is very simple.
  - ▶ You don't have to specify a cookie value when you delete a cookie.
  - ▶ Just set the expires parameter to a passed date:
  - ▶ `document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;"`;
  - ▶ You should define the cookie path to ensure that you delete the right cookie.
  - ▶ Some browsers will not let you delete a cookie if you don't specify the path.
- 

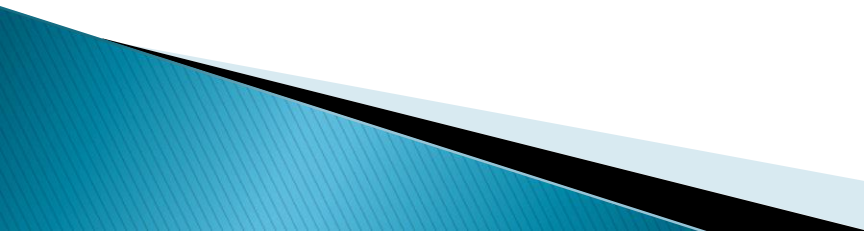


# A Function to Set a Cookie

```
▶ function setCookie(cname, cvalue, exdays) {  
    var d = new Date();  
    d.setTime(d.getTime() +  
    (exdays*24*60*60*1000));  
    var expires = "expires=" + d.toUTCString();  
    document.cookie = cname + "=" + cvalue + ";" +  
    expires + ";path=/";  
}
```



# Validation

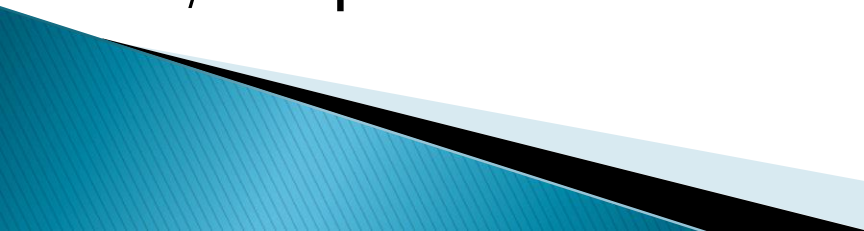
- ▶ JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.
  - ▶ Basic Validation – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
  - ▶ Data Format Validation – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.
- 

```
<script type = "text/javascript">
<!--
// Form validation code will come here.
function validate() {

    if( document.myForm.Name.value == "" ) {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
    }
    if( document.myForm.Email.value == "" ) {
        alert( "Please provide your Email!" );
        document.myForm.Email.focus() ;
        return false;
    }
    if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 ) {

        alert( "Please provide a zip in the format #####" );
        document.myForm.Zip.focus() ;
        return false;
    }
    if( document.myForm.Country.value == "-1" ) {
        alert( "Please provide your country!" );
        return false;
    }
    return( true );
}
//-->
</script>
```

```
▶ <script type = "text/javascript">
▶   <!--
▶     function validateEmail() {
▶       var emailID = document.myForm.EMail.value;
▶       atpos = emailID.indexOf("@");
▶       dotpos = emailID.lastIndexOf(".");
▶
▶       if (atpos < 1 || ( dotpos - atpos < 2 )) {
▶         alert("Please enter correct email ID")
▶         document.myForm.EMail.focus() ;
▶         return false;
▶       }
▶       return( true );
▶     }
▶   //-->
▶ </script>
```



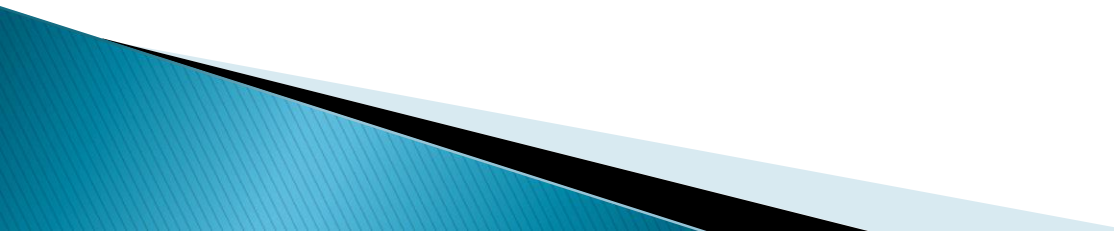
# Messages and Confirmation

- ▶ Prompt("string","string")
- ▶ Confirm("string")
- ▶ Alert("string")

# Status Bar

- ▶ Status bar usually displays helpful information about the operation of the browser
- ▶ Text strings can be displayed in the status bar using a function.

Eg : `self.status="test message";`



# Writing to a different frame

- ▶ `<frameset rows="40%,*" border="no">`
- ▶ `<frame src="top.html" name="top">`
- ▶ `<frame src="bottom.html" name="bottom">`
- ▶ `</frameset>`

- ▶ `<html>`
- ▶ `<head>`
- ▶ `<script language="javascript" src="js_ex.js"> </script>`
- ▶ `</head>`
- ▶ `<body >`
- ▶ `<h1 align="center" id="b1"> passing values from one frame to another</h1><br><br>`
- ▶ `<form name="f1" >`  
Enter the color : `<input type="text" name="txt1" id="txt1">`  
`<input type="button" value="Click Here!" onClick="fn1();">`
- ▶ `</form>`
- ▶ `</body>`
- ▶ `</html>`



## Bottom.html

- ▶ <html>
- ▶ <head>
- ▶ <script language="javascript" src="js\_ex.js"></script>
- ▶ </head>
- ▶ <body>
- ▶ <h1>LOOK HERE TO SEE THE CHANGES </H1>
- ▶ <p id="p1">simple text</p>
- ▶ <form name="f2">
- ▶ <table id="t1">
- ▶ <tr><td>Result : </td><td><input type="text" name="txt2 id="txt2"></td></tr>
- ▶ </table>
- ▶ </form>
- ▶ </body>
- ▶ </html>

## Js\_ex.js

- ▶ function fn1() //it works only in mozilla
- ▶ {

```
//alert(document.forms[0].elements[0].value);  
//alert(parent.frames['bottom'].document.forms[0].elements[0].value);  
//document.forms[0].elements[0].style.color="red";
```

- ▶ var btm=parent.frames['bottom'].document;
- ▶ var top=document.forms[0].elements;
- ▶ var bg=top.txt1.value;

- ▶ btm.forms[0].elements[0].value=bg;
- ▶ btm.forms[0].elements[0].style.color=bg;

- ▶

- ▶ }

# Validation example

```
<html>
<script>
function fn_validate()
{
name=document.getElementById("txtname").value;
age=document.getElementById("txtage").value;
RE_name=new RegExp("^[A-Z][a-zA-Z_.']+$","g");
RE_age=new RegExp("^[0-9]+[0-9]$");

//validation for name
if(name=="")
{
    alert("Name should not be empty");
    return false;
}
else if (!name.match(RE_name))
{
    alert("Name should be valid");
    return false;
}

//validation for age
else if(age=="")
{
    alert("Age should not be empty");
    return false;
}
else if (!age.match(RE_age))
{
    alert("Age should be valid");
    return false;
}

    return true;
}
</script>
```

```
<body>
```

```
<h1 align="center">Data validation example</h1><br><br><br>
```

```
<form onsubmit="return fn_validate()" action="next.html">
```

```
<table cellpadding="10px" border=1 bgcolor="blue" align="center">  
<tr>
```

```
    <td><b>Name:</b></td> <td><input type="text" id="txtname" ></td>  
</tr>
```

```
<tr>  
    <td><b>Age:</b></td> <td> <input type="text" id="txtage"></td>  
</tr>
```

```
<tr>  
    <td colspan="2" align="center">  
        <input type="submit" value="Submit">  
    </td>  
</tr>
```

```
</table>
```

```
</form>
```

```
</body>
```

```
</html>
```