# JAVASCRIPT

# JAVASCRIPT

- JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera.

# WHAT IS JAVASCRIPT?

- A lightweight programming language ("scripting language")
  - ✓ used to make web pages interactive
  - ✓ insert dynamic text into HTML (ex: user name)
  - ✓ react to events (ex: page load user click)
  - ✓ get information about a user's computer (ex: browser type)
  - ✓ perform calculations on user's computer (ex: form validation)
- A web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities.

# Where to Put your Scripts

- You can have any number of scripts
- Scripts can be placed in the HEAD or in the BODY
  - ✓ In the HEAD, scripts are run before the page is displayed
  - ✓ In the BODY, scripts are run as the page is displayed
- In the HEAD is the right place to define functions and variables that are used by scripts within the BODY

# Linking to a JavaScript file: script

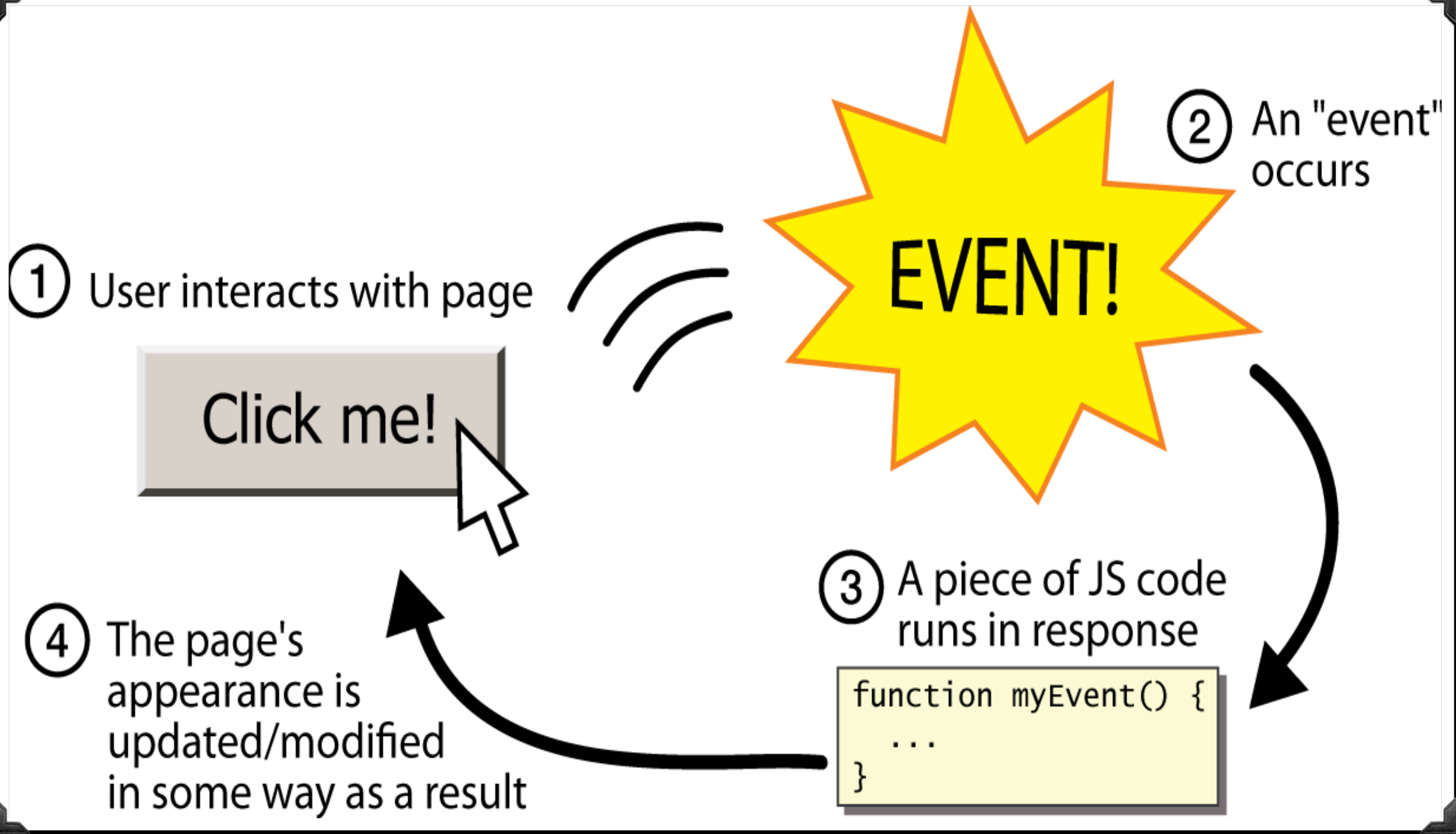
- Script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS).

<script src="filename" type="text/javascript"></script>

<script type = "text/javascript"> JS Code </script>

*HTML*

# Event-driven programming



# JavaScript Statement

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<script type="text/javascript">
document.write('This is my first JavaScript
    Page');
</script>
</body>
</html>
```

# JavaScript Statement

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<script type="text/javascript">
    document.write("<h1>This is my
    first JavaScript Page</h1>");
</script>
</body>
</html>
```

HTML written inside JavaScript

# JavaScript Statements - Alert

- 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.

```
<head>
<title>JS</title>
<script>
alert("Welcome to JS world!")
</script>
</head>
```

# JavaScript Statements - Confirm

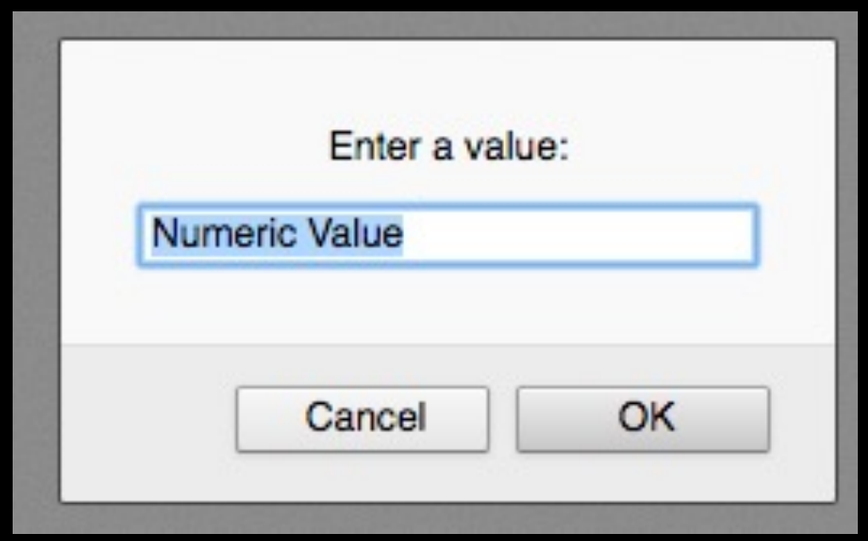
- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
<script>
confirm("Welcome to JS world!")
</script>
```

# JavaScript Statements - Prompt

- 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.

```
<script>
x=prompt("Enter a value:","Numeric Value");
document.write("welcome: ", + x);
</script>
```



# JAVASCRIPT VARIABLES

# JAVASCRIPT VARIABLES

- There are 3 ways to declare a JavaScript variable:
  - ✓ var
  - ✓ let
  - ✓ const

- **Var** - var declarations are globally scoped or function/locally scoped.
- **Let** - let is block scoped. A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block.
- **Const** - Variables declared with the const maintain constant values. const declarations share some similarities with let declarations.

# VAR - JS VARIABLE

```
<script>
    var myName = "Jayakumar";
    document.write(myName); // Jayakumar
    //re-declaring the var VALUE is ALLOWED
    myName = "Jayakumar Sadhasivam"
    document.write("<br><br>");
    document.write(myName);
    // redeclaring the VAR variable IS ALLOWED
    var myName = "JavaScript"
    document.write("<br><br>");
    document.write(myName);
</script>
```

Jayakumar

Jayakumar Sadhasivam

JavaScript

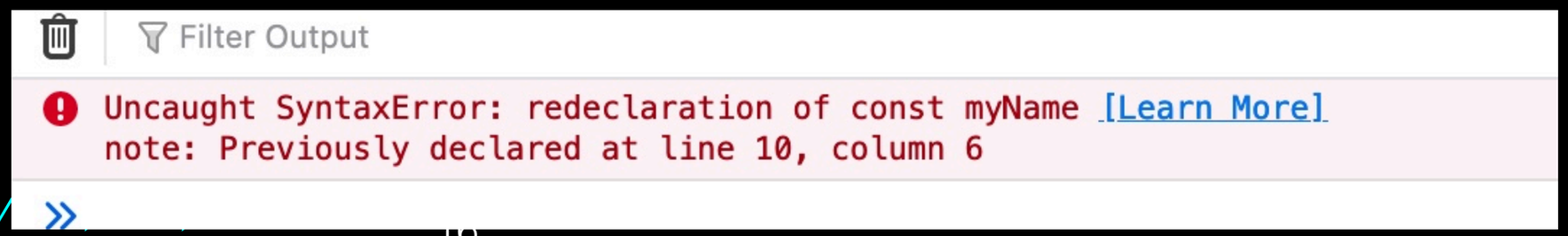
# LET - JS VARIABLE

```
<script>
    let myName = "Jayakumar";
    document.write(myName); // Jayakumar
    //re-declaring the LET VALUE is ALLOWED
    myName = "Jayakumar Sadhasivam"
    document.write("<br><br>");
    document.write(myName);
    // redeclaring the LET variable IS NOT ALLOWED
    let myName = "JavaScript"
    document.write("<br><br>");
    document.write(myName);
</script>
```

```
Uncaught SyntaxError: redeclaration of let myName
note: Previously declared at line 10, column 4
```

# CONST - JS VARIABLE

```
<script>
    const myName = "Jayakumar";
    document.write(myName); // Jayakumar
    //re-declaring the CONST VALUE is NOT ALLOWED
    myName = "Jayakumar Sadhasivam"
    document.write("<br><br>");
    document.write(myName);
    //redeclaring the CONST variable IS NOT ALLOWED
    const myName = "JavaScript"
    document.write("<br><br>");
    document.write(myName);
</script>
```

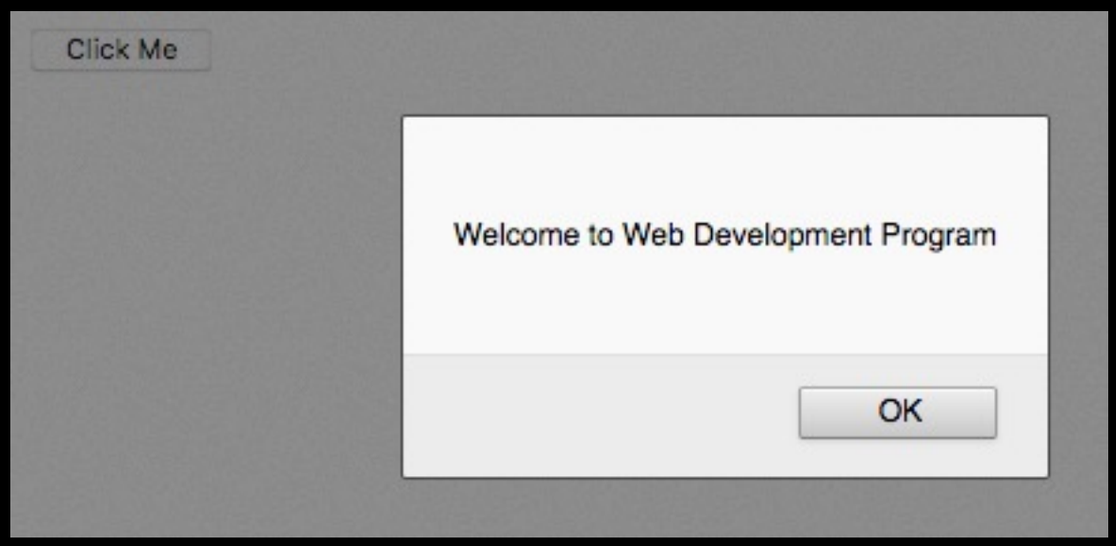


# JavaScript Functions

- JavaScript code is executed during the page loading or when the browser fires an event
  - ✓ All statements are executed at page loading
  - ✓ Some statements just define functions that can be called later
- Function calls or code can be attached as "event handlers" via tag attributes
  - ✓ Executed when the event is fired by the browser

# JavaScript Functions

```
<script>
function abc() {
alert("Welcome to Web Development Program")
}
</script>
</head>
<body>
<input type="button" value="Click Me"
     onclick="abc()">
</body>
```



# Display using console.log() Method

- The console.log() method writes a message to the console.
- The console is useful for testing purposes.

```
<body>
 <script>
     console.log("Welcome to JavaScript");
 </script>
</body>
```

# JavaScript Display Possibilities

- JavaScript can "display" data in different ways:
  - ✓ Writing into an alert box, using **window.alert()**
  - ✓ Writing into the HTML output using **document.write()**
  - ✓ Writing into an HTML element, using **innerHTML**

# Window.alert()

```
<body>
<h1><font color="red">
        using Windows Alert Function
</font></h1>
<script>
window.alert(5 + 6);
</script>
</body>
```

# document.write()

```
<body>
<h1>
<font color="blue">
        using document write Function
</font>
</h1>
<script>
document.write("<h1>",5 + 6,"</h1>");
</script>
</body>
```

using document write Function

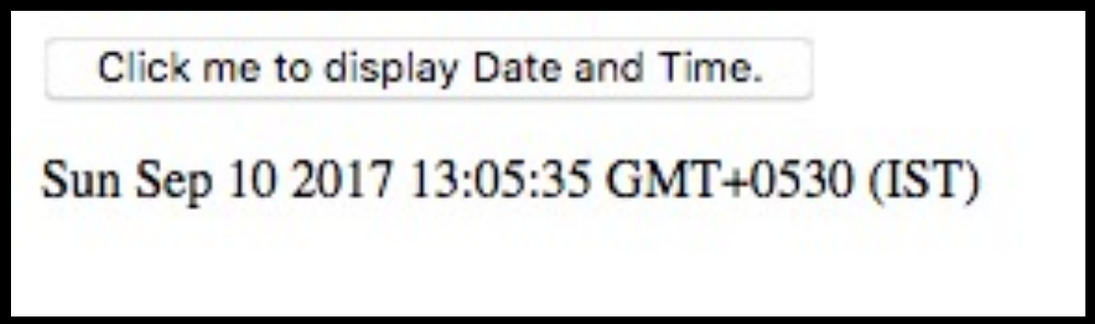
11

# innerHTML

```
<body>
<h1><font color="blue">
        using innerHTML Function
</font></h1>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML
    = 5 + 6;
</script>
</body>
```

# JavaScript innerHTML

```
<html>
<body>
<script type="text/javascript"></
    script>
<button type="button"
    onclick="document.getElementById('de
    mo').innerHTML = Date()">
Click me to display Date and Time.
</button>
<p id="demo"></p>
</script>
</body>
</html>
```



# innerHTML and CSS

```
<p id="abcd">JavaScript can change the style of an HTML element.</
        p>
<button onclick="abc()">Click Me!</button>
<script>
    function abc() {
        document.getElementById('abcd').style.fontSize = '35px';
        document.getElementById('abcd').style.color = 'red';
  }
</script>
```

# JavaScript Data Types

- Primitive Data Types
  - ✓ Number: integer & floating-point numbers
  - ✓ Boolean: true or false
  - ✓ String: a sequence of alphanumeric characters
- Composite data types (or Complex data types)
  - ✓ Object: a named collection of data
  - ✓ Array: a sequence of values (an array is actually a predefined object)
- Special data types
  - ✓ Null: the only value is "null" – to represent nothing.
  - ✓ Undefined: the only value is "undefined" – to represent the value of an unintialized variable

# JavaScript Data Types

- String
  - ✓ Use escaped character sequence to represent special character (e.g.: \", \n, \t)

```
var myName = "You can use both single or double quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

# JavaScript Operators and Constructs

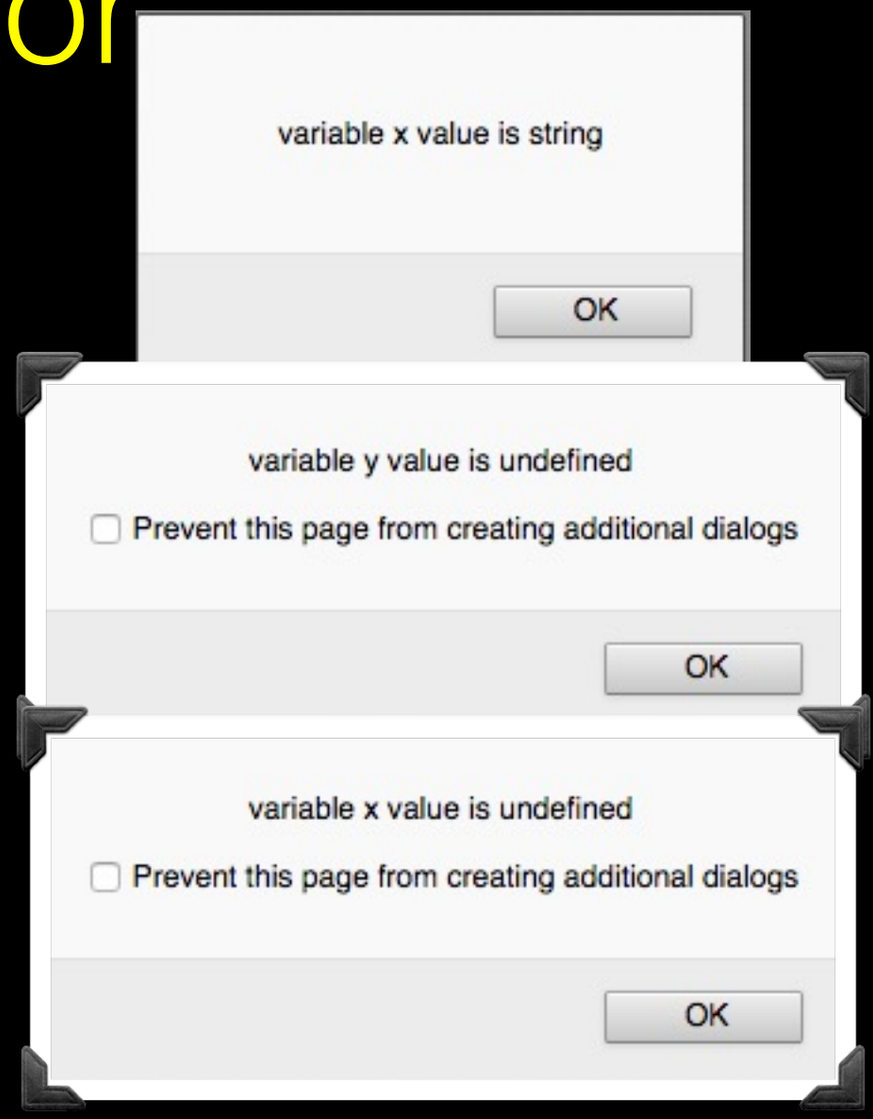
- JavaScript has most of the operators
  - ✓ Arithmetic(+,-,*,/,%)
  - ✓ Assignment(=,+=,-=,*=/=,%=,++,—)
  - ✓ Logical(&&,||,!)
  - ✓ Comparison(<,>,<=,>=,==)
- Constructs:
  - ✓ if, else,
  - ✓ while, for,
  - ✓ switch case

# JavaScript Events

- JavaScript can be made to respond to user events
- Common Events:
  - ✓ onload and onunload : when a page is first visited or left.
  - ✓ onfocus, onblur, onchange : events pertaining to form elements
  - ✓ onsubmit : when a form is submitted
  - ✓ onmouseover, onmouseout : for "menu effects"

# typeof Operator

```
<script type="text/javascript">
var x="hello", y;
alert("variable x value is " + typeof x);
alert("variable y value is " + typeof y);
alert("variable z value is " + typeof z);
</script>
```




- An unary operator that tells the type of its operand.
- Returns a string which can be "number", "string", "boolean", "object", "function", “undefined", and “null”.
- An array is internally represented as an object.

# Operators

Arithmetic operators

+, - , %, /, %

Post/pre increment/decrement

++,, --

Comparison operators

==, !=, >, >=, <, <=

===, !== (Strictly equals and strictly not equals)

i.e., Type and value of operand must match / must not match

# == VS ===

```
// Type conversion is performed before comparison
var v1 = ("5" == 5);  // true

// No implicit type conversion.
// True if only if both types and values are equal
var v2 = ("5" === 5); // false

var v3 = (5 === 5.0); // true

var v4 = (true == 1); // true (true is converted
    to 1)

var v5 = (true == 2); // false (true is converted
    to 1)

var v6 = (true == "1") // true
```

# Logical Operators

- ! – Logical NOT

- && – Logical AND
  - ✓ OP1 && OP2
  - ✓ If OP1 is true, expression evaluates to the value of OP2.
  - ✓ Otherwise the expression evaluates to the value of OP1.
  - ✓ Results may not be a boolean value.

- || – Logical OR
  - ✓ OP1 || OP2
  - ✓ If OP1 is true, expression evaluates to the value of OP1.
  - ✓ Otherwise the expression evaluates to the value of OP2.

# Logical Operators Example

```
var tmp1 = null && 1000; // tmp1 is null
var tmp2 = 1000 && 500;  // tmp2 is 500
var tmp3 = false || 500; // tmp3 is 500
var tmp4 = "" || null;  // tmp4 is null
var tmp5 = 1000 || false; // tmp5 is 1000
```

```
// If foo is null, undefined, false, zero,
   NaN, or an empty string, then set foo to
   100.
foo = foo || 100;
```

# Everything is Object

- Every variable can be considered as object
  - ✓ For example strings and arrays have member functions:

```
var test = "some string";
alert(test[7]); // shows letter 'r'
alert(test.charAt(5)); // shows letter 's'
alert("test".charAt(1)); //shows letter 'e'
alert("test".substring(1,3)); //shows 'es'
```

```
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7); // appends 7 to end of array
arr=[1,3,4,7]
alert (arr[3]); // shows 7
```

# JavaScript Syntax

- The JavaScript syntax is similar to C# and Java
  - ✓ Operators (+, %, =, !=, &&, ++, …)
  - ✓ Conditional statements (if, else)
  - ✓ Loops (for, while)
  - ✓ Arrays (my_array[]) and associative arrays (my_array['abc'])
  - ✓ Functions (can return value)
  - ✓ Function variables (like the C# delegates)

# Decision Statement

# The if Statement

- Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.
- if (*condition*) {
  *block of code to be executed if the condition is true*
  }
- Make a "Good day" greeting if the hour is less than 18:00:
- if (hour < 18) {
  greeting = "Good day";
  }

# The else Statement

- Use the **else** statement to specify a block of code to be executed if the condition is false.
- if (hour < 18) {
  greeting = "Good day";
  } else {
  greeting = "Good evening";
  }

# The else if Statement

- if (*condition1*) {
  *block of code to be executed if condition1 is true*
  } else if (*condition2*) {
  *block of code to be executed if the condition1 is false and condition2 is true*
  } else {
  *block of code to be executed if the condition1 is false and condition2 is false*
  }

# Example

- If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":
- if (time < 10) {
  greeting = "Good morning";
  } else if (time < 20) {
  greeting = "Good day";
  } else {
  greeting = "Good evening";
  }

# The JavaScript Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.
- switch(*expression*) {

```
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        code block
}
```

# Evaluation procedure

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break Keyword When JavaScript reaches a **break** keyword, it breaks out of the switch block.
- A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.
- The **default** keyword specifies the code to run if there is no case match:

- The getDay() method returns the weekday as a number between 0 and 6.
- (Sunday=0, Monday=1, Tuesday=2 ..)
- This example uses the weekday number to calculate the weekday name:
- switch (new Date().getDay()) {

```
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
```

# JavaScript While Loop

- while (*condition*) {
  *code block to be executed*
  }
- while (i < 10) {
  text += "The number is " + i;
  i++;
  }

# The Do/While Loop

- do {
  *code block to be executed*
  }
  while (*condition*);
- do {
  text += "The number is " + i;
  i++;
  }
  while (i < 10);

# For loop

- for (*statement 1*; *statement 2*; *statement 3*) {
  *code block to be executed*
  }
- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.
- for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
  }

# The For/In Loop

- The JavaScript for/in statement loops through the properties of an object:
- var person = {fname:"John", lname:"Doe", age:25};

```
var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

# The Break Statement

- The break statement can also be used to jump out of a loop.
- The **break statement** breaks the loop and continues executing the code after the loop
- for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
  }

# Continue Statement

- The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 3:
- for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
  }

# Object

- JavaScript is an Object Based Programming language. An Object Based Programming language allows you to define your own objects and make your own variable types.
- An object is just a special kind of data. An object has properties and methods.

```
var val = new String(string);
```

- Properties are the values associated with an object.

```
var txt="Hello World!";
document.write(txt.length);
```

- Methods are the actions that can be performed on objects.

```
var str="Hello world!";
document.write(str.toUpperCase());
```

# String object

```
<script>
//The String object is used to manipulate a stored piece of text.

var txt="Hello world!";
document.write(txt.length); //12
document.write(txt.toUpperCase()); //HELLO WORLD!
document.write(txt.match("world") ); //world
document.write(txt.match("World") ) ;//null W-Caps
document.write(txt.indexOf("world")); //6

var str="Visit Microsoft!";
document.write(str.replace("Microsoft","CTS"));
        //Visit CTS!
</script>
```

```
12
HELLO WORLD!
world
null
6
Visit CTS!
```

```
let txt = "Hello World!";
document.write("<p>Big: " + txt.big() + "</p>");
document.write("<p>Small: " + txt.small() + "</p>");
document.write("<p>Bold: " + txt.bold() + "</p>");
document.write("<p>Italic: " + txt.italics() + "</p>");
document.write("<p>Strike: " + txt.strike() + "</p>");
document.write("<p>Fontcolor:"+txt.fontcolor("red")+"</p>");
document.write("<p>Fontsize: "+ txt.fontsize(10)+"</p>");
document.write("<p>Subscript: " + txt.sub() + "</p>");
document.write("<p>Superscript: " + txt.sup() + "</p>");
document.write("<p>Link: " + txt.link("http://jayakumars.in") + "</p>");
document.write("<p>Blink: " + txt.blink() + " (does not work in IE, Chrome, or
      Safari)</p>");
```

Big: Hello World!

Small: Hello World!

Bold: **Hello World!**

Italic: *Hello World!*

Strike: ~~Hello World~~!

Fontcolor:Hello World!

Fontsize: Hello World!

Subscript: Hello World!

Superscript: Hello World!

Link: Hello World!

Blink: Hello World! (does not work in IE, Chrome, or Safari)

- charAt(index) --> Returns the character at the specified index
- concat(str1,str2...)
- lastIndexOf() —> method returns the position of the last occurrence of a specified value in a string.
- slice(bindx,eindx) --> extracts a section of a string and returns a new string
- split(separator) --> a String object into an array of strings by separating the string into substrings.
- substr(start,length)
- substring(indx,indx)
- toLowerCase()
- search(regexp)
- localeCompare() --> returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.

# Date object

- The Date object is used to work with dates and times.
- Date objects are created with **new Date()**.
- There are four ways of instantiating a date
  - ✓ `var d = new Date();`
  - ✓ `var d = new Date(milliseconds);`
  - ✓ `var d = new Date(dateString);`
  - ✓ `var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);`

# Date - Shows Current Time

```
<body>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
</body>
```

# Date - Shows Predefined Time

```
<body>
<p id="demo"></p>
<script>
var d = new Date("January 20, 2024
    11:13:00");
document.getElementById("demo").inner
    HTML = d;
</script>
</body>
```

Sat Jan 20 2024 11:13:00 GMT+0530 (IST)

# Date Object

The Date object is used to work with dates and times.

- var d = new Date();
- getDate() - Returns the day of the month (from 1-31)
- getDay() - Returns the day of the week (from 0-6)
- getFullYear() - Returns the year
- getHours() - Returns the hour (from 0-23)
- getMilliseconds() - Returns the milliseconds (from 0-999)
- getMinutes() - Returns the minutes (from 0-59)
- getMonth() - Returns the month (from 0-11)
- getSeconds() - Returns the seconds (from 0-59)
- getTime() - Returns the number of milliseconds since midnight Jan 1 1970, and a specified date
- getTimezoneOffset() - Returns the time difference between UTC time and local time, in minutes

- `toLocaleDateString()` – Returns the date portion of a `Date` object as a string, using locale conventions
- `toLocaleTimeString()` – Returns the time portion of a `Date` object as a string, using locale conventions
- `toLocaleString()` – Converts a `Date` object to a string, using locale conventions
- `toString()` – Converts a `Date` object to a string
- `toTimeString()` – Converts the time portion of a `Date` object to a string
- `toUTCString()` – Converts a `Date` object to a string, according to universal time
- `UTC()` – Returns the number of milliseconds in a date since midnight of January `1`, `1970`, according to UTC time

# Array

- An array is represented by the **Array** object. To create an array of N elements, you can write

**var myArray = new Array(N);**

- Index of array runs from 0 to N-1.
- Can store values of different types
- Property "**length**" tells the # of elements in the array.
- Consists of various methods to manipulate its elements. e.g., **reverse(), push(), concat()**, etc

# Array Example

```
<script>
var Car = new Array(3);
Car[0] = "Ford";
Car[1] = "Toyota";
Car[2] = "Honda";
// Create an array of three elements with
    initial values
var Car2 = new Array("Ford", "Toyota",
    "Honda");

// Create an array of three elements with
    initial  values
var Car3 = ["Ford", "Toyota", "Honda"];
</script>
```

# Array Example

```
<script>
// An array of 3 elements, each element is undefined
var tmp1 = new Array(3);

// An array of 3 elements with initial values
var tmp2 = new Array(10, 100, -3);

// An array of 3 elements with initial values
// of different types
var tmp3 = new Array(1, "a", true);

// Makes tmp3 an array of 10 elements
tmp3.length = 10;  // tmp[3] to tmp[9] are undefined.

// Makes tmp3 an array of 100 elements
tmp3[99] = "Something";
// tmp[3] to tmp[98] are undefined.
</script>
```

# Adding Array Element

The easiest way to add a new element to an array is using the push method:

```
<script>
var fruits =
    ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");
// adds a new element (Lemon) to fruits
var fruits =
    ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";
</script>
```

# Associated Arrays

- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does not support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.

```
<script>
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length; // person.length will return 3
var y = person[0];     // person[0] will return “John"
</script>
```

# Associated Arrays

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length; //person.length will return 0
var y = person[0]; //Person[0] will return undefined
var z = person["firstName"];    //will return John
```

- If you use a named index, JavaScript will redefine the array to a standard object.
- After that, all array methods and properties will produce incorrect results.

# Array Push

- The push() method adds a new element to an array (at the end):
- The push() method returns the new array length:

```
var fruits =
    ["Banana", "Orange", "Apple", "Mango"];

var x = fruits.push("Kiwi");
                //  the value of x is 5
```

# Array Pop

- The pop() method removes the last element from an array
- The pop() method returns the value that was "popped out”

```
var fruits =
    ["Banana", "Orange", "Apple", “Mango"];

fruits.pop();      // Removes the last
    element ("Mango") from fruits
```

# Array Unshift

- The unshift() method adds a new element to an array (at the beginning), and “unshifts" older elements
- The unshift() method returns the new array length.

```
var fruits =
    ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

# Array Shift

- The shift() method removes the first array element and "shifts" all other elements to a lower index

```
var fruits =
    ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
// Removes the first element "Banana" from
    fruits
```

# Changing Elements

- Array elements are accessed using their index number
- Array indexes start with 0. [0] is the first array element, [1] is the second, [2] is the third

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
    fruits[0] = "Kiwi";
// Changes the first element of fruits to "Kiwi"
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
    fruits[fruits.length] = "Kiwi";
// Appends "Kiwi" to fruit
```

# Delete

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete
- Using delete may leave undefined holes in the array. Use pop() or shift() instead.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];

// Changes the first element in fruits to undefined
```

# Splicing an Array

- The splice() method can be used to add new items to an array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", “Kiwi");
```

- The first parameter (2) defines the position where new elements should be added (spliced in).
- The second parameter (0) defines how many elements should be removed.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

# Slicing an Array

- The slice() method slices out a piece of an array into a new array.
- The slice() method creates a new array. It does not remove any elements from the source array.

```
var fruits=
    ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1); // Orange
var citrus = fruits.slice(3); // Apple
var citrus = fruits.slice(1, 3);
    // Orange, Lemon, Apple
```

# Converting array into string

- The JavaScript method toString() converts an array to a string of (comma separated) array values.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.toString());
```

# Join

- The join() method also joins all array elements into a string.
- It behaves just like toString(), but in addition you can specify the separator:

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
document.write(fruits.join(" and “));
```

```
//Banana and Orange and Apple and Mango
```

# Combining arrays

- The concat() method creates a new array by concatenating two arrays

```
var num1 = [10,20];
var num2 = [30,40,50];
var num = num1.concat(num2);
```

# Sorting an Array

- The sort() method sorts an array alphabetically

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.sort());
                    // Sorts the elements of fruits
document.write(fruits.reverse());
// Reverses the order of the elements
```

Apple,Banana,Mango,Orange

Orange,Mango,Banana,Apple

# Numeric Sort

- By default, the sort() function sorts values as strings.
- This works well for strings ("Apple" comes before "Banana").
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".
- Because of this, the sort() method will produce incorrect result when sorting numbers.
- You can fix this by providing a compare function:

# Compare Function

- The purpose of the compare function is to define an alternative sort order.
- The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a-b}
```

- When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

- When comparing 40 and 100, the sort() method calls the compare function(40,100).
- The function calculates 40-100, and returns -60 (a negative value).
- The sort function will sort 40 as a value lower than 100.

```
var points = ["40", "100", "1", "5", "25", “10","304","203"];
points.sort(function(a, b){return a - b});
document.write(points);//1,5,10,25,40,100,203,304
```

1,5,10,25,40,100,203,304

# Find the Highest or Lowest Value

```
var points = ["40", "100", "1", "5", "25", “10"];
==========================================================
points.sort(function(a, b){return b - a});
// now points[0] contains the HIGHEST value

document.write(points); //100,40,25,10,5,1
==========================================================
points.sort(function(a, b){return a - b});
// now points[0] contains the LOWEST value

document.write(points); //1,5,10,25,40,100
```

100,40,25,10,5,1
1,5,10,25,40,100

# Exercise

- Assume that you are creating a new array called birds
  - How do you declare a new empty array?
  - Declare a new array with space allotted for 4 elements?
  - Declare a new array containing the names of 4 birds (note, some types of birds are robin, sparrow, hummingbird, eagle, crow)
  - Insert “robin” at the beginning.
  - Insert “peacock” at the end.
  - Delete last element in an array.
  - Delete second element in an array
  - Display only first two elements in an array.
  - Sort the elements in an array.
  - Join the array elements with “+”

# Null & Undefined

- An undefined value is represented by the keyword "**undefined**".
  - ✓ It represents the value of an uninitialized variable
- The keyword " **null** " is used to represent “nothing”
  - ✓ Declare and define a variable as “null” if you want the variable to hold nothing..
  - ✓ Avoid leaving a variable undefined.

# Type Conversion (To Boolean)

- The following values are treated as false
  - ✓ null
  - ✓ undefined
  - ✓ +0, -0, NaN (Not a Number)
  - ✓ "" (empty string)

# Type Conversion

- Converting a value to a number

```
var numberVar = someVariable – 0;
```

- Converting a value to a string

```
var stringVar = someVariable + "";
```

- Converting a value to a boolean

```
var boolVar = !!! someVariable ;
```

# Regular Expression

# Regular Expression

- An object which describes a sequence of characters that specify a pattern.
- This pattern matches against a string of text when performing search or replace.
- To create regular expression, the user can either go for
  - ✓ Using Literals
  - ✓ Using RegExp() constructor

# Using Literals

- To create a regular expression using literal, the user to assign the regular expression to a variable.
- Used to execute on compile time.
- Syntax:
  - **var varname = /pattern/options;**
    - ★ Pattern: providing the text to be used in the search
    - ★ Options: providing options for modifying search patterns.
      - ✳ Options
        - i - used to ignore case
        - g - used to match all occurrences of the pattern in the string. (Global Matching)
        - m - used to match over multiple lines.

# Using RegExp() Constructor

- creates a regular expression object.
- It is used to execute at runtime.
- Takes two arguments
  - ✓ String
  - ✓ Flag(optional such as i,g,m etc..)
- Syntax:
  - ✓ **var varname = new RegExp("string", "options");**

# Testing Regular Expression

- RegExp() object has two methods to test a match for a string.
  - ✓ test(): Tests for a match in the given string and either true or false.

```
Syntax: regvarname.test(string);
```

  - ✓ exec(): executes a search to find a match for a specified pattern in a string. If found returns string, otherwise returns null.

```
Syntax: regvarname.exec(string);
```

```
<body>
<script>
 var ip="JayaKumar";
 var r=/kum/;
 var op=r.exec(ip);
 alert(op);
 var r1=/kum/i;
 var op1=r1.test(ip);
 alert(op1);
</script>
</body>
```

# RE - String Methods

- Match()
  - ✓ Same as exec() used to search for a pattern in a string and returns an arrary of patterns that was matched.
  - ✓ If no match found returns null.
  - ✓ Syntax:

```
var varname = String.match(reg);
```

# RE - String Methods

- Search():
  - ✓ Used to search and return the position of the given pattern in the string.
  - ✓ Indexing starts at 0 and if pattern is not found return -1.
  - ✓ Syntax:

```
var varname = String.search(reg);
```

# RE - String Methods

- replace():
  - ✓ Used to search for a string and replace the string with another string.
  - ✓ If no match is found returns null.
  - ✓ Syntax:

```
var varname = String.replace(reg,”replacestring");
```

# Metacharacters

- These characters have a special meaning preceded by a blackslash.
- It is used to allow the user to control the search pattern in some way.
  - **.(dot)**: Used to find a single character.
  - \w :Used to find a word character from a-z, 0-9 and underscore.
  - \W : Used to find a non-word character. like $,%,?
  - \d : Used to find a digit from 0-9.
  - \D : Used to find a non-digit chacter.
  - \s : Used to find a white space character.
  - \S: Used to find a non white space character.
  - \b : Used to find a match at the beginning or end of a word.
  - \B : used to find a match not at the beginning or end of a word.

# Brackets

Used to specify the range of the characters such as alphabets, digits etc..

- **[a-z]** : used to match any no.of alphabets specified between a to z.
- **[0-9]** : Used to match any no.of digits specified between 0 to 9.
- **[^a-z]** : used to match any no.of alphabets not specified between a to z.
- **[^0-9]** : Used to match any no.of digits not specified between 0 to 9.
- **[a|b]** : used to find any alternative from the specified ones. Any type of alternative can be specified.

# Quantifiers

Used to indicate the frequency of the character sequences.

- **j+** : used to match string containing one or more occurrence of j.
- **j*** : used to match string containing zero or more occurrence of j.
- **j?** : used to match string containing zero or one occurrence of j.
- **j{x}** : used to match any string containing x characters.
- **j{x,y}** : used to match any string containing x to y characters.
- **j{x,}** : used to match any string containing alteast x characters.
- **j$** : used to match any string with the given pattern at the end of the string.
- **^j** : Used to match any string with the given pattern at the beginning of the string.

# Form Validation

- The mechanism used to validate the form on the client side itself without disturbing the server.
  - ✓ Form validation can be done
    - ✳ Checking for empty fields
    - ✳ Checking for alphabetics data
    - ✳ Checking for number data
    - ✳ Checking for length restriction.