Java Script

Intro. to JavaScript “JS “

- ✓ **JS** is a client-side scripting language that runs entirely inside the web browser.
- ✓ **JS** brings a dynamic functionality to your websites and offers effects that are not otherwise possible.
- ✓ **JS** runs inside the browser and has direct access to all the elements in a web document.
- ✓ **JS** first appeared in the Netscape Navigator browser in 1995.
- ✓ **JS** is different from JAVA
- ✓ **JS** brings a webpage to life by responding to users requests.
- ✓ **JS** sits with HTML & CSS

JS In HTML

- ✓ The primary method of inserting JavaScript into an HTML page is via the **< script >** element.
- ✓ The < script > element:
- ➢ **charset** — Optional. The character set of the code specified using the src attribute. This attribute is rarely used, because most browsers don ’ t honor its value.
- ➢ **defer** — Optional. Indicates that the execution of the script can safely be delayed until after the document ’ s content has been completely parsed and displayed.
- ➢ **Async** — Optional. The script is executed asynchronously with the rest of the page (the script will be executed while the page continues the parsing)
- ➢ **language** — Deprecated. Originally indicated the scripting language being used by the code block (such as “ JavaScript ” , “ JavaScript1.2 ” , or “ VBScript “). Most browsers ignore this attribute; it should not be used.
- ➢ **src** — Optional. Indicates an external file that contains code to be executed.
- ➢ **type** — Required. Seen as a replacement for language ;

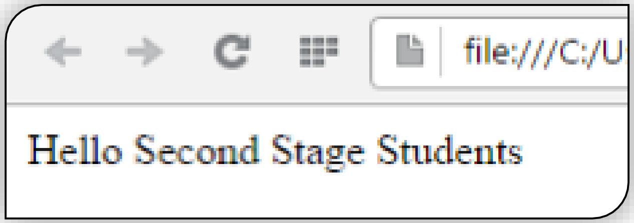
- ✓ There are two ways to use the < script > element: embed JavaScript code directly into the page or include JavaScript from an external file.

JS Inside HTML

- ✓ To call it up, you place your JavaScript code between opening **<script>** and closing **</script>** HTML tags.

- ✓ It could be placed inside the head part or inside the body part of the **HTML document**.
- ✓ Ex:

```
<html>
<head><title>Hello World</title></head>
<body>
        <script type="text/javascript">
        document.write("Hello Second Stage Students")
        </script>
 </body>
</html>
```



Hello Second Stage Students

JS OUTPUT

- Data in JS can be displayed in these different ways:

➢ In HTML element, using innerHTML.

➢ In HTML output using document.write().

➢ In an alert box, using window.alert().

➢ In the browser console, using console.log().

➢ **<noscript>** and **</noscript>** pair of tags. These are used when you wish to offer alternative HTML to users whose browsers do not support JavaScript or who have it disabled.

innerHTML

```
<html>
<body>
<h1>Output in JS using innerHTML </h1>
<p>Watch the paragraph below</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "H"+"E"+"L"+"L"+"O"
</script>
</body>
</html>
```

- **document.getElementById(id)** is a method that's used to access an HTML element
- The **id** attribute defines the HTML element.
- The **innerHTML** property defines the HTML content

window.alert()

- Alert() or window.alert() is a method used to display data.

```
<html>
<body>
<h1>Output in JS using innerHTML </h1>
<p>Watch the paragraph below</p>
<p id="demo"></p>
<script>
window.alert("H"+"E"+"L"+"L"+"O");
</script>
</body>
</html>
```

Console.log()

- Consol.log() is a method used to display data on consol screen. Its almost used for debugging purposes.

```
<html>
<body>
<script>
console.log(2+3);
</script>
</body>
</html>
```

Document writing

- ✓ **document.write** simply outputs the supplied string to the current document, where it is displayed.
- ✓ **document.write** could be used to write html elements, like “header or a paragraph”.
- ✓ Syntax

document.write(exp1, exp2, exp3, ...)

- ✓ exp1, exp2, exp3, ... Optional. What to write to the output stream. Multiple arguments can be listed and they will be appended to the document in order of occurrence.

//Write some text directly to the HTML document:

document.write("Hello World!");

// This will write a header:

document.write("<h1>Header or title</h1>");

// This will write a paragraph:

document.write("<p>Random Paragraph</p>");

//Write HTML elements with text directly to the HTML document:

document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");

Document writing

- ✓ The **document.writeln()** method is similar to write(), only it adds a newline character after each statement.

```
<body>
<p>Note that write() does NOT add a new line after each statement:</p>
<pre>
<script>
document.write("Hello World!");
document.write("Have a nice day!");
</script>
</pre>
<p>Note that writeln() add a new line after each statement:</p>
<pre>
<script>
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</pre></body>
```

Document writing

//Write some text directly to the HTML document, with a new line after each statement (using <br>):

document.write("Hello World! <br>");

document.write("Have a nice day!");

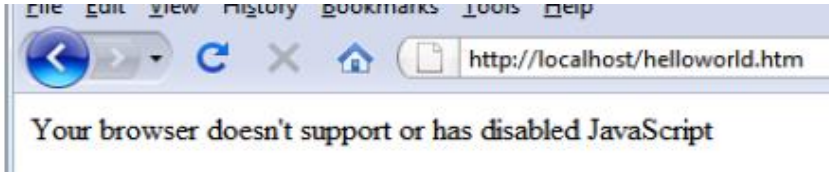
//Write the Date object directly to the HTML document:

document.write(Date());

//Write the bold date as string

```
< p > The current date and time is:
        < script type=”text/javascript” >
        document.write(“ < strong > ” + (new Date()).toString() + “ < /strong > ”);
        < /script >
< /p >
```

- ✓ **<noscript>** and **</noscript>** pair of tags. These are used when you wish to offer alternative HTML to users whose browsers do not support JavaScript or who have it disabled.


File Edit View History Bookmarks Tools Help
http://localhost/helloworld.htm

Your browser doesn't support or has disabled JavaScript

Including JavaScript Files(External scripting)

- ✓ In addition to writing JavaScript code directly in HTML documents, you can include files of JavaScript code. The syntax for this is:

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

- ✓ There is no need for the <script>and </script> tags in the external js file.
- ✓ // symbol is used to denote that everything follows on that line must be ignored.

Ex. // This is a comment

- ✓ For multiline comments /* Comment */ is used

Ex: /* This is a section of multiline
comments that will not be
interpreted */

Statements

- Statements are terminated by a semicolon, though omitting the semicolon makes the parser determine where the end of a statement occurs, as in the following examples:

var sum = a + b   //valid even without a semicolon - **not recommended**

var diff = a - b;   //valid - **preferred**

- Including semicolons helps prevent errors of omission, such as not finishing what you were typing, and allows developers to compress the Script code by removing extra white space (such compression causes syntax errors when lines do not end in a semicolon).
- Multiple statements can be combined into a code block by using C - style syntax, beginning with a left curly brace ({) and ending with a right curly brace (}):

```
if (test){
test = false;
alert(test);}
```

- Control statements, such as if , require code blocks only when executing multiple statements. However, it is considered a best practice to always use code blocks with control statements, even if there ' s only one statement to be executed, as in the following examples:

if (test)

alert(test);   **//valid, but error-prone and should be avoided**

if (test){   **//preferred**

alert(test); }

Data Type and typeof opertator

✓Because JS is loosely typed, there needs to be a way to determine the data type of a given variable.

✓The **typeof** operator provides that information.

✓Using the **typeof** operator on a value returns one of the following strings:

- **“undefined”** →if the value is undefined
- **“boolean”**→ if the value is a Boolean
- **“string”** → if the value is a string
- **“number**” → if the value is a number
- **“object”** → if the value is an object or null
- **“function”** → if the value is a function

✓ **The typeof operator is called like this:**

```
var message = “some string”;
alert(typeof message); //”string”
alert(typeof(message)); //”string”
alert(typeof 95); //”number”
```

Examples

```
var s = “Nicholas”;
var b = true;
var i = 22;
var u;
var n = null;
var o = new Object();
alert(typeof s); //string
alert(typeof i); //number
alert(typeof b); //boolean
alert(typeof u); //undefined
alert(typeof n); //object
alert(typeof o); //object
```

Note that because typeof is an operator and not a function, no parentheses are required (although they can be used).

isNaN() Function

- ✓ **NaN** is a special numeric value, short for **N**ot **a** **N**umber , which is used to indicate when an operation intended to return a number has failed.
- ✓ For example, dividing any number by 0 typically causes an error in other programming languages, halting code execution. In JS, dividing a number by 0 returns NaN , which allows other processing to continue.
- ✓ The value NaN has a couple of unique properties. **First,** any operation involving NaN always returns NaN (for instance, NaN /10), which can be problematic in the case of multistep computations. **Second,** NaN is not equal to any value, including NaN . For example, the following returns false :

  alert(NaN == NaN); → //false
- ✓ For this reason, JS provides the isNaN() function. This function accepts a single argument, which can be of any data type, to determine if the value is " not a number. "
- ✓ When a value is passed into isNaN() , an attempt is made to convert it into a number. Some non - number values convert into numbers directly, such as the string "10" or a Boolean value. Any value that cannot be converted into a number causes the function to return true .

- ➢ alert(isNaN(NaN)); → //true
- ➢ alert(isNaN(10)); → //false - 10 is a number
- ➢ alert(isNaN("10")); → //false - can be converted to number 10
- ➢ alert(isNaN("blue")); → //true - cannot be converted to a number
- ➢ alert(isNaN(true)); → //false - can be converted to number 1

Variables

- A variable may include only the letters a-z, A-Z, 0-9, the $ symbol, and the underscore .
- No other characters, such as spaces or punctuation, are allowed in a variable name.
- The first character of a variable name can be only a-z, A-Z, $, or _ (no numbers).
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.
- **String Variables:** JS string variables should be enclosed in either single or double quotation marks, like this:

  greeting = "Hello there"

  warning = 'Be careful‘
- You may include a single quote ’ within a double-quoted “ string or a double quote within a single-quoted string, but a quote of the same type must be escaped using the backslash character, like this:

  greeting = "\"Hello there\" is a greeting"

  warning = '\'Be careful\' is a warning'

To print the value of a string variable→ **document.write(greeting)**

- **Numeric Variables:** Creating a numeric variable is as simple as assigning a value, like these examples:

  count = 42; **temperature = 98.4;**

To print the value of a numeric variable→ **document.write(count)**

Arrays

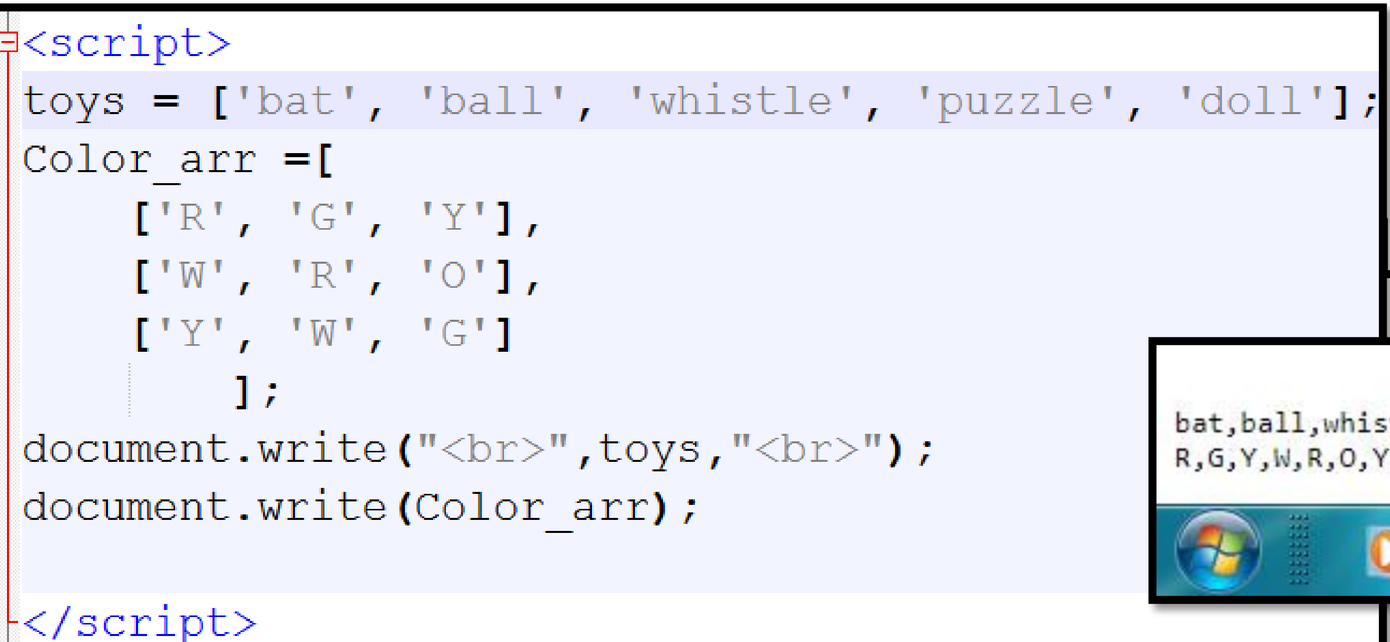
- In JS an array can contain string or numeric data, as well as other arrays.
- To assign values to an array, use the following syntax (which in this case creates an array of strings):

Ex: **toys = ['bat', 'ball', 'whistle', 'puzzle', 'doll']**

- To create a multidimensional array, nest smaller arrays within a larger one.

EX: Color_arr =[['R', 'G', 'Y'],['W', 'R', 'O'],['Y', 'W', 'G']]

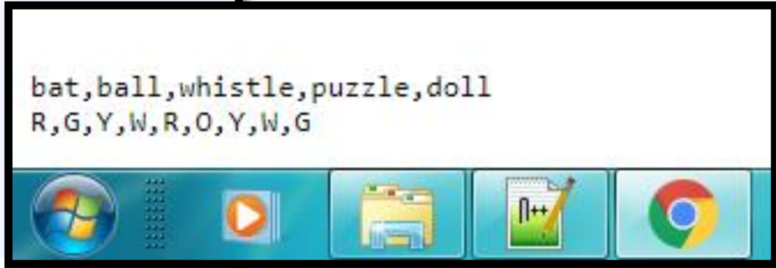
To print the value of a Color_arr → **document.write(**Color_arr**)**



```
<script>
toys = ['bat', 'ball', 'whistle', 'puzzle', 'doll'];
Color_arr =[
    ['R', 'G', 'Y'],
    ['W', 'R', 'O'],
    ['Y', 'W', 'G']
        ];
document.write("<br>",toys,"<br>");
document.write(Color_arr);

</script>
```

o/p



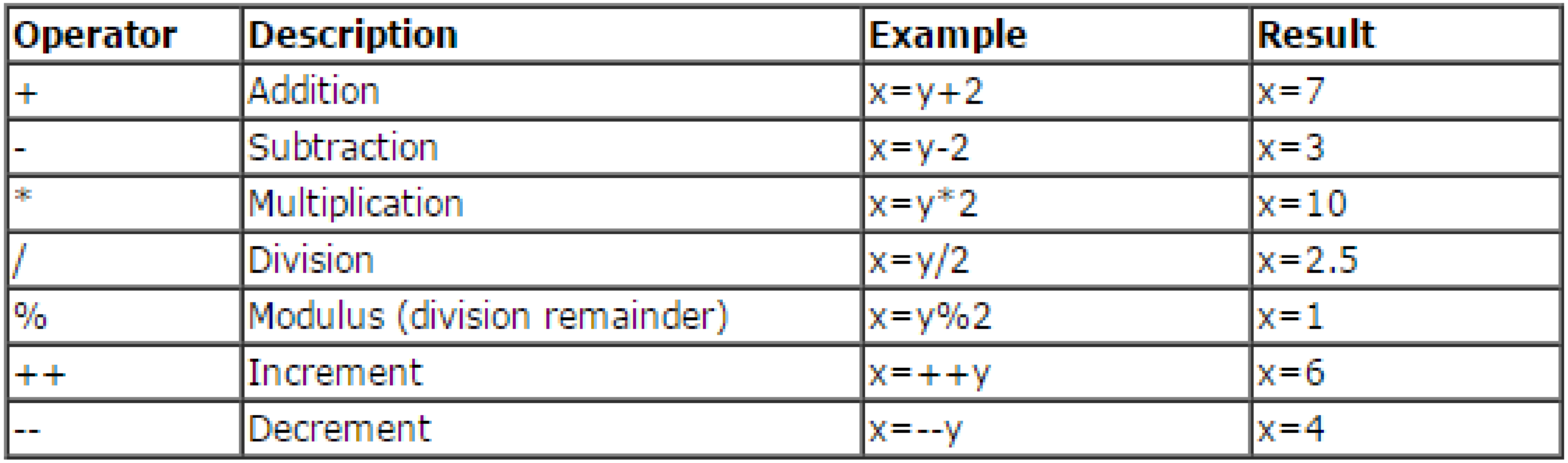
```
bat,ball,whistle,puzzle,doll
R,G,Y,W,R,O,Y,W,G
```

JavaScript Operators

➢ JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

✓ Given that y=5, the table below explains the arithmetic operators:



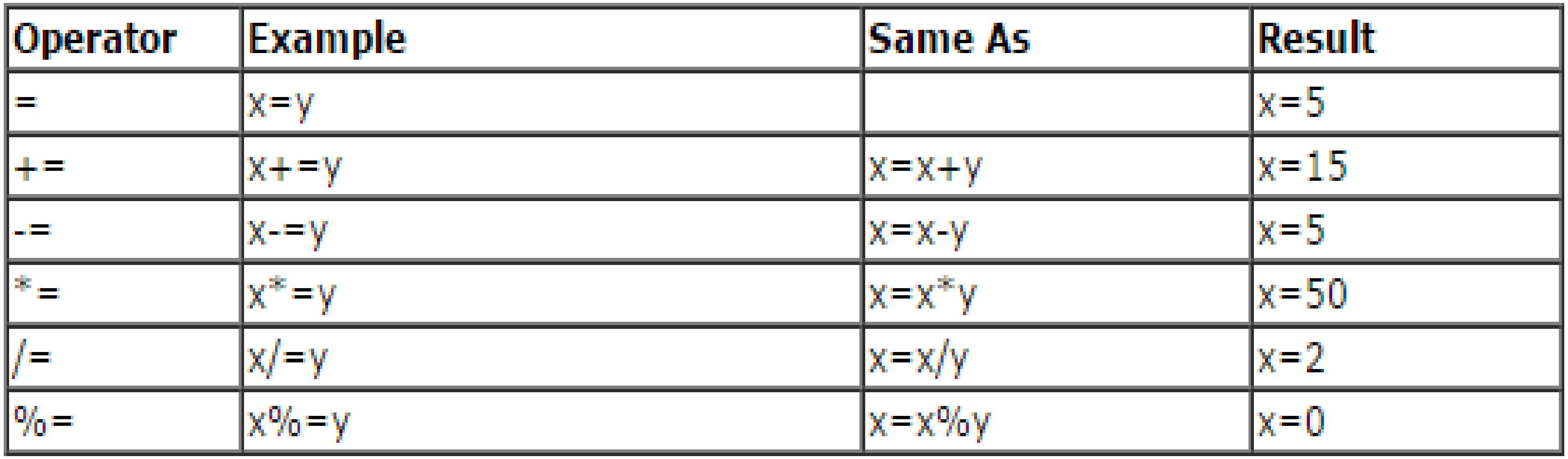
Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

JavaScript Operators

- ➢ JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

- ✓ Given that x=10 and y=5, the table below explains the assignment operators:



Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The + Operator Used on Strings

- ➢ The + operator can also be used to add string variables or text values together.
- ➢ To add two or more string variables together, use the + operator.

Ex:

txt1="What a very";

txt2="nice day";

txt3=txt1+txt2;

Result:

After the execution of the statements above, the variable txt3 contains:

txt3="What a verynice day“

- ➢ To insert a space into the expression:
- ✓ txt3=txt1+" "+txt2;
- ✓ **Result:**

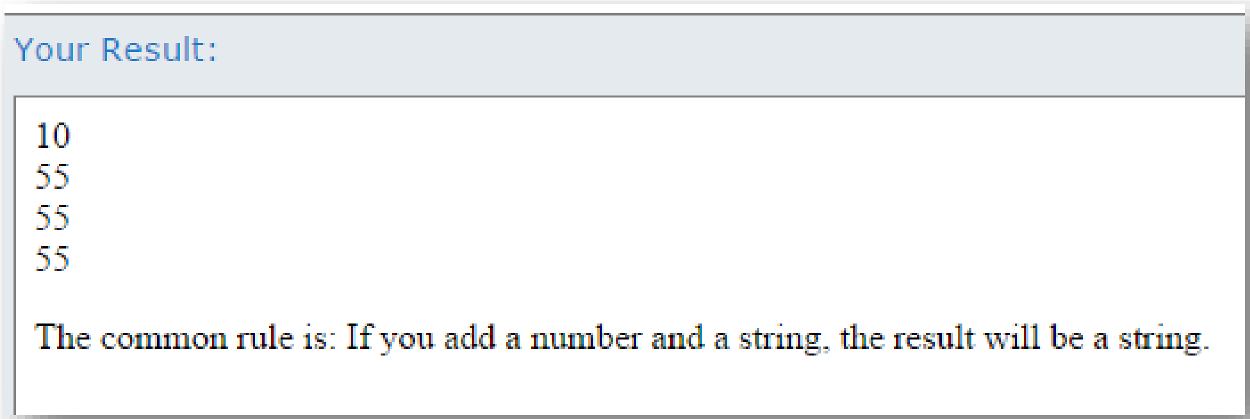
txt3="What a very nice day"

Adding Strings and Numbers

- If you add a number and a string, the result will be a string.

Ex:

```
<body>
<script type="text/javascript">
x=5+5;
document.write(x);
document.write("<br />");
x="5"+"5";
document.write(x);
document.write("<br />");
x=5+"5";
document.write(x);
document.write("<br />");
x="5"+5;
document.write(x);
document.write("<br />");
</script>
<p>The common rule is: If you add a number and a string,
the result will be a string.</p>
</body>
```

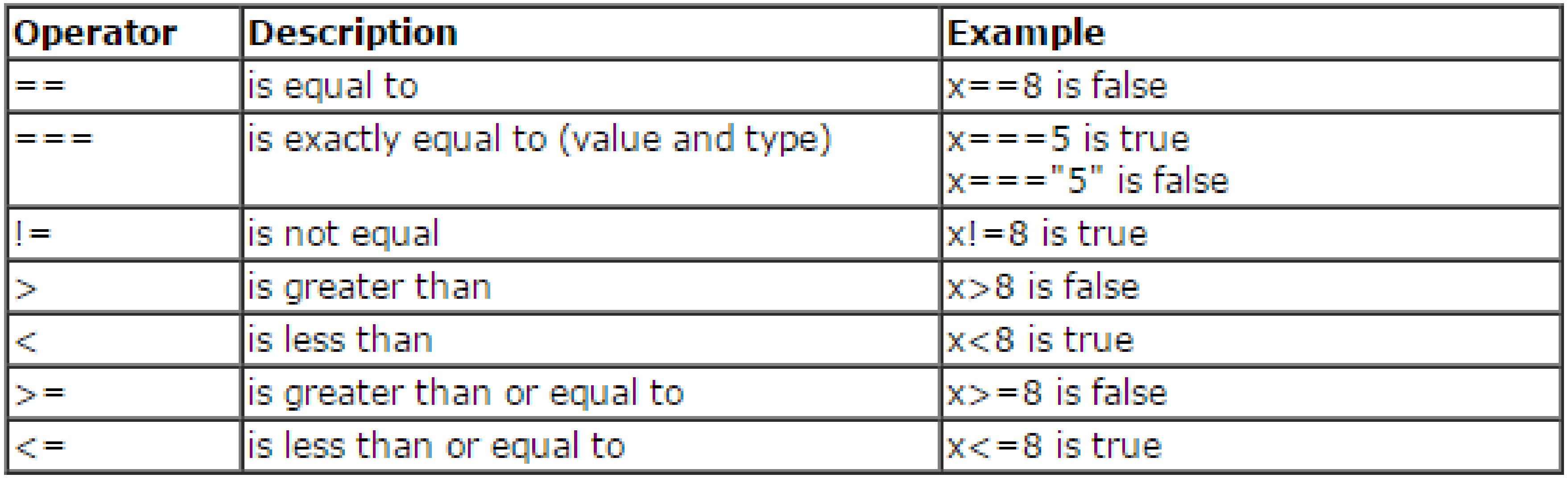


```
10
55
55
55

The common rule is: If you add a number and a string, the result will be a string.
```

Comparison Operators

- comparison operators are used in logical statements to determine equality or difference between variables or values.
- Given that x=5, the table below explains the comparison operators:



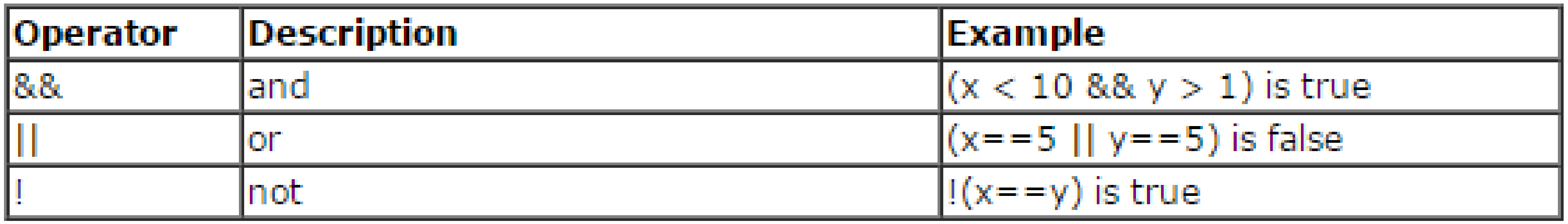
Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

- Ex:

```
if (age<18) document.write("Too young");
```

Logical Operators

- ➢ Logical operators are used to determine the logic between variables or values.
- Given that x=6 and y=3, the table below explains the logical operators:



Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

Conditional Operator

- ➢ JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.
- Syntax:

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

- Ex:

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

Result:

If the variable visitor has the value of "PRES", then the variable greeting will be assigned the value "Dear President " else it will be assigned "Dear".

JS Functions

- ✓ Functions are the core of any language, because they allow the encapsulation of statements that can be run anywhere and at any time.
- ✓ Functions are declared using the function keyword, followed by a set of arguments and then the body of the function. The basic syntax is as follows:

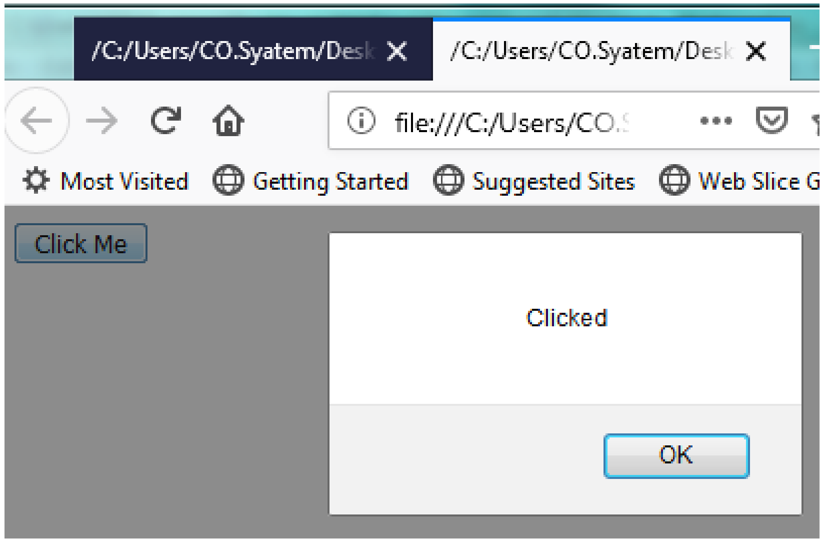
Syntax: function functionName(arg0, arg1,...,argN) {statements}

- ✓ Ex1: **function** sayHi(name, message) **{** alert(“Hello “ + name + “,” + message); **}**
- ✓ This function can then be called by using the function name, followed by the function arguments enclosed in parentheses (and separated by commas, if there are multiple arguments).
- ✓ The code to call the sayHi() function looks like this:

sayHi(“Nicholas”, “how are you today?”);

EVENTS

- ✓ Events are certain actions performed either by the user or by the browser itself. These events have names like **click , load ,** and **mouseover** .
- ✓ A function that is called in response to an event is called an event handler (event listener).
- ✓ Event handlers have names beginning with **“ on ”** , so an event handler for the click event is called **onclick** and an event handler for the load event is called **onload** .
- ✓ For example, to execute some JavaScript when a button is clicked, the following can be used:
- ✓ < input type=”button” value=”Click Me” onclick=”alert(‘Clicked’)” / >
- ✓ When this button is clicked, an alert is displayed. This interaction is defined by specifying the onclick attribute and assigning some JavaScript code as the value.



Clicked

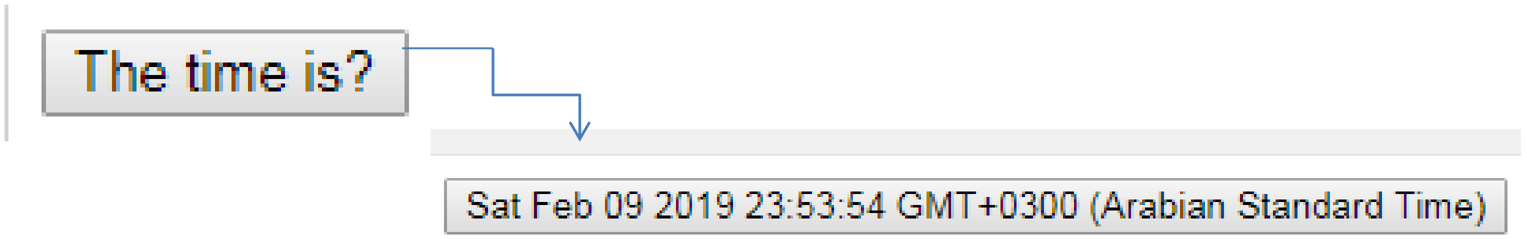
- ✓ **For more events please visit the link below:**

https://www.w3schools.com/jsref/dom_obj_event.asp

EVENTS Examples

Ex1:
```
<!DOCTYPE html>
<html>
<body>
<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>
<p id="demo"></p>
</body>
</html>
```

Ex2:
```
<!DOCTYPE html>
<html>
<body>
<button onclick="this.innerHTML=Date()">The time is?</button>
</body>
</html>
```



```
Sat Feb 09 2019 23:53:54 GMT+0300 (Arabian Standard Time)
```

EVENTS Examples

```
<!DOCTYPE html>
<html>
<body>
<p>A function is triggered when the
user is pressing a key in the input
field.</p>
<input type="text"
onkeydown="myFunction()">


<script>
function myFunction() {
  alert("You pressed a key inside the input
field");
                                   }
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<input type="text"
oncopy="myFunction()" value="Try to
copy this text">
<p id="demo"></p>


<script>
function myFunction() {

document.getElementById("demo").inne
rHTML = "You copied text!"
                                   }
</script>
</body>
</html>
```

EVENTS Examples

```
<!DOCTYPE html>
<html>
<head>
<script>
function m() {
  var x =
document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>
</head>
<body>
<h1 onclick="this.innerHTML =
'Ooops!'">Click on this text!</h1>
Enter your name: <input type="text"
id="fname" onchange="m()">
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<div onmouseover="mOver(this)"
onmouseout="mOut(this)"
style="background-
color:brown;width:120px;height:20px;padding:
40px;">
Mouse Over </div>
<script>
function mOver(obj) {
  obj.innerHTML = “Here we are"
}
function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>
</body>   </html>
```

Calling a Function with Arguments

```
ExampleFunction(argument1,argument2)

function ExampleFunction(var1,var2)

{

some code

}
```

The variables and the arguments must be in the expected order. The first variable is given the value of the first passed argument etc.

Ex: `<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>`

```
<script>

function myFunction(name,job)

{   alert("Welcome " + name + ", the " + job);    }

</script>
```

Functions With a Return Value

- Sometimes you want your function to return a value back to where the call was made.
- This is possible by using the return statement.
- When using the return statement, the function will stop executing, and return the specified value.

Ex1:

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

Ex2:

```
function add(num1, num2) {
sum = num1 + num2;
return sum;
}
var result = add(10, 20);       //30
alert(result);                  //30
```

Java Script Methods

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
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)

Method	Description
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970
getTimezoneOffset()	Returns the time difference between UTC time and local time, in minutes
getUTCDate()	Returns the day of the month, according to universal time (from 1-31)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time (four digits)
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)

Example1

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display todays day of the month.</p>
<button onclick="FindYear()">Try it</button>
<p id="demo"></p>

<script>
function FindYear() {
   var d = new Date();
   var n = d.getFullYear();
document.getElementById("demo").innerHTML = n;
}
</script>

</body>
</html>
```

Example2

```
<html>
<head>
<script>
function displayDate()
{
document.getElementById("d").innerHTML=Date();
}
</script>
</head><body>
<h1>My First JavaScript</h1>
<p id="d">This is a paragraph.</p>
<button type="button"
   onclick="displayDate()">Display
   Date</button>
</body>
</html>
```

String Methods

- ✓ The **length** property returns the length of a string:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.write(txt.length);            //26
```

- ✓ The **indexOf()** method returns the index of (the position of) the first occurrence of a specified text in a string:

```
var str = "Please locate where 'locate' occurs!";
document.write( str.indexOf("locate"));        //7
```

- ✓ The **search()** method searches a string for a specified value and returns the position of the match:

```
var str = "Please locate where 'locate' occurs!";
document.write(str.search("locate"));          //7
```

- ✓ **slice()** extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the start position, and the end position (end not included).

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be: Banana

String Methods

- ✓ The **replace()** method replaces a specified value with another value in a string:

```
 str = "Please visit Microsoft!";
Document.write(str.replace("Microsoft", "W3Schools"));     // Please visit W3Schools
```

- ✓ By default, the replace() function replaces only the first match and its is case sensitive.
- ✓ A string is converted to upper case with **toUpperCase()**

```
  var text1 = "Hello World!";       var text2 = text1.toUpperCase();
  Document.write(text2);  // text2 is text1 converted to upper   //HELLO WORLD!
```

- ✓ A string is converted to lower case with **toLowerCase()**

```
   var text1 = "Hello World!";       // String
   var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

- ✓ **concat()** joins two or more strings:

```
   var text1 = "Hello";               var text2 = "World";
   var text3 = text1.concat(" ", text2);
```

- ✓ **The trim()** method removes whitespace from both sides of a string:

```
  var str = "       Hello World!        ";
  alert(str.trim());
```

String Methods

✓The **charAt()** method returns the character at a specified index (position) in a string:

```
var str = "HELLO WORLD";
str.charAt(0);            // returns H
```

✓The **charCodeAt()** method returns the unicode of the character at a specified index in a string. The method returns a UTF-16 code (an integer between 0 and 65535).

```
var str = "HELLO WORLD";
str.charCodeAt(0);        // returns 72
```

✓property access **[ ]** on strings:

```
var str = "HELLO WORLD";
str[0];                   // returns H
```

✓It does not work in Internet Explorer 7 or earlier. It makes strings look like arrays (but they are not). If no character is found, [ ] returns undefined, while charAt() returns an empty string. It is read only. str[0] = "A" gives no error (but does not work!)

✓A string can be converted to an array with the split() method:

```
var txt = "a,b,c,d,e";   // String
Arr=txt.split(",");       // Split on commas
Arr[0]=‘e’;
```

Array Methods

✓ **toString()** converts an array to a string of (comma separated) array values.

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

Result: Banana,Orange,Apple,Mango

✓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.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result: Banana * Orange * Apple * Mango

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();            // Removes the last element ("Mango") from fruits
X= fruits.pop();                // the value of x is "Mango"
```

Array Methods

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

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

fruits.push("Kiwi");   // Adds a new element ("Kiwi") to fruits

- ✓ shift() and unshift()

var x = fruits.shift();   // the value of x is "Banana“   //works like pop()

fruits.unshift("Lemon"); // Returns the new length of the array //works like push()

- ✓ The **length** property provides an easy way to append a new element to an array:

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

fruits[fruits.length] = "Kiwi";   // Appends "Kiwi" to last index of fruits

- ✓ 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

- ✓ The **concat()** creates a new array by merging (concatenating) existing arrays:var var myGirls = ["Cecilie", "Lone"];

var myBoys = ["Emil", "Tobias", "Linus"];

var myChildren = myGirls.concat(myBoys);   // Cecilie,Lone,Emil,Tobias,Linus

Array Methods

✓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). While 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.

•The splice() method returns an array with the deleted items:

var removed = fruits.splice(2, 2, "Lemon", "Kiwi");

document.getElementById("demo3").innerHTML = "Removed Items:<br> " + removed;

✓With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

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

fruits.splice(0, 1); **//** Removes the first element of fruits //result is Orange,Apple,Mango

✓The **slice(From, To)** method slices out a piece of an array into a new array.

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

var citrus = fruits.slice(1); //slice from index 1   //Orange,Lemon,Apple,Mango

Javascript Math Object

- ➢ The JavaScript Math object allows you to perform mathematical tasks on numbers.

✓ **Math.round(x)** returns the value of x rounded to its nearest integer:

Math.round(4.7);   // returns 5   Math.round(4.4);   // returns 4

✓ **Math.pow(x, y)** returns the value of x to the power of y:

Math.pow(8, 2);   // returns 64

✓ **Math.sqrt(x)** returns the square root of x:

Math.sqrt(64);   // returns 8

✓ **Math.abs(x)** returns the absolute (positive) value of x:

Math.abs(-4.7);   // returns 4.7

✓ **Math.min()** and **Math.max()** can be used to find the lowest or highest value in a list of arguments:

Math.min(0, 150, 30, 20, -8, -200); // returns -200

✓ **Math.ceil(x)** returns the value of x rounded up to its nearest integer:

Math.ceil(4.4);   // returns 5

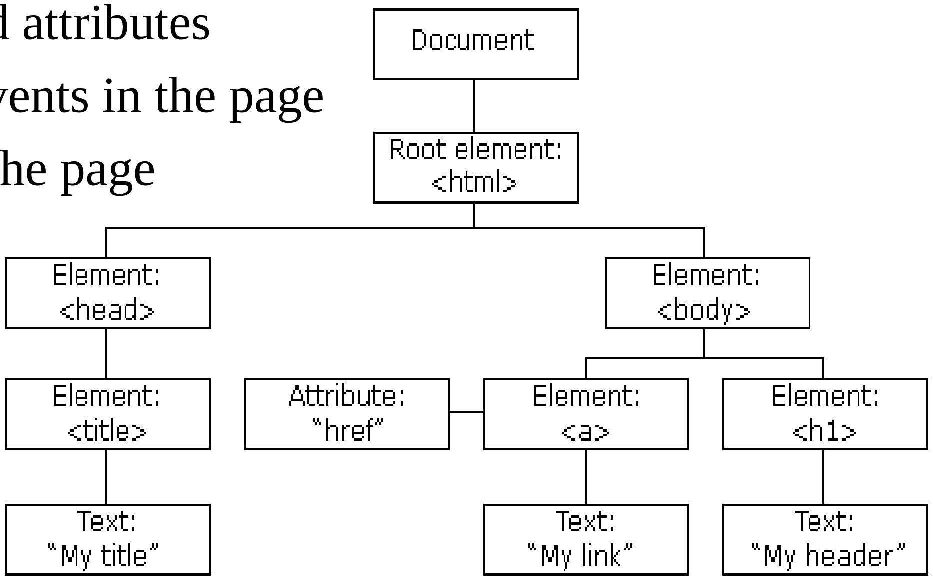
✓ **Math.floor(x)** returns the value of x rounded down to its nearest integer:

Math.floor(4.7);   // returns 4

✓ Math.sin(), Math.cos(), Math.PI, Math.random() // returns a random integer lower than 1

Document Object Model

- ✓ The **D**ocument **O**bject **M**odel (DOM) is an application programming interface (API) for HTML and XML documents.
- ✓ The DOM represents a document as a hierarchical tree of nodes, allowing developers to add, remove, and modify individual parts of the page.
- ✓ When a web page is loaded, the browser creates a Document Object Model of the page.
  - ✓ JavaScript can change all the HTML elements in the page
  - ✓ JavaScript can change all the HTML attributes in the page
  - ✓ JavaScript can change all the CSS styles in the page
  - ✓ JavaScript can remove existing HTML elements and attributes
  - ✓ JavaScript can add new HTML elements and attributes
  - ✓ JavaScript can react to all existing HTML events in the page
  - ✓ JavaScript can create new HTML events in the page



JavaScript - HTML DOM Methods & Properties

- ✓ HTML DOM **methods** are actions you can perform (on HTML Elements), (like add or deleting an HTML element). Ex: The getElementById Method
- ✓ HTML DOM **properties** are values (of HTML Elements) that you can set or change, (like changing the content of an HTML element). Ex: The innerHTML Property.

✓ Finding HTML Elements Methods

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

✓ Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

JavaScript - HTML DOM Methods & Properties

✓ **Changing HTML elements**

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attributes = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

✓ **Adding and Deleting Elements**

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Finding HTML Element

✓ **By ID**

```
document.getElementById(“mm").innerHTML =“Hello There!”;
```

✓ **By Tag Name**

```
<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="dd"></p>
<script>
var x = document.getElementsByTagName("p");          //x will creat an array of paragraphs
document.getElementById("dd").innerHTML =
'The text in first paragraph (index 0) is: ' + x[0].innerHTML;
</script>
```

✓ **By Class Name**

```
var x = document.getElementsByClassName("intro");
```

Changing HTML Elements and styles

- ✓ **Changing HTML Content**
- ✓ document.getElementById(“mm").innerHTML =“Hello There!”;

- ✓ **Changing the Value of an Attribute**

```
<img id=“imgg" src=“img1.type">
<script>
document.getElementById(“imgg").src = “img2.type";
</script>
```

- ✓ **Changing HTML Style**
- ✓ To change the style of an HTML element, use this syntax:
- ✓ document.getElementById(id).style.property = new style

```
document.getElementById("p2").style.color = "blue";
```

- ✓ **Changing HTML Style Using Events**

```
<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>
```

Changing HTML Styles Example

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDiv {
  border: thick solid blue;
}
</style>
</head>
<body>
<div id="myDiv">This is a div element.</div>
<br>
<button type="button" onclick="myFunction()">Change color of the left border</button>
<script>
function myFunction() {
  document.getElementById("myDiv").style.borderLeftColor = "red";
}
</script>
</body>  </html>
```

Screen Object

- ✓The screen object contains information about the visitor's screen.
- ✓**Screen Object Properties**

➢**availHeight**	Returns the height of the screen (excluding the Windows Taskbar)
➢**availWidth**	Returns the width of the screen (excluding the Windows Taskbar)
➢**height**	Returns the total height of the screen
➢**pixelDepth**	Returns the color resolution (in bits per pixel) of the screen
➢**width**	Returns the total width of the screen

Example1:

```
<script>
function myFunction() {
  var x = "Color Depth: " + screen.colorDepth + " bits per pixel";
  document.getElementById("demo").innerHTML = x; //Color Depth: 24 bits per pixel
```

Example2:

```
function myFunction() {
  var x = "Total Width: " + screen.width + "px";
  document.getElementById("demo").innerHTML = x; //Total Width: 1366px
}
```

The Window Object

- ✓The **window** object represents an open window in a browser.
  - The **open()** method opens a new browser window.
  - The **close()** method close the window.
  - **window.stop()** Stop the window from loading.

Syntax: window.open(URL, name , specs, replace)

```
window.open("https://www.google.com", "_blank",
"toolbar=yes,scrollbars=yes,resizable=yes,top=500,left=500,width=400,height=400"
);
```

- print() method prints the contents of the current window.
- The **window.location** object can be used to get the current page address (URL) and to redirect the browser to a new page.

✓**window.location.href**	//returns (URL) of the current page
✓**window.location.hostname**	//returns the domain name of the web host
✓**window.location.pathname**	//returns the path and filename of the page
✓**window.location.protocol**	//returns the web protocol used (http: or https:)
✓**window.location.assign**	//loads a new document

Window Object Example

```
<button onclick="openWin()">Open "myWindow"</button>
<button onclick="closeWin()">Close "myWindow"</button>
<button onclick="PrintFunction()">Print this page</button>
<p id="demo"></p>
<script>
function openWin() {
    w= window.open("https://www.google.com", "_blank",
"toolbar=yes,scrollbars=yes,resizable=yes,top=500,left=500,width=400,height=400" );
//w.document.write("<h1>This is 'myWindow'</h1>");
  w.document.title="New Window";
}
function closeWin()       {    w.close();           }
function PrintFunction() {    window.print();  }

x=window.location.href;
document.getElementById("demo").innerHTML = x;
</script>
```

prompt() method

- ✓The prompt() method displays a dialog box that prompts the visitor for input.
- ✓A prompt box is often used if you want the user to input a value before entering a page.
- ✓The prompt() method returns the input value if the user clicks "OK". If the user clicks "cancel" the method returns null.
- ✓Syntax :

prompt(text, defaultText)

- Text is String parameter and it’s Required, represent the text to be displayed in the dialog box
- defaultText is String parameter and it’s Optional, represents the default input text
- Ex:

```
var age = prompt("Please enter your Age", “25");
if (age != null) {
  document.getElementById("demo").innerHTML =  “Your age is  " + age+ “years old";
}
```

prompt() method example

```
var text;
var favDrink = prompt("What's your favorite drink?");
switch(favDrink) {
  case “Cappuccino":
    text = "Excellent choice! Cappuccino is good for your mood.";
    break;
  case “Coffee":
    text = “Coffee is my favorite too!";
    break;
  case “Miranda":
    text = "Really? Are you sure the Mirandais your favorite?";
    break;
  default:
    text = "I have never heard of that one..";
    break;
}
```

getAttributeNode() & hasAttribute() methods

- ✓The **getAttributeNode()** method returns the attribute node with the specified name of an element, as an Attr object.

Tip: Use the **attribute.value** property to return the value of the attribute node.

Tip: Use the **getAttribute()** method if you just want to return the attribute value.

- ✓Syntax

element.**getAttributeNode**(attributename)

- ✓**Return Value**: An Attr object, representing the specified attribute node.
- ✓If the attribute does not exist, the return value is null or an empty string ("").
- ✓The **hasAttribute**() method returns true if the specified attribute exists, otherwise it returns false.

Tip: Use **setAttribute**() to add a new attribute or change the value of an existing attribute on an element.

- ✓Syntax

element.**hasAttribute**(attributename)

- ✓**attributename** is **Required**. It’s the name of the attribute you want to check if exists.
- ✓**Return Value:** A Boolean, returns true if the element has attributes, otherwise false

getAttributeNode() & hasAttribute() Example

```
<body>
<a   id="m“   href="https://www.google.com"  target="_blank">GooGle</a>.
<button id="myBtn" onclick="myFunction()"> Try has attribute </button>
<button onclick="myFunction1()">Try get attribute</button>
 <script>
function myFunction() {
  var x = document.getElementById("m");
  if (x.hasAttribute("target")) {
    alert("target");  }
               }
function myFunction1() {
  var elmnt = document.getElementById("m");
  var attr = elmnt.getAttributeNode("target").value;                    //returns   “_blank”
  document.getElementById("demo").innerHTML = attr;
}
</script>
 </body>
```

attributes, name, specified & value properties

- ✓The **attributes property** returns a collection of the specified node's attributes.
- ✓Tip: Numerical indexing is useful for going through all of an element's attributes.

- ✓The **name property** returns the name of the attribute.

- ✓The **value property** sets or returns the value of the attribute.

- ✓The **specified property** returns true if the attribute is specified.
- ✓Returns true also if the attribute has been created but not been attached to an element yet. Otherwise it returns false.

Example

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the name of the button's first attribute.</p>
<button onclick="myFunction()" id="dd">Try it</button>                //1st button
<button onclick="myFunction()" class="dd">Try it</button>             //2nd button
<p id="d2"></p>
<script>
function myFunction() {
  var b = document.getElementsByTagName("BUTTON")[0];            //[0] first button
  var x = b.attributes[0].name;                        //[0] first attribute of first button onclick
 var y = b.attributes[0].value;                               //returns the value of onclick
var z = b.getAttributeNode("onclick").specified;         //returns true
document.getElementById("d2").innerHTML = “the 1st attribute of 1st button is ”+ x +
the value of 1st att. Is” +  y  + ”the onclick att. Is specifide?”  +   z ;   }
</script>  </body>  </html>
```

Number() method

- ✓The **Number()** function converts the object argument to a number that represents the object's value.
- ✓If the value cannot be converted to a legal number, NaN is returned.
- ✓Ex:

```
 Number(true);              // 1
  Number(false);            // 0
Number(“false”);            // NaN

  Number(new Date() );      //1551018642950//milliseconds since January 1, 1970
 Number(“888”);             //888
  Number("999 888");        //NaN
```

The parseInt() function parses a string and returns an integer.

The radix parameter is used to specify which numeral system to be used, for example, a radix of 16 (hexadecimal) indicates that the number in the string should be parsed from a hexadecimal number to a decimal number.

parseInt() method

- ✓The **parseInt()** function parses a string and returns an integer.
- ✓Syntax

parseInt(string, radix)

String IS **Required**. The string to be parsed

Radix IS **Optional**. A number (from 2 to 36) that represents the numeral system to be used

- ✓The radix parameter is used to specify which numeral system to be used, for example, a radix of 16 (hexadecimal) indicates that the number in the string should be parsed from a hexadecimal number to a decimal number.
- ✓**If the radix parameter is omitted, JavaScript assumes the following:**
- ✓If the string begins with "0x", the radix is 16 (hexadecimal)
- ✓If the string begins with any other value, the radix is 10 (decimal)

Note1: Only the first number in the string is returned!

Note2: Leading and trailing spaces are allowed.

Note3: If the first character cannot be converted to a number, parseInt() returns NaN.

parseInt() EXAMPLE

```
var a = parseInt("10") + "<br>";                10
  var b = parseInt("10.00") + "<br>";           10
  var c = parseInt ("10.33") + "<br>";          10
  var d = parseInt("34 45 66") + "<br>";        34
  var e = parseInt("   60   ") + "<br>";        60
  var f = parseInt("40 years") + "<br>";        40
  var g = parseInt("He was 40") + "<br>";       NaN
//WITH RADIX
  var h = parseInt("10", 10)+ "<br>";           10
  var i = parseInt("10",8)+ "<br>";             8
  var j = parseInt("10", 2)+ "<br>";            2
  var k = parseInt("0x10")+ "<br>";             16
  var l = parseInt("10", 16)+ "<br>";           16
```

parseFloat() EXAMPLE

- ✓The **parseFloat()** function parses a string and returns a floating point number.
- ✓This function determines if the first character in the specified string is a number. If it is, it parses the string until it reaches the end of the number, and returns the number as a number, not as a string.

Note1: Only the first number in the string is returned!

Note2: Leading and trailing spaces are allowed.

Note3: If the first character cannot be converted to a number, parseFloat() returns NaN.

✓**EXAMPLES**	**RETURED VALUE**
var a = parseFloat("10") + "<br>";	10
var b = parseFloat("10.00") + "<br>";	10
var c = parseFloat("10.33") + "<br>";	10.33
var d = parseFloat("34 45 66") + "<br>";	34
var e = parseFloat(" 60 ") + "<br>";	60
var f = parseFloat("40 years") + "<br>";	40
var g = parseFloat("He was 40") + "<br>";	NaN

The eval() function

- ✓ The **eval()** function evaluates or executes an argument.
- ✓ If the argument is an expression, **eval()** evaluates the expression. If the argument is one or more JavaScript statements, **eval()** executes the statements.
- ✓ Example:

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to evaluate/execute JavaScript code/expressions.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = 10;       var y = 20;
  var a = eval("x * y");
  var b = eval("2+ 2");
  var c = eval("x + 17");
  var res = a + b + c;
  document.getElementById("demo").innerHTML = res;                    //result 231
}
</script>                    </body>                    </html>
```

selectedIndex property

- ✓ The **selectedIndex** property sets or returns the index of the selected option in a drop-down list. The index starts at 0.
- ✓ **Note1**: If the drop-down list allows multiple selections it will only return the index of the first option selected.
- ✓ **Note2**: The value "-1" will deselect all options (if any).
- ✓ **Note3**: If no option is selected, the selectedIndex property will return -1.
- ✓ Syntax
- ➢ **Return** the selectedIndex property:

selectObject.selectedIndex

- ➢ **Set** the selectedIndex property:

selectObject.selectedIndex = number

selectedIndex example

```
<!DOCTYPE html>
<html>
<body>
Select a fruit and click the button:
<select id="mySelect">
  <option>Apple</option>
  <option>Orange</option>
  <option>Pineapple</option>
  <option>Banana</option>
</select>
<button type="button" onclick="myFunction()">Display index</button>
<script>
function myFunction() {
  var x = document.getElementById("mySelect").selectedIndex;
  var y = document.getElementById("mySelect").options;
  alert("Index: " + y[x].index + " is " + y[x].text);        }
</script>              </body>              </html>
```

DOM STYLES

Property	Description	CSS
alignContent	Sets or returns the alignment between the lines inside a flexible container when the items do not use all available space	3
alignItems	Sets or returns the alignment for items inside a flexible container	3
alignSelf	Sets or returns the alignment for selected items inside a flexible container	3
animation	A shorthand property for all the animation properties below, except the animationPlayState property	3
animationDelay	Sets or returns when the animation will start	3
animationDirection	Sets or returns whether or not the animation should play in reverse on alternate cycles	3
animationDuration	Sets or returns how many seconds or milliseconds an animation takes to complete one cycle	3
animationFillMode	Sets or returns what values are applied by the animation outside the time it is executing	3
animationIterationCount	Sets or returns the number of times an animation should be played	3

animationName	Sets or returns a name for the @keyframes animation	3
animationTimingFunction	Sets or returns the speed curve of the animation	3
animationPlayState	Sets or returns whether the animation is running or paused	3
background	Sets or returns all the background properties in one declaration	1
backgroundAttachment	Sets or returns whether a background-image is fixed or scrolls with the page	1
backgroundColor	Sets or returns the background-color of an element	1
backgroundImage	Sets or returns the background-image for an element	1
backgroundPosition	Sets or returns the starting position of a background-image	1
backgroundRepeat	Sets or returns how to repeat (tile) a background-image	1
backgroundClip	Sets or returns the painting area of the background	3
backgroundOrigin	Sets or returns the positioning area of the background images	3
backgroundSize	Sets or returns the size of the background image	3

border	**Sets or returns borderWidth, borderStyle, and borderColor in one declaration**	**1**
borderBottom	**Sets or returns all the borderBottom properties in one declaration**	**1**
borderBottomColor	**Sets or returns the color of the bottom border**	**1**
borderBottomLeftRadius	**Sets or returns the shape of the border of the bottom-left corner**	**3**
borderBottomRightRadius	**Sets or returns the shape of the border of the bottom-right corner**	**3**
borderBottomStyle	**Sets or returns the style of the bottom border**	**1**
borderBottomWidth	**Sets or returns the width of the bottom border**	**1**
borderCollapse	**Sets or returns whether the table border should be collapsed into a single border, or not**	**2**
borderColor	**Sets or returns the color of an element's border (can have up to four values)**	**1**
borderImage	**A shorthand property for setting or returning all the borderImage properties**	**3**

borderImageOutset	Sets or returns the amount by which the border image area extends beyond the border box	3
borderImageRepeat	Sets or returns whether the image-border should be repeated, rounded or stretched	3
borderImageSlice	Sets or returns the inward offsets of the image-border	3
borderImageSource	Sets or returns the image to be used as a border	3
borderImageWidth	Sets or returns the widths of the image-border	3
clip	Sets or returns which part of a positioned element is visible	2
color	Sets or returns the color of the text	1
cursor	Sets or returns the type of cursor to display for the mouse pointer	2
direction	Sets or returns the text direction	2
display	Sets or returns an element's display type	1
emptyCells	Sets or returns whether to show the border and background of empty cells, or not	2

filter	**Sets or returns image filters (visual effects, like blur and saturation)**	**3**
font	**Sets or returns fontStyle, fontVariant, fontWeight, fontSize, lineHeight, and fontFamily in one declaration**	**1**
fontFamily	**Sets or returns the font family for text**	**1**
fontSize	**Sets or returns the font size of the text**	**1**
fontStyle	**Sets or returns whether the style of the font is normal, italic or oblique**	**1**
fontVariant	**Sets or returns whether the font should be displayed in small capital letters**	**1**
fontWeight	**Sets or returns the boldness of the font**	**1**
height	**Sets or returns the height of an element**	**1**
justifyContent	**Sets or returns the alignment between the items inside a flexible container when the items do not use all available space.**	**3**
left	**Sets or returns the left position of a positioned element**	**2**
letterSpacing	**Sets or returns the space between characters in a text**	**1**

lineHeight	Sets or returns the distance between lines in a text	1
listStyle	Sets or returns listStyleImage, listStylePosition, and listStyleType in one declaration	1
listStyleImage	Sets or returns an image as the list-item marker	1
listStylePosition	Sets or returns the position of the list-item marker	1
listStyleType	Sets or returns the list-item marker type	1
margin	Sets or returns the margins of an element (can have up to four values)	1
marginBottom	Sets or returns the bottom margin of an element	1
opacity	Sets or returns the opacity level for an element	3
overflow	Sets or returns what to do with content that renders outside the element box	2
padding	Sets or returns the padding of an element (can have up to four values)	1
paddingBottom	Sets or returns the bottom padding of an element	1

position	**Sets or returns the type of positioning method used for an element (static, relative, absolute or fixed)**	**2**
textDecoration	**Sets or returns the decoration of a text**	**1**
textDecorationColor	**Sets or returns the color of the text-decoration**	**3**
textDecorationLine	**Sets or returns the type of line in a text-decoration**	**3**
textDecorationStyle	**Sets or returns the style of the line in a text decoration**	**3**
textIndent	**Sets or returns the indentation of the first line of text**	**1**
textJustify	**Sets or returns the justification method used when text-align is "justify"**	**3**
textOverflow	**Sets or returns what should happen when text overflows the containing element**	**3**
textShadow	**Sets or returns the shadow effect of a text**	**3**
textTransform	**Sets or returns the capitalization of a text**	**1**
top	**Sets or returns the top position of a positioned element**	**2**

transform	Applies a 2D or 3D transformation to an element	3
transformOrigin	Sets or returns the position of transformed elements	3
transformStyle	Sets or returns how nested elements are rendered in 3D space	3
transition	A shorthand property for setting or returning the four transition properties	3
transitionProperty	Sets or returns the CSS property that the transition effect is for	3
transitionDuration	Sets or returns how many seconds or milliseconds a transition effect takes to complete	3
transitionTimingFunction	Sets or returns the speed curve of the transition effect	3
transitionDelay	Sets or returns when the transition effect will start	3
verticalAlign	Sets or returns the vertical alignment of the content in an element	1
visibility	Sets or returns whether an element should be visible	2
whiteSpace	Sets or returns how to handle tabs, line breaks and whitespace in a text	1

width	**Sets or returns the width of an element**	**1**
wordBreak	**Sets or returns line breaking rules for non-CJK scripts**	**3**
wordSpacing	**Sets or returns the spacing between words in a text**	**1**
wordWrap	**Allows long, unbreakable words to be broken and wrap to the next line**	**3**
widows	**Sets or returns the minimum number of lines for an element that must be visible at the top of a page**	**2**
zIndex	**Sets or returns the stack order of a positioned element**	**2**