JAVA SCRIPT

JavaScript is the programming language of HTML and the Web.

Programming makes computers do what you want them to do.

JavaScript is easy to learn.

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

JavaScript Can Change HTML Content

One of many HTML methods is **getElementById()**.

This example uses the method to "find" an HTML element (with id="demo"**)**, and changes the element content (**innerHTML**) to "Hello JavaScript":

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

<h1>What Can JavaScript Do?</h1>
<p  id="demo">JavaScript can change HTML content.</p>
<button type="button"
onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">
Click Me!</button>
</body>
</html>
```

JavaScript Can Change HTML Styles (CSS)

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can change the style of an HTML element.</p>
<script>
function myFunction() {
  var x = document.getElementById("demo");
  x.style.fontSize = "25px";
  x.style.color = "red";
}
</script>

<button type="button" onclick="myFunction()">Click Me!</button>
```

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

JavaScript Can Validate Data

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Can Validate Input</h1>
<p>Please input a number between 1 and 10:</p>
<input id="numb" type="number">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x, text;
  // Get the value of the input field with id="numb"
  x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

JavaScript can be placed in the <body> and the <head> sections of an HTML page.

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code, that can be executed when "asked" for.

For example, a function can be executed when an **event** occurs, like when the user clicks a button.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h1>JavaScript in Head</h1>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.
The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript in Body</h1>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</body>
</html>
```

External JavaScript

Scripts can also be placed in external files.

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the **file extension .js**.

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}

To use an external script, put the name of the script file in the src (source) attribute of the <script> tag:

<!DOCTYPE html>

<html>

<body>

<h1>External JavaScript</h1>

A Paragraph.

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

Note: myFunction is stored in an external file called "myScript.js".

<script src="myScript.js"></script>

</body>

</html>

External JavaScript Advantages

Placing JavaScripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

JavaScript Output

JavaScript does NOT have any built-in print or display functions.

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**.
- Writing into the browser console, using **console.log()**.

Window.alert()

- You can use an alert box to display data:

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

document.write()

For testing purposes, it is convenient to use **document.write()**:

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**: for example

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<button onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

```
<!DOCTYPE html>

<html><body>
```

```
<h1>My First Web Page</h1>
<p>My First Paragraph.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

To "display data" in HTML, (in most cases) you will set the value of an innerHTML property.

Using console.log()

In your browser, you can use the **console.log()** method to display data.

Activate the browser console with F12, and select "Console" in the menu.

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<p>
Activate debugging in your browser (Chrome, IE, Firefox) with F12, and select "Console" in the debugger menu.
</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

JavaScript Statements

JavaScript statements are separated by **semicolons**.

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Literals

The most important rules for writing fixed values are:

Numbers are written with or without decimals:

```
<html>
<body>
<h1>JavaScript Numbers</h1>
<p>Number can be written with or without decimals.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 10.50;
document.getElementById("demo").innerHTML = 'John Doe';
</script>
</body>
</html>
```

JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **define** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x is defined as a variable.
then, x is assigned the value of 6:</p>
<p id="demo"></p>
<script>
var  x;
x = 6;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

JavaScript Operators

JavaScript uses an **assignment operator** (=) to **assign** values to variables:

```
<html>
<body>
<h1>Assigning Values</h1>
<p>In JavaScript the = operator is used to assign values to variables.</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
document.getElementById("demo").innerHTML = x + y;
```

```
</script>
</body>
</html>
```
JavaScript uses **arithmetic operators** (+ - * /) to **compute** values:
e.g. document.getElementById("demo").innerHTML = (5 + 6) * 10;

JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 * 10 evaluates to 50:

```
<html>
<body>
<h1>JavaScript Expressions</h1>
<p>Expressions compute to values.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 * 10;
</script>
</body>
</html>
```
Expressions can also contain variable values:e.g. in above example
```
<script>
var x = 5;
document.getElementById("demo").innerHTML = x * 10;
</script>
```
The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

```
document.getElementById("demo").innerHTML = "John" + " "  + "Doe";
```

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes **//** or between **/*** and ***/** is treated as a **comment**.

JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, an underscore (_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

Numbers are not allowed as the first character.
This way JavaScript can easily distinguish identifiers from numbers.

JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

```
var length = 16;                               // Number
var lastName = "Johnson";                      // String
var cars = ["Saab", "Volvo", "BMW"];           // Array
var x = {firstName:"John", lastName:"Doe"};    // Object
```

The Concept of Data Types

JavaScript will also treat the first operand as a string.

```
var x = 3 + "Volvo";
```

1. <!DOCTYPE html>
<html>
<body>

<script>
var x = 3 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>

2. <script>
var x = **16 + 4 + "Volvo";**
document.getElementById("demo").innerHTML = x;
</script>
Result=20Volvo

3. <script>
var x = "Volvo" + 16 + 4;
document.getElementById("demo").innerHTML = x;
</script>

Result: Volvo164

JavaScript Has Dynamic Types

JavaScript has dynamic types. This means that the same variable can be used as different types:

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

Example

```
var answer = "It's alright";             // Single quote inside double quotes
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var carName1 = "Volvo XC60";
var carName2 = 'Volvo XC60';
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';
document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2 + "<br>" +
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
</script>
</body>
</html>
```

Result:

Volvo XC60

Volvo XC60

It's alright

He is called 'Johnny'

He is called "Johnny"

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;        // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

e.g. <script>

var x1 = 34.00;

var x2 = 34;

var y = 123e5;

var z = 123e-5;

**document.getElementById("demo").innerHTML = x1 + "
" + x2 + "
" + y + "
" + z**

</script>

JavaScript Booleans

Booleans can only have two values: true or false.

Example

```
var x = true;
var y = false;
```

Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

e.g. <script>

var cars = ["Saab","Volvo","BMW"];

```
document.getElementById("demo").innerHTML = cars[0];
</script>
```

JavaScript Objects

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

```
<script>
var person = {
   firstName : "John",
   lastName  : "Doe",
   age       : 50,
   eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

Result: John is 50 years old.

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

The typeof Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable:

e.g.

```
<script>
document.getElementById("demo").innerHTML =
typeof "john" + "<br>" +
typeof 3.14 + "<br>" +
typeof false + "<br>" +
typeof [1,2,3,4] + "<br>" +
typeof {name:'john', age:34};
</script>
```

Result:

string
number
boolean
object
object

Undefined

In JavaScript, a variable without a value, has the value **undefined**. The typeof is also **undefined**.

Example

```
var person;
```

<script>
var person;
document.getElementById("demo").innerHTML =
**person + "
" + typeof person;**
</script>

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

e.g. `person = undefined;`

<script>

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

var person = undefined;

document.getElementById("demo").innerHTML =

person + "
" + typeof person;

</script>

Empty Values

An empty value has nothing to do with undefined.

An empty string variable has both a value and a type.

e.g. `var car = "";`

<script>

var car = "";

document.getElementById("demo").innerHTML =

"The value is: " +

car + "
" +

"The type is:" + typeof car;

</script>

Result

The value is:
The type is:string

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

You can empty an object by setting it to null:

e.g. `var person = null;`

<script>

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

var person = null;

document.getElementById("demo").innerHTML = typeof person;

</script>

Result: object

You can also empty an object by setting it to undefined:

e.g. var person=undefined;

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.The purpose of code blocks is to define statements to be executed together.One place you will find statements grouped together in blocks, are in JavaScript functions:

```
e.g. function myFunction() {
    document.getElementById("demo").innerHTML = "Hello Dolly.";
    document.getElementById("myDIV").innerHTML = "How are you?";
}
```

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top

debugger	Stops the execution of JavaScript, and calls (if available) the debugging fur
do ... while	Executes a block of statements, and repeats the block, while a condition is
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
Return	Exits a function
Switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
Var	Declares a variable

JavaScript Functions

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

A JavaScript function is executed when "something" invokes it (calls it).

```
<!DOCTYPE html>
<html>
<body>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
   return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

JavaScript Function Syntax

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses**()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***

The code to be executed, by the function, is placed inside curly brackets: **{}**

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

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** received by the function when it is invoked.

Inside the function, the arguments are used as local variables

Function Return

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

```
<!DOCTYPE html>
<html>
<body>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
```

</html>

Functions Used as Variables

In JavaScript, you can use functions the same way as you use variables.

You can use:

```
var text = "The temperature is " + toCelsius(32) + " Centigrade";
```

Instead of:

```
var x = toCelsius(32);
var text = "The temperature is " + x + " Centigrade";
```

<!DOCTYPE html>

<html>

<body>

** **

<script>

document.getElementById("demo").innerHTML =

"The temperature is " + toCelsius(32) + " Centigrade";

function toCelsius(fahrenheit) {

return (5/9) * (fahrenheit-32);

}

</script>

</body></html>

Conditional Statements

1. if

2. if.....else

3.else.....if

4.switch

The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

<!DOCTYPE html>

<html>

<body>

** Display "Good day!" if the hour is less than 18:00: **

** Good Evening! **

<script>

if (new Date().getHours() < 18) {

document.getElementById("demo").innerHTML = "Good day!";

}

</script>

</body>

</html>

The if.... else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

```
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
}
```

<!DOCTYPE html>

<html>

<body>

** Click the button to display a time-based greeting: **

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

** **

<script>

function myFunction() {

var hour = new Date().getHours();

var greeting;

```
  if (hour < 18) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>
</body></html>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        default code block
}
```

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

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

<script>
var day;
switch (new Date().gethours()) {
  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";
    break;
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script></body></html>
```

Sometimes, in a switch block, you will want different cases to use the same code

```
<html>
<body>
<p id="demo"></p>
<script>
var text;
switch (6) {
  case 1:
```

```
    case 2:
    case 3:
    default:
      text = "Looking forward to the Weekend";
      break;
    case 4:
    case 5:
      text = "Soon it is Weekend";
      break;
    case 0:
    case 6:
      text = "It is Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>
```

Event handling

They are JavaScript code that are not added inside the <script> tags, but rather, inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc. The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

onclick:	Use this to invoke JavaScript upon clicking (a link, or form boxes)
onload:	Use this to invoke JavaScript after the page or an image has finished loading.
onmouseover:	Use this to invoke JavaScript if the mouse passes by some link
onmouseout:	Use this to invoke JavaScript if the mouse goes pass some link
onunload:	Use this to invoke JavaScript right after someone leaves this page.

Onclick event:

```
<html>
<body>
<script>
function inform(){
   alert("You have activated me by clicking the grey button! Note that the
event handler is added within the event that it handles, in this case, the form
button event tag")
}
</script>
<form>
<input type="button" name="test" value="Click me" onclick="inform()">
</form>
</body>
</html>
```

Onload event: The onload event handler is used to call the execution of JavaScript after a page, frame or image has completely loaded. It is added like this:

```
 1.<body onload="inform()"> //Execution of code //will begin after the page
has loaded.

2.<frameset onload="inform()"> //Execution of code //will begin after the
current frame has loaded.

3.<img src="whatever.gif" onload="inform()"> //Execution of code will begin
after the image has loaded.
<frameset cols="25%,*,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
e.g. for onload event
<html>
<head><title>Body onload example</title>
</head>
<body onload="alert('This page has finished loading!')">
```

```
Welcome to my page
</body>
</html>
```

onUnload event handler

onunload executes JavaScript *immediately after* someone leaves the page. A common use (though not that great) is to thank someone as that person leaves your page for coming and visiting.

```
<body onunload="alert('Thank you. Please come back to this site and visit us soon, ok?')">
```

Difference between mouseover,mousemove and mouseenter event

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
   width: 100px;
   height: 100px;
   border: 1px solid black;
   margin: 10px;
   float: left;
   padding: 30px;
   text-align: center;
   background-color: lightgray;
}
p {
   background-color: white;
}
</style>
</head>
<body>
<div onmousemove="myMoveFunction()">
  <p>onmousemove: <br> <span id="demo">Mouse over me!</span></p>
</div>
<div onmouseenter="myEnterFunction()">
  <p>onmouseenter: <br> <span id="demo2">Mouse over me!</span></p>
</div>
<div onmouseover="myOverFunction()">
  <p>onmouseover: <br> <span id="demo3">Mouse over me!</span></p>
</div>
<div onmouseout="myOutFunction()">
  <p>onmouseout: <br> <span id="demo4">Mouse out me!</span></p>
</div>

<script>
```

```
var x = 0;
var y = 0;
var z = 0;
function myMoveFunction() {
  document.getElementById("demo").innerHTML = z+=1;
}
function myEnterFunction() {
  document.getElementById("demo2").innerHTML = x+=1;
}
function myOverFunction() {
  document.getElementById("demo3").innerHTML = y+=1;
}
function myOutFunction() {
  document.getElementById("demo4").innerHTML = y+=1;
}
</script>
</body>
</html>
```

Different Kinds of Loops

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
- **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

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

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

<!DOCTYPE html>

<html><body>

** Click the button to loop through a block of code five times. **

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

** **

<script>

function myFunction() {

var text = "";

var i;

```
  for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body></html>
```

Statement 1

Normally you will use statement 1 to initiate the variable used in the loop (i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i, len, text;
for (i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

And you can omit statement 1 (like when your values are set before the loop starts):

```
<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Statement 2 is also optional. Statement 3 is also optional.

Introduction to Arrays in JavaScript

An array a regular data structure comprises individual elements that can be referenced to one or more integer index variables, the number of such indices being the number of

dimensions or number of elements in the array.In JavaScript, an array is usually represented using the format **arrayName[i],** where i is the index or subscript which represents the position of the element within the array**.** The array always starts with the zeroth element. For example, an array A of 8 elements comprises elements A[0], A[1], A[2], A[3], A[4],A[5], A[6], A[7].

Let's illustrate the above example further. The values of each element shown in the table below:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
54	60	23	45	67	90	85	34

From the above table, the value of A[0]=54, A[1]=60, A[2]=23 and so forth.

We can perform arithmetic operations on the array elements. For example,

x=A[1]+A[2]+A[3]=137

The above operations add up the values of three array elements and store the sum in the variable x.

22.2 Declaring and Allocating Arrays

In JavaScript, an array can be declared using a number of ways. If the values of the array are unknown or not initialized at the beginning, we use the following statement:

var A= new Array(12);

which create an array (or rather array object) that consists of 12 elements. The operator **new** allocates memory to store 12 elements of the array. The process of creating a new object is known as creating an instance of the object, in this case an Array object.

Different arrays may be created using one single declaration, separated by commas,as follows:

var x=new Array(20), y=new Array(15), z= new Array(30)

Example

```
<head>
```

```
<script>
function Start()
{ var A= new Array(10);
for (var i=0; i<A.length;++i)
A[i]=i*10;
for (var i=0; i<A.length;++i)
document.writeln ((“element “+i+” is “+A[i]+”<br>”);
}
</script>
</head>
<body>
</body>
```

&nsbp;

The output is shown below:

element 0 is 0
element 1 is 10
element 2 is 20
element 3 is 30
element 4 is 40
element 5 is 50
element 6 is 60
element 7 is 70
element 8 is 80
element 9 is 90

Arrays

Creating an Array

Two ways:

(i)An array literal is the easiest way to create a JavaScript Array

var *array_name* = [*item1*, *item2*, ...];

e.g. var cars = ["Saab", "Volvo", "BMW"];

(ii) Using the JavaScript Keyword new

e.g. var cars = new Array("Saab", "Volvo", "BMW");

Access the Elements of an Array

access an array element by referring to the **index number**.

e.g. var name = cars[0];

```
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML =
    cars;
</script>
```

Changing an Array Element

```
<script>
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML =
   cars;
</script>
```

Arrays are Objects

- Arrays are a special type of objects.
- The typeof operator in JavaScript returns "object" for arrays.
- Arrays use **numbers** to access its "elements“

E.g. person[0]

- Objects use **names** to access its "members“

E.g. person.firstName

Array Elements Can Be Objects

- We can have objects in an Array.
- We can have functions in an Array.
- we can have arrays in an Array:

```
<script>
function myf()
{
return ("hello");
}
fruit=["apple","grapes"];
var cars = [Date(), myf(), fruit];
document.getElementById("demo").innerHTML = cars;
</script>
```

Array Properties and Methods

- The length Property

```
<script>
var fruits = ["Banana", "Orange", "Apple",
   "Mango"];
document.getElementById("demo").innerHTML
   = fruits.length;
</script>
```

Looping Array Elements

```
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>
```

Array Methods

(i) Adding Array Elements

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>
```

- New element can also be added to an array using the length property:

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

fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to fruits

fruits[6] = "Lemon";    // create undefined holes in an array
```

Associative Arrays

- Many programming languages support arrays with named indexes.
- 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**.

- E.g. Cars[0];✓
- You should use **objects** when you want the element names to be **strings (text)**.

Avoid new Array()

- var points = new Array(); // Bad
- var points = []; // Good

- var points = new Array(40, 100, 1, 5, 25, 10); // Bad
- var points = [40, 100, 1, 5, 25, 10]; // Good

new keyword only complicates the code

It can also produce some unexpected results:

- var points = new Array(40, 100); // Creates an array with two elements (40 and 100)

What if I remove one of the elements?

- var points = new Array(40); // Creates an array with 40 undefined elements !!!!!

How to Recognize an Array

- The problem is that the JavaScript operator typeof returns "object":

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

   typeof fruits;    // returns object
```

Answer is:

```
<script>
var fruits = ["Banana", "Orange", "Apple",
   "Mango"];
document.getElementById("demo").innerHTML
   = Array.isArray(fruits);  //return true
</script>
```

Array Methods
(i) toString()

toString() converts an array to a string of (comma separated)

```
<script>
var fruits = ["Banana", "Orange", "Apple",
"Mango"];
document.getElementById("demo").innerHTML =
fruits.toString();
</script>
```

(ii) join()

- The join() method also joins all array elements into a string.

```
<script>
var fruits = ["Banana", "Orange", "Apple",
    "Mango"];
document.getElementById("demo").innerHTML =
    fruits.join(" * ");
</script>
```

Popping and Pushing

When we work with arrays, it is easy to remove elements and add new elements.

- Popping items **out** of an array,
- pushing items **into** an array

Popping

- The pop() method removes the last element from an array:

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML =
   fruits;
document.getElementById("demo2").innerHTML =
   fruits.pop();   //print removed element
document.getElementById("demo3").innerHTML =
   fruits;
</script>
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

- this 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
```

- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:
- var fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.unshift("Lemon");
-

Deleting Elements

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

```
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 more than one items and delete item in an array:

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

- 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**.

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

- last two parameter are optional

Merging (Concatenating) Arrays

```
<script>
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML =
    myChildren;

//var myChildren = arr1.concat(arr2, arr3);
```

The concat() method can also take values as arguments:

```
<script>
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias",
   "Linus"]);
document.getElementById("demo").innerHTML
   = myChildren;
</script>
```

Slicing an Array

- The slice() method slices out a piece of an array into a new array.
- This example slices out array element from array according to given number:

```
var fruits =
    ["Banana", "Orange", "Lemon", "Apple", "Man
    go"];
    var citrus = fruits.slice(1,3);//exclude the third
    position element
```

Orange, Lemon

Sorting an Array

The sort() method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple",
   "Mango"];
function myFunction()
 {
  fruits.sort();
  document.getElementById("demo").innerHTML
   = fruits;
}
```

Reversing an Array

The reverse() method reverses the elements in an array.

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


fruits.sort();
fruits.reverse();
```

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.

- This can be solved by **compare function**:

```
<button onclick="myFunction()">Try it</button>
<script>
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points;

function myFunction() {
  points.sort(function(a, b){return a - b});
  document.getElementById("demo").innerHTML = points;
}
</script>
```

The Compare Function

The compare function should return a negative, zero, or positive value, depending on the arguments.

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

- If the result is negative a is sorted before b.
- If the result is positive b is sorted before a.
- If the result is 0 no changes are done with the sort order of the two values.

```
<button onclick="myFunction1()">Sort Alphabetically</button>
<button onclick="myFunction2()">Sort Numerically</button>

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

<script>
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points;

function myFunction1() {
  points.sort();
  document.getElementById("demo").innerHTML = points;
}

function myFunction2() {
  points.sort(function(a, b){return a - b});
  document.getElementById("demo").innerHTML = points;
}
</script>
```

Anonymous function

- A function without a name is called anonymous function.

```
<script>
var x = function (a, b) {return a * b};
document.getElementById("demo").innerHTML = x(4, 3);
</script>
```

The Function() Constructor

1. JavaScript functions are defined with the function keyword.
2. Functions can also be defined with a built-in JavaScript function constructor called Function().

```
<script>
var myFunction = new Function("a", "b", "return a * b");
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
```

Self-Invoking Functions

- Function expressions can be made "self-invoking".
- A self-invoking expression is invoked (started) automatically, without being called.
- Function expressions will execute automatically if the expression is followed by ().

```
(function () {
    var x = "Hello!!";  // I will invoke myself
})();
```

- This function is a **anonymous self-invoking function** (function without name).

JavaScript Closures

- JavaScript variables can belong to the **local** or **global** scope.
- Global variables can be made local (private) with **closures**.

Global Variables

(local variable)

```
function myFunction() {
    var a = 4;
    return a * a;
}
```

- **a** is a **local** variable.
- A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.
- Global and local variables with the same name are different variables. Modifying one, does not modify the other.

(global)

```
var a = 4;
function myFunction() {
  return a * a;
}
```

- **a** is a **global** variable.
- In a web page, global variables belong to the window object.
- Global variables can be used (and changed) by all scripts in the page (and in the window).

Variable Lifetime

- Global variables live as long as your application (your window / your web page) lives.
- Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

A Counter Dilemma

```
<script>
// Initiate counter
var counter = 0;

function add() {
  counter += 1;
}

// Call add() 3 times
add();
add();
add();

document.getElementById("demo").innerHTML = "The counter is: " + counter;
</script>
```

Result: 3

- There is a problem with the solution above: Any code on the page can change the counter, without calling add().

The counter should be local to the add() function, to prevent other code from changing it:

If we remove global counter

```
var counter=0;
function add() {
  var counter = 0;
  counter += 1;
}

// Call add() 3 times
add();
add();
add();
```

Result: 0 (global variable value)

```
function add() {
    var counter = 0;
    counter += 1;
  }

  // Call add() 3 times
  add();
  add();
  add();
```

It did not work because we reset the local counter every time we call the function.
A JavaScript inner function can solve this.

Result: 1

JavaScript Nested Functions

```
<script>
document.getElementById("demo").innerHTML =
    add();
function add() {
  var counter = 0;
  function plus() {counter += 1;}
  plus();
  return counter;
}
</script>
```

This could have solved the counter dilemma, if we could reach the plus() function from the outside.

But, We also need to find a way to execute counter = 0 only once.

JavaScript Closures

- var add = (function () {
  var counter = 0;
  return function () {counter
  += 1; return counter}
  })();

  add();
  add();
  add();

self-invoking functions

Result: 3


THANKS

Passing array to function

- we can pass the entire array as a parameter to a function in JavaScript. This method of array passing is called call by reference in JavaScript.

  - To pass an array argument to a function, simply pass the name of an array (a reference to an array) without brackets.

```
<script>
  let nums = new Array(20, 10, 25, 15, 35, 40);
  let arrayLength = nums.length;
  document.write("Original array elements are: ", "<br/>");

  for(i = 0; i < arrayLength; i++) {
      document.write(nums[i]+ " ");
  }
```

```
document.write("<hr>");
// Function to pass an array by reference.
 function modifyArray(x) {
 document.write("Modified array elements: ", "<br/>");
 for(i = 0; i < arrayLength; i++) {
    document.write(nums[i] * 5 + " ");
  }
}
// Calling function by passing array.
  modifyArray(nums); // entire array passed by reference.
</script>
```

- In the above example program, we have passed an array nums into modifyArray() as pass by reference. That is, the parameter nums is passed into modifyArray() function with reference to x. Inside the function the array elements are multiplied by 5 and displayed.

Recursion and Iteration

A program is called recursive when an entity calls itself.

A program is call iterative when there is a loop (or repetition).

```
<html>
<head>

<script>
let number=prompt("enter the number");
function factorial(number){
  var result = 1;

  for(var i = 1; i <= number; i++){
    result = result * i;
  }
  return result;

}
```

```
function factorial1(number) {
    if (number === 1) {
        return 1;
    }
    return number * factorial1(number - 1);
}
document.write(factorial(number));
document.write(factorial1(number));
</script>
</head>
</html>
```

Boolean Object

Boolean is an object that represents value in two states: *true* or *false*.

JavaScript Boolean object can be created by Boolean() constructor

var b=new Boolean(value);

The default value of JavaScript Boolean object is *false*.

e.g. var b=new Boolean(1);

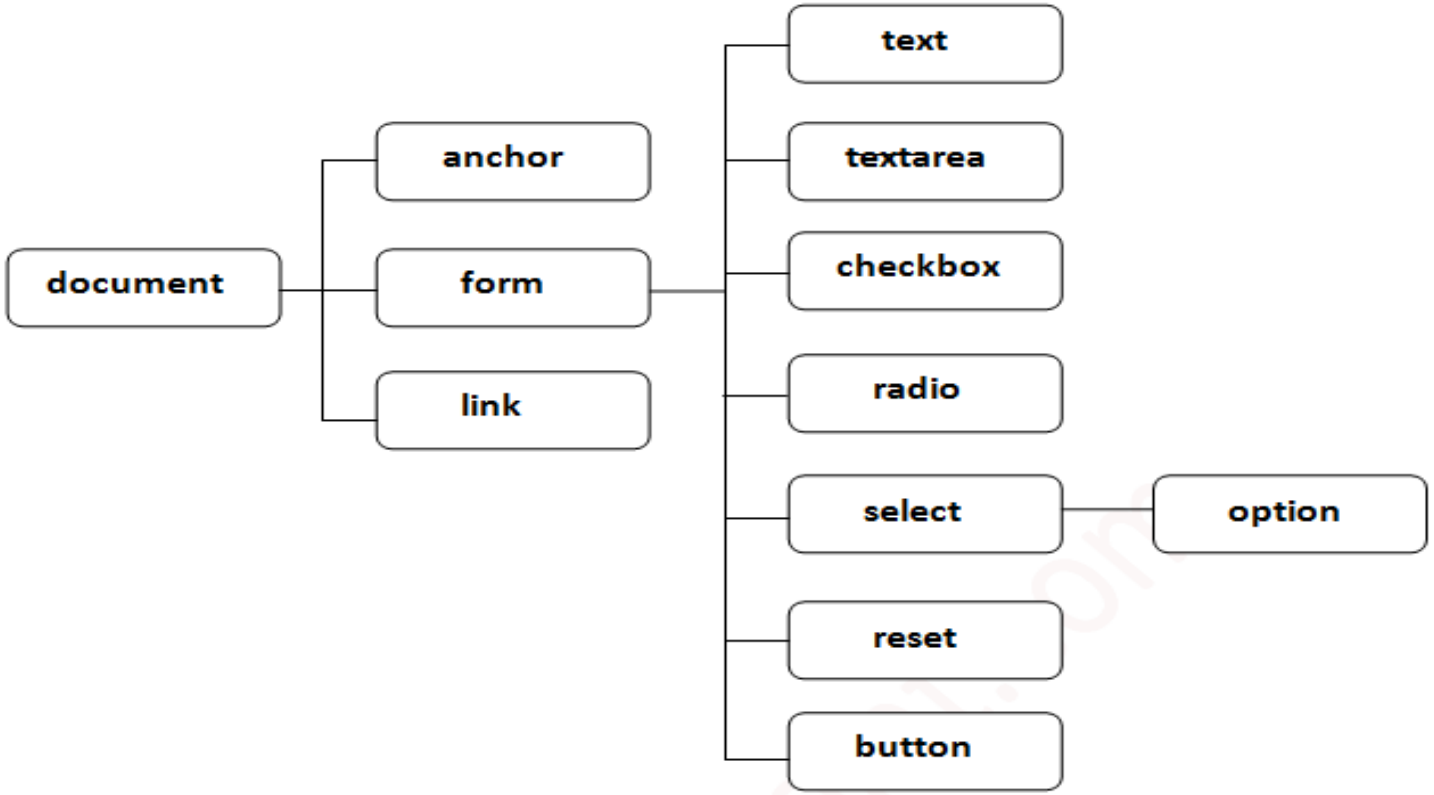
- var b1=new Boolean(1);
- var b2=new Boolean(0);
- var b3=new Boolean(NaN);
- var b4=new Boolean(“”);
- var b5=new Boolean(“hello”);
- var b6=new Boolean(null);

Document Object

- The **document object** represents the whole html document.
- When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document.

- It has properties and methods.

Properties of document object



```
<html>
    <head>
        <title>
            this is document objject
        </title>
        </head>
         <body>
            <a name="1" href="">hello</a>
            <a href="">hello</a>
            <a href="">hello</a>
            <a href="">hello</a>

<script>

document.write(document.title);
document.write(document.anchors.length);
</script>
```

```
<html>
<body>

<h1>The Document Object</h1>
<h2>The links Property</h2>

<p><a href="/html/default.asp">HTML l</a></p>
<p><a href="/css/default.asp">CSS </a></p>

<p>The URL of the first link in the document is:</p>
<p id="demo"></p>

<script>
let url = document.links[0];
document.getElementById("demo").innerHTML = url;
</script>

</body>
</html>
```

Methods

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

JavaScript Events

Event

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

- When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

onclick Event

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

```
 <html>
<head>

<script type="text/javascript">

function sayHello() {
alert("Hello World")
}
</script>

</head>
<body>
<p>Click the following button and see result</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello" />
</form>

</body>
</html>
```

onmouseover

- <html>
- <head>
- **<script type="text/javascript">**
- **function over() {**
- **document.write ("Mouse Over");**
- **}**
- **</script>**

- </head>
- <body>
- Bring your mouse inside the division to see the result:
- <div **onmouseover="over()**">
- <h2> This is inside the division </h2>
- </body>

onsubmit Event

- **onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

- The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

Form validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

- 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.

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">

function validate()
{
var emailID = document.myForm.EMail.value;
atpos = emailID.indexOf("@");//e.g. @gmail.com  or bc.12@gmail.c
dotpos = emailID.lastIndexOf(".");     //abc@gmail.com
if (atpos < 1 || ( dotpos - atpos < 2 ))
{
alert("Please enter correct email ID")
document.myForm.EMail.focus() ;
return false;
}
```

```
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;
}
```

```
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi"
   name="myForm" onsubmit="return(validate (
   ));">
<table cellspacing="2" cellpadding="2"
   border="1">
```

```
<tr>
<td align="right">Name</td>
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
<td align="right">Country</td>
```

```
<td>
<select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">USA</option>
<option value="2">UK</option>
<option value="3">INDIA</option>
</select>
</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
</body>
</html>
```

Javascript Debugging

- It is difficult to write JavaScript code without a debugger.
- Your code might contain syntax errors, or logical errors, that are difficult to diagnose.
- Often, when JavaScript code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.

Debugger

- Searching for errors in programming code is called code debugging.

- Debugging is not easy. But fortunately, all modern browsers have a built-in debugger. Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

- With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.

- Activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

The console.log() Method

- If your browser supports debugging, you can use console.log() to display JavaScript values in the debugger window:

- ```
  <!DOCTYPE html>
  <html>
  <body>
  <h1>My First Web Page</h1>
  <script>
  a = 5;
  b = 6;
  c = a + b;
  console.log(c);
  </script>

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

Setting Breakpoints

- In the debugger window, you can set breakpoints in the JavaScript code.At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.After examining values, you can resume the execution of code (typically with a play button).

The debugger Keyword

- The **debugger** keyword stops the execution of JavaScript, and calls (if available) the debugging function.
- This has the same function as setting a breakpoint in the debugger.
- If no debugging is available, the debugger statement has no effect.
- With the debugger turned on, this code will stop executing before it executes the third line.

```
var x = 15 * 5;   //var x==15*5;
debugger;
document.getElementbyId("demo").innerHTML = x;
```

objects

javaScript

different ways to create new objects:

1. Define and create a single object, using an object literal.
2. Define and create a single object, with the keyword new.
3. Define an object constructor, and then create objects of the constructed type.

Object Literal

Using an object literal, you both define and create an object in one statement.

```
var person = {firstName:"John", lastName:"Doe",
    age:50, eyeColor:"blue"};
```

Using the JavaScript Keyword new

```
<script>
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

Using an Object Constructor

- Sometimes we like to have an "object type" that can be used to create many objects of one type.

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}


var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age;
</script>
```

JavaScript Objects are Mutable

- If y is an object, the following statement will not create a copy of y:

var x = y; // This will not create a copy of y.

Any changes to y will also change x, because x and y are the same object.

The object x is not a **copy** of y. It **is** y. Both x and y points to the same object.

```
<script>
var person = {firstName:"John", lastName:"Doe",
   age:50, eyeColor:"blue"}

var x = person;
x.age = 10;

document.getElementById("demo").innerHTML =
x.firstName + " is " + person.age + " years old.";
</script>
```

Object Properties

```
<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
    txt += person[x] + " ";
}
document.getElementById("demo").innerHTML =
    txt;
</script>
```

Adding New Properties

```
<script>
var person = {
    firstname:"John",
    lastname:"Doe",
    age:50,
    eyecolor:"blue"
};
person.nationality = "English";
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.nationality + ".";
</script>
```

Deleting Properties

```
<script>
var person = {
    firstname:"John",
    lastname:"Doe",
    age:50,
    eyecolor:"blue"
};
delete person.age;
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years old.";
</script>
```

JavaScript Methods

- A JavaScript **method** is a property containing a **function definition**.

```
<script>
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};
document.getElementById("demo").innerHTML =
    person.fullName();
</script>
```

Defining methods to an object is done inside the constructor function

```
<script>
function person(firstName,lastName,age,eyeColor) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.eyeColor = eyeColor;
    this.changeName = function (name) {
        this.lastName = name;
    }
}
var myMother = new person("Sally","Rally",48,"green");
myMother.changeName("Doe");
document.getElementById("demo").innerHTML =
"My mother's last name is " + myMother.lastName;
</script>
```

Object Prototypes

Every JavaScript object has a prototype.The **prototype** is also an object. All **JavaScript** objects inherit their properties and methods from their **prototype.**

Creating a Prototype

- With a constructor function, you can use the **new** keyword to create new objects from the same prototype:

```
<script>
function person(first, last, age, eye) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age;
</script>
```

Adding a Property to an Object

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
myFather.nationality = "English";
document.getElementById("demo").innerHTML =
"My father is " + myFather.nationality;
</script>
```

Adding a Method to an Object

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
myFather.name = function() {
    return this.firstName + " " + this.lastName;
}
document.getElementById("demo").innerHTML =
"My father is " + myFather.name();
</script>
```

Adding Properties to a Prototype

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
person.nationality = "English";
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
document.getElementById("demo").innerHTML =
"My father is " + myFather.nationality;
</script>
```

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
    this.nationality = "English";      ✔
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
document.getElementById("demo").innerHTML =
"My father is " + myFather.nationality + ". My mother is "
    + myMother.nationality;
</script>
```

Adding Methods to a Prototype

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
    this.name = function() {
        return this.firstName + " " + this.lastName
    };
}

var myFather = new person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"My father is " + myFather.name();
</script>
```

Using the prototype Property

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
person.prototype.nationality = "English";
var myFather = new person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"My father is " + myFather.nationality;
</script>
```

```
<script>
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
person.prototype.name = function() {
    return this.firstName + " " + this.lastName
};
var myFather = new person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"My father is " + myFather.name();
</script>
```

Using Built-In Methods

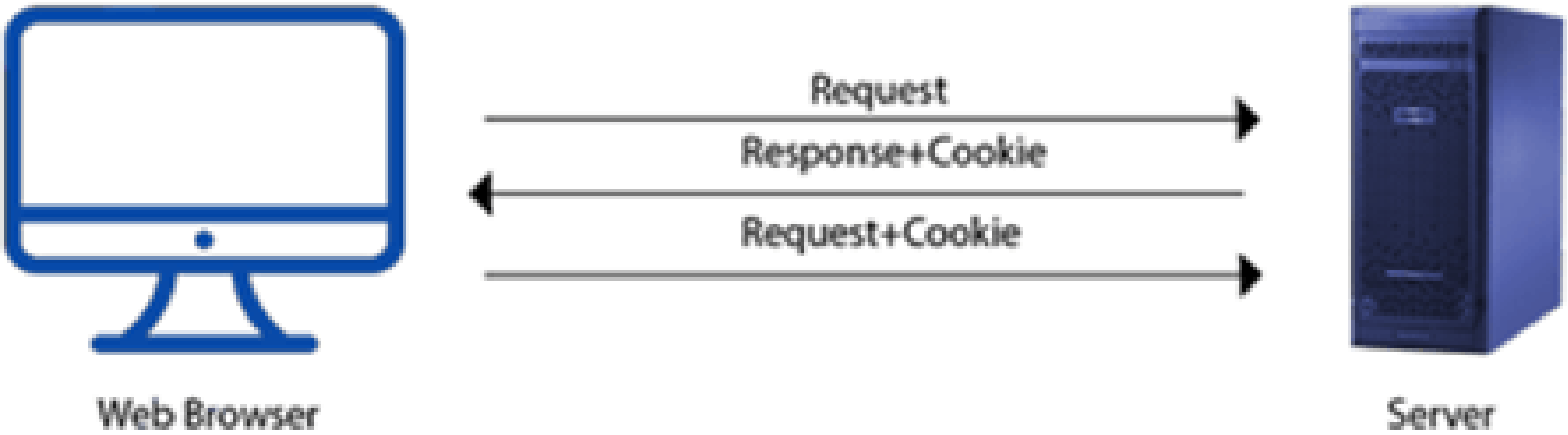
```
var message = "Hello world!";
    var x = message.toUpperCase();
```

JavaScript Cookie

- A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.
    - A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

How Cookies Works?

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
- So, to recognize the old user, we need to add the cookie with the response from the server.
- browser at the client-side.
- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.


Request
Response+Cookie
Request+Cookie
Web Browser
Server

create a Cookie in JavaScript

- In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

E.g.

```
document.cookie="name=value";
```

- <!DOCTYPE html>
- **<html>**
- **<head>**
- **</head>**
- **<body>**
- **<input** type="button" value="setCookie" onclick="setCookie()">
- **<input** type="button" value="getCookie" onclick="getCookie()">
-
-

- **<script>**
- function setCookie()
- {
- document.cookie="username=Duke Martin";
- }
- function getCookie()
- {
- if(document.cookie.length!=0)
- {
- alert(document.cookie);
- }
-

```
else
        {
        alert("Cookie not available");
        }
    }
    </script>

</body>
</html>
```

Cookie Attributes

- JavaScript provides some optional attributes that enhance the functionality of cookies.

expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

- function setCookie()
- {
- document.cookie="username=Monika;expires=Sun, 20 Aug 2030 12:00:00 UTC; max-age=“1000; path=/; domain=javatpoint.com ";
- }