Static Web programming

4th class

Dr. Athraa Jasim Mohammed

The First course

Tenth Lecture

2020-2021

Introduction to JavaScript

JavaScript was released by Netscape and Sun Microsystems in 1995. However, JavaScript is not the same thing as Java.

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML
- pages JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)

Put a JavaScript into an HTML page

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.

```
<script type="text/JavaScript">
...some JavaScript

</script>
```

The **document.write** command is a standard JavaScript command for writing output to a page.

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page

```
<script type="text/javascript">
document.write("Hello World!");
</script>
```

The example below shows how to add HTML tags to the JavaScript:

```
document.write("<h1>Hello World!</h1>");
```

JavaScript is a sequence of statements to be executed by the browser.

JavaScript is Case Sensitive

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

JavaScript Statements

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do, JavaScript is a sequence of JavaScript statements. Each statement is executed by the browser in the sequence they are written.

This example will write a heading and two paragraphs to a web page:

```
<script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Blocks

JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and ends with a right bracket }, the purpose of a block is to make the sequence of statements execute together.

This example will write a heading and two paragraphs to a web page

```
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
}
</script>
```

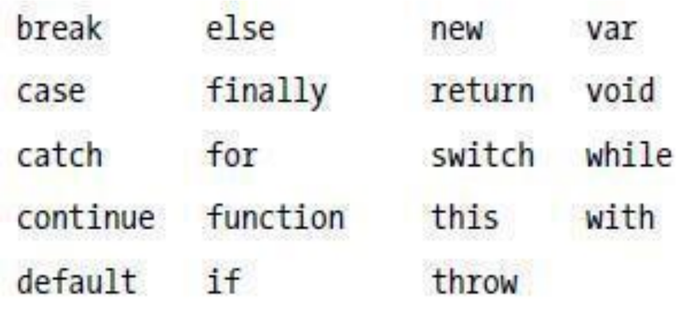
JavaScript Variables

Variables in JavaScript are much like those in any other language; you use them to hold values in such a way that the values can be explicitly accessed in different places in the code.

<u>Rules for JavaScript variable names:</u>

Variable names are case sensitive (y and Y are two different variables) Variable names must begin with a letter.

<u>Table JavaScript keyword</u>



break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	

<u>Creating JavaScript Variables</u>

Creating variables in JavaScript is most often referred to as "declaring" variables. You can declare JavaScript variables with the **<u>var</u>** keyword:

var x;
var carname;

However, you can also assign values to the variables when you declare them:

var x=5;
var name="Alex";

After the execution of the statements above, the variable **x** will hold the value **5**, and **name** will hold the value **Alex**.

When the string is used in a dialog window, the escape sequence, **<u>\n</u>**, is interpreted literally, and a <u>newline</u> is published:

var string_value = "This is the first line\nThis is the second line";

This results in:

This is the first line

This is the second line

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 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.

```
txt1="What a very";
txt2="nice day";
```

```
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a verynice day".

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

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

"What a very nice day"

Adding Strings and Numbers

The rule is: If you add a number and a string, the result will be a string!
Example

```
x=5+5;
document.write(x);

x="5"+"5";
document.write(x);

x=5+"5";
document.write(x);

x="5"+5;
document.write(x);
```

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false.

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

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 (Ternary) Operator

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

Syntax

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

Example

```
var age =16;

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

The output for this example is:

```
Too young
```

Static Web programming

4th class

Dr. Athraa Jasim Mohammed

The First course

Eleventh Lecture

2020-2021

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

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

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10
var d=new Date();
var time=d.getHours();
if (time<10)
  {
  document.write("<b>Good
morning</b>"); } </script>
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

If...else Statement

Use the if....else statement to execute some code if a condition is true and Another code if the condition is not true.

Syntax

```
if (condition)

  {
  code to be executed if condition is true
  }

else
  {

  code to be executed if condition is not true
  }
```

If...else if ...else Statement

Use the if ....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if condition1 and condition2 are not true
  }
```

The JavaScript Switch Statement

Syntax

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

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

Example

```
 <script type="text/javascript">
 //Monday=1, Tuesday=2, etc.

Day=1;
 switch (Day)
{
case 1:
  document.write("Monday ");

  break;
 case 5:
  document.write("Finally Friday");
  break;
 case 6:
  document.write("Super Saturday");
  break;
 case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

The continue Statement

The continue statement will break the current loop and continue with the next value

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
Alert("sometext");
```

```
<html>
<body>
<h1>Demo: alert()</h1>
<script>
            alert("This is alert box!"); // display string message

            alert(100); // display number

            alert(true); // display Boolean
</script>
</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to
verify or accept something.

When a confirm box pops up, the user will have to click either
"OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks
"Cancel", the box returns false.

Syntax

confirm("sometext");

```
<html>
<body>
        <h1>Demo: confirm()</h1>

  <script>
                 var userPreference;

                if (confirm("Do you want to save changes?") == true)
                        { userPreference = "Data saved
                uccessfully!"; } else {
                        userPreference = "Save Canceled!";
                }

                document.write( userPreference);

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

Prompt Box

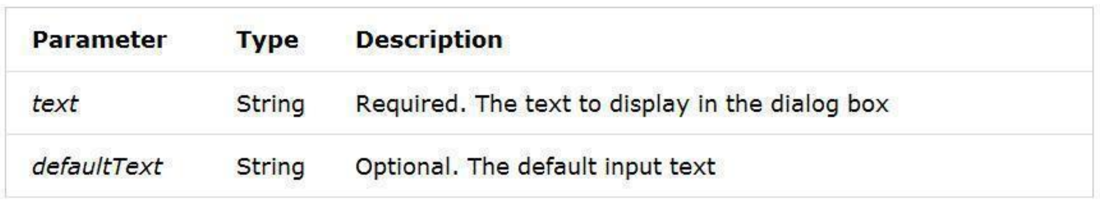
A prompt box is often used if you want the user to input a
value before entering a page.

When a prompt box pops up, the user will have to click either
"OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

prompt("text","defaulttext");



Parameter	Type	Description
text	String	Required. The text to display in the dialog box
defaultText	String	Optional. The default input text

```
<html>
<body>
        <h1>Demo: prompt()</h1>

        <script>
                var tenure = prompt("Please enter preferred tenure in years", "15");

                document("You have entered " + tenure + " years");
   </script>
</body>
</html>
```

<u>What output of this code</u>

```
<script>

var name = prompt("Please correct your e-
mail address:", "un@edu.iq");
document.write("Your e-mail address is ",
name);

</script>
```

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The Second course

First Lecture

2020-2021

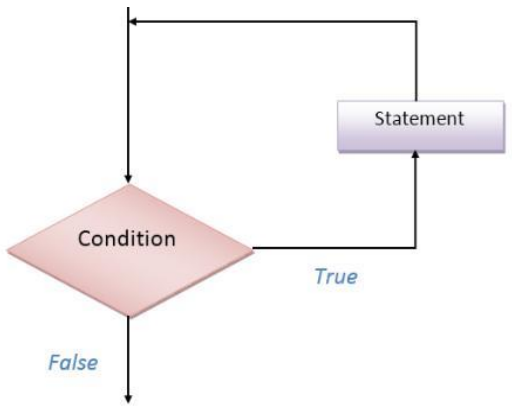
<u>parseInt()</u> or **<u>parseFloat()</u>**

are functions in JavaScript which can help you convert the values into integers or floats respectively.

```
var x = prompt("Enter a Value", "0");
 var y = prompt("Enter a Value", "0");
 var num1 = parseInt(x);
 var num2 = parseInt(y);
```

How to use Loop?

Loops are useful when you have to execute the same lines of code repeatedly, for a specific number of times or as long as a specific condition is true. Suppose you want to type a ‘Hello’ message 100 times in your webpage. Of course, you will have to copy and paste the same line 100 times. Instead, if you use loops, you can complete this task in just 3 or 4 lines.



Different Types of Loops

There are mainly four types of loops in JavaScript.

- for loop
- for/in a loop (explained later)
- while loop
- do…while loop

➢ For loop

Syntax:

```
for(statement1; statement2; statment3)
{
lines of code to be executed
}
```

- The statement1 is executed first even before executing the looping code. So, this statement is normally used to assign values to variables that will be used inside the loop.
- The statement2 is the condition to execute the loop.
- The statement3 is executed every time after the looping code is executed.

```
<script type="text/javascript">
            document.write("<b>Using for loops </b><br />");
           for (var i=0;i<9;i++)
           {
           document.write("*" + "<br />");
           }
    </script>
```

HW/write the code for printing the below figure?

```
*****
****
***
**
*
```

➢ While loop

Syntax:

```
while(condition)
{
lines of code to be executed
}
```

The "while loop" is executed as long as the specified condition is true. Inside the while loop, you should include the statement that will end the loop at some point of time.

```
<script type="text/javascript">
    document.write("<b>Using while loops </b><br />");
    var i = 0, j = 1, k;
    document.write(" series less than 40<br />");

    while(i<40)
    {
    document.write(i + "<br />");
    k = i+j;
    i = j;
    j = k;
    }
</script>
```

- **<u>do…while loop</u>**

Syntax:

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

The do…while loop is very similar to while loop. The only difference is that in do…while loop, the block of code gets executed once even before checking the condition

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The second course

Second Lecture

2020-2021

JavaScript Functions

A function is a piece of program wrapped in a value. Such values can be applied in order to run the wrapper program. For example, in a browser environment, the binding prompt holds a function that shows a little dialog box asking for user input. It is used like this:

prompt("Enter pass code","") ; Executing a function is called invoking, calling, or applying it. You can call a function by putting parentheses after an expression that produces a function value. Usually, you'll directly use the name of the binding that holds the function. The values between the parentheses are given to the program inside the function. In the example, the prompt function uses the string that we give it as the text to show in the dialog box. Values given to functions are called arguments. Different functions might need a different number or different types of arguments.

The prompt function isn't used much in modern web programming, mostly because you have no control over the way the resulting dialog looks but can be helpful in toy programs and experiments.

A function will be executed by an event or by a call to the function. To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function. You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <**head**> and in the <**body**> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function function-name(var1,var2,...,varX)
{
some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The {and the} defines the start and end of the function.

Note:

- A function with no parameters must include the parentheses () after the function name.
- Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example

```
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");   }
</script>
</head>
<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
```

```
2- <"script type = "text/javascript>
 //Function definition
function welcomeMsg(name)
{
document.write("Hello " + name + " welcome to conference ");
}
 //creating a variable
var nameVal = "Admin”;
 //calling the function
welcomeMsg(nameVal);
</script>
```

The Return Statement

The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement. The example below returns the product of two numbers (a and b):

Example

```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>
<body>
<script type="text/javascript">
document.write(product(4,3));
</script></body> </html>
```

The Lifetime of JavaScript Variables

If you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are <html>

Example

```
<head>
<script type="text/javascript">
Function myfunction(txt)
{ alert(txt);}
</script>  </head>  <body>
<form>
<input type="button" onclick="myfunction('Hello')" value="Call
function">
</form>
<p>By pressing the button above, a function will be called with "Hello"
as a parameter. The function will alert the parameter.</p>
</body> </html>
```

The HTML specification refers to these as intrinsic events and defines 18 as listed below:

Event Handler	**Event that it handles**
onChange	User has changed the object, then attempts to leave that field (i.e. clicks elsewhere).
onClick	User clicked on the object.
onDblClick	User clicked twice on the object.
onKeydown	A key was pressed over an element.
onKeyup	A key was released over an element.
onKeypress	A key was pressed over an element then released.
onLoad	The object has loaded.
onMousedown	The cursor moved over the object and mouse/pointing device was pressed down.
onMouseup	The mouse/pointing device was released after being pressed down.
onMouseover	The cursor moved over the object (i.e. user hovers the mouse over the object).
onMousemove	The cursor moved while hovering over an object.
onMouseout	The cursor moved off the object
onReset	User has reset a form.
onSelect	User selected some or all of the contents of the object. For example, the user selected some text within a text field.
onSubmit	User submitted a form.
onUnload	User left the window (i.e. user closes the browser window).

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The second course

Third Lecture

2020-2021

Array

JavaScript provides a data type specifically for storing sequences of values. It is called an *array* and is written as a list of values between square brackets, separated by commas. An array is a special variable, which can hold more than one value, at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

1- let listOfNumbers = [2, 3, 5, 7, 11];

2- cars1="Saab";
cars2="Volvo";
cars3="BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array! An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own ID so that it can be easily accessed.

Create an Array

An array can be defined in three ways. The following code creates an Array object called myCars

1:

```
var myCars=new Array(); // regular array (add an optional integer
myCars[0]="Saab";       // argument to control array's size)
myCars[1]="Volvo";
myCars[2]="BMW";
```

2:

```
var myCars=new Array("Saab","Volvo","BMW"); // condensed array
```

3:

```
var myCars=["Saab","Volvo","BMW"]; // literal array
```

Note: If you specify numbers or true/false values inside the array then the variable type will be Number or Boolean, instead of String.

Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0. The following code line:

```
document.write(myCars[0]);
```

will result in the following output: Saab

Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
myCars[0]="Opel";
```

Now, the following code line:

```
document.write(myCars[0]);
```

will result in the following output: Opel

Arrays - concat()

```
var parents = ["Jani", "Tove"];
var children = ["Cecilie", "Lone"];
var family = parents.concat(children);
document.write(family);
```

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

Here is another example:

document.write ("You \& I are student");

The example above will produce the following output:

You & I are students!

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	**Outputs**
\'	**single quote**
\"	**Double quote**
\&	**Ampersand**
\\	**Backslash**
\n	**new line**
\b	**Backspace**

String

We can read properties like length and toUpperCase from string values.

1- Return the length of a string

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

2- Style strings

```
var txt = "Hello World!";

document.write("<p>Big: " + txt.big() + "</p>");

document.write("<p>Bold: " + txt.bold() + "</p>");

document.write("<p>Italic: " + txt.italics() + "</p>");
```

3- The toLowerCase() and toUpperCase() methods

```
var txt="Hello World!";

document.write(txt.toLowerCase() + "<br />");

document.write(txt.toUpperCase());
```

4- The match() method

The match() method searches a string for a match against a regular expression, and returns the matches, the method will return only the first match in the string.

```
var str="Hello world!";

document.write(str.match("world")+"<br />");//output: world

document.write(str.match("World"));// output :Null
```

5- Replace characters in a string - replace()

```
var str="Visit Microsoft!";

document.write(str.replace("Microsoft"," Schools"));
```

6- The indexOf() method:

Return the position of the first found occurrence of a specified value in a string.

- This method is used to return the position of the first occurrence of a specified value inside a string variable.
- The return value of this method is -1 in case the specified value is not found.
- The indexOf() method is case sensitive.

```
var str="Hello world!";

document.write(str.indexOf("d") + "<br />");//10

document.write(str.indexOf("world")); //6

var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];
```

```
Document.write(beasts.indexOf('bison')); // expected output: 1

Document.write(beasts.indexOf('giraffe'));// expected output:-1
```

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The second course

Fourth Lecture

2020-2021

Create a Date Object

Objects and arrays (which are a specific kind of object) provide ways to group several values into a single value. Conceptually, this allows us to put a bunch of related things in a bag and run around with the bag, instead of wrapping our arms around all of the individual things and trying to hold on to them separately.

The Date object is used to work with dates and times. Date objects are created with the Date() constructor.

new Date() // current date and time

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object..

Some examples of instantiating a date:

```
var today = new Date();
var d1 = new Date("October 13, 1975 11:13:00");
var d2 = new Date(79,5,24);
var d3 = new Date(79,5,24,11,33,0);
```

Set and Get Dates

We can easily manipulate the date by using the methods available for the Date object. In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Compare Two Dates

The Date object is also used to compare two dates. The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();
if (myDate>today)
  {
  alert("Today is before 14th January 2010");
  }
else {
  alert("Today is after 14th January 2010");
  }
```

Method of object set and get date

Where get is used to retrieve a specific component from a date, set is used to modify components of a date.

Function	Description	Returned Values
getDate()	Day of the month	1-31
getDay()	Day of the week (integer)	0-6
getFullYear()	Year (full four digit)	1900+
getHours()	Hour of the day (integer)	0-23
getMinutes()	Minutes (since last hour)	0-59
getMonth()	Month	0-11
getSeconds()	Seconds (since last minute)	0-59
getYear()	Year	0-99 for years
setDate()	Sets the day, given a number between 1-31	Date in milliseconds
setFullYear()	Sets the year, given a four digit number	Date in milliseconds
setHours()	Sets the hour, given a number between 0-23	Date in milliseconds
setMinutes()	Sets the minutes, given a number between 0-59	Date in milliseconds
setMonth()	Sets the month, given a number between 0-11	Date in milliseconds
setSeconds()	Sets the seconds,l given a number between 0-59	Date in milliseconds

setYear()	Sets the year, given either a two digit or four digit number	Date in milliseconds

JavaScript Math Object

round() Parameters: This function accepts a single parameter, var. It is the number which you want to **round** off. Return Value: The Math.**round()** function returns the value of the given number **rounded** to the nearest integer.

document.write(Math.round(0.60) + "<br />");

random()
The Math.random() function is used to return a floating-point random number between range [0,1) , 0 (inclusive) and 1 (exclusive).This random number can then be scaled according to the desired range

```
document.write(Math.random() + "<br />");

//return a random integer between 0 and 10

document.write(Math.floor(Math.random()*10));
```

max()
use max() to return the number with the highest value of two specified numbers.

```
document.write(Math.max(0,150,30,20,38) + "<br />");

document.write(Math.max(-5,10) + "<br />");
```

min()
use min() to return the number with the lowest value of two specified numbers.

```
document.write(Math.min(-5,-10) + "<br />");
document.write(Math.min(1.5,2.5));
```

charAt

charAt() gives you the character at a certain position. For instance, when you do

```
var b = 'I am a JavaScript hacker.'
document.write(b.charAt(5))
```

Split

split() is a specialized method that you need sometimes. It allows you to split a string at the places of a certain character. You must put the result in an array, not in a simple variable.

Note split() does not work in Netscape 2 and Explorer 3.

Let's split b on the spaces

```
var b = 'I am a JavaScript hacker.'
var temp = new Array();
temp = b.split(' ');
```

Now the string has been split into 5 strings that are placed in the array temp. The **spaces themselves are gone.**

```
temp[0] = 'I';
temp[1] = 'am';
temp[2] = 'a';
temp[3] = 'JavaScript';
temp[4] = 'hacker.';
```

String Search Function

The search() method searches a string for a specified value and returns the position of the match. If there is a match, it will return the position in the string where the match was found and returns -1 if no match is found. The search value can be a string or a regular expression.

Search Function Regular Expression

The most important thing to remember when creating a regular expression is that it must be surrounded with slashes ***/regular expression/***. With that knowledge let's search a string to see if a common name "Alex" is inside it.

```
<script type="text/javascript">
var myRegExp = /Alex/;
var string1 = "Today John went to the store and talked with Alex.";
var matchPos1 = string1.search(myRegExp);
if(matchPos1 != -1)
```

```
        document.write("There was a match at position " +
matchPos1);
else
        document.write("There was no match in the first string");
</script>
```

JavaScript Form Validation

There's nothing more troublesome than receiving orders, guestbook entries, or other form submitted data that are incomplete in some way. You can avoid these headaches once and for all with JavaScript's amazing way to combat bad form data with a technique called "form validation".
JavaScript **document.getElementById**

JavaScript getElementById

There's an easy way to access any HTML element, and it's through the use of *id* attributes and the *getElementById* function.
If you want to quickly access the value of an HTML input give it an *id* to make your life easier. This small script below will check to see if there is any text in the text field "myText". The argument that *getElementById* requires is the *id* of the HTML element you wish to utilize.

```
<script type="text/javascript">
function notEmpty(){
        var myTextField = document.getElementById('myText');
        if(myTextField.value != "")
                alert("You entered: " + myTextField.value)
        else
                alert("Would you please enter some text?")
}
</script>
<input type='text' id='myText' />
<input type='button' onclick='notEmpty()' value='Form Checker' />
```

The quickest way to check if an input's string value is all numbers is to use a regular expression **/^[0-9]+$/** that will only match if the string is all .numbers and is at least one character long.

JavaScript Code:
// If the element's string matches the regular expression it is all numbers.

```
var numericExpression = /^[0-9]+$/;
```

We wanted to see if the input's string was all numbers so we made a regular expression to check for numbers [0-9] and stored it as **numericExpression**.
Inside each string is a function called a match that you can use to see if the string matches a certain regular expression. We accessed this function like so: **elem.value.match(expression here)**.

We wanted to see if the input’s string was all numbers so we made a regular expression to check for numbers [0-9] and stored it as numericExpression.

We then used the match function with our regular expression. If it is numeric then match will return true, making if statement pass the test and function **isNumeric** will also return true. However, if the expression fails because there is a letter or other character in our input's string then we'll display our **helperMsg** and return false.

```
if(elem.value.match(numericExpression)){
        return true;
  else    return false;
```

Checking for All Letters

This function will be identical to isNumeric except for the change to the regular expression we use inside the match function. Instead of checking for numbers we will want to check for all letters.
If we wanted to see if a string contained only letters we need to specify an expression that allows for both lowercase and uppercase letters:
/^[a-zA-Z]+$/ .

JavaScript Code:
// If the element's string matches the regular expression it is all letters

```
function isAlphabet(elem, helperMsg){
        var alphaExp = /^[a-zA-Z]+$/;
        if(elem.value.match(alphaExp)){
                return true;
        }else{          alert(helperMsg);   return false; } }
```

Form Validation - Restricting the Length

Being able to restrict the number of characters a user can enter into a field is one of the best ways to prevent bad data. For example, if you know that the zip code field should only be 5 numbers you know that 2 numbers is not sufficient.
Below we have created a lengthRestriction function that takes a text field and two numbers. The first number is the minimum number of characters and the second is the maximum number of a characters the input can be. If you just want to specify an exact number then send the same number for both minimum and maximum.

JavaScript Code:

```
function lengthRestriction(elem, min, max){
	var uInput = elem.value;
	if(uInput.length >= min && uInput.length <= max){
		return true;
	}else{
		alert("Please enter between " +min+ " and " +max+ "
characters");
		elem.focus();
		return false;
	}
}
```

Form Validation - Email Validation

And for our grand finale we will be showing you how to check to see if a user's email address is valid. Every email is made up for 5 parts:
A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores The at symbol @ A combination of letters, numbers, hyphens, and/or periods A period The top level domain (com, net, org, us, gov, ...)
.
Valid Examples:
bobby.jo@filltank.net
jack+jill@hill.com
the-stand@steven.king.com
Invalid Examples:
@deleted.net - no characters before the @
free!dom@bravehe.art - invalid character !
shoes@need_shining.com - underscores are not allowed in the domain name

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The second course

Fifth Lecture

2020-2021

The innerHTML

The innerHTML property can be used to modify your document's HTML on the fly. When you use innerHTML, you can change the page's content without refreshing the page. This can make your website feel quicker and more responsive to user input. The innerHTML property is used along with getElementById within your JavaScript code to refer to an HTML element and change its contents, despite this, it is supported in all major browsers, which stands for Document Object Model, is the hierarchy that you can use to access and manipulate HTML objects from within your JavaScript.

The `innerHTML` Syntax

The syntax for using `innerHTML` goes like this:

```
document.getElementById('{ID of
element}').innerHTML = '{content}';
```

In this syntax example, `{ID of element}` is the ID of an HTML element and `{content}` is the new content to go into the element.

Basic `innerHTML` Example

Here's a basic example to demonstrate how `innerHTML` works.

Code:

```
<html>
<head>
<script type="text/javascript">
function Msg1(){
  document.getElementById('myText').innerHTML = 'Thanks!';
}
function Msg2(){
  document.getElementById('myText').innerHTML = 'Try message 1
again...';}
</script>
</head>
<body><form>
<input type="button" onclick="Msg1()" value="Show Message 1" />
<input type="button"  onclick="Msg2()" value="Show Message 2" />
```

```
<p id="myText"></p>
</form>
</body>
</html>
```

Result:
Thanks!

This code includes two functions and two buttons. Each function displays a different message and each button triggers a different function. In the functions, the `getElementById` refers to the HTML element by using its ID. We give the HTML element an ID of "myText" using `id="myText"`.

So in the first function for example, you can see that

```
document.getElementById('myText').innerHTML = 'Thanks!';
```

is setting the innerHTML of the "myText" element to "Thanks!".

In the previous, we used an event handler to trigger off a call to our function. There are 18 event handlers that you can use to link your HTML elements to a piece of JavaScript.

When you write a JavaScript function, you will need to determine when it will run. Often, this will be when a user does something like a click or hover over something, submit a form, double clicks on something etc.

Using JavaScript, you can respond to an event using event handlers. You can attach an event handler to the HTML element for which you want to respond to when a specific event occurs.

For example, you could attach JavaScript's on Mouseover event handler to a button and specify some JavaScript to run whenever this event occurs against that button.

Example 2: `innerHTML` With User Input
Here's an example of using `innerHTML` with a text field. Here, we display whatever the user has entered into the input field.

Code:

```
<script type="text/javascript">
function showMsg(){
  var userInput = document.getElementById('userInput').value;
  document.getElementById('userMsg').innerHTML = userInput;
}
</script>
<input type="text" maxlength="40" id="userInput"
  onkeyup="showMsg()" value="Enter text here..." />
<p id="userMsg"></p>
```

Result:

Example 3: Formatting with `getElementById`

In this example, we use the `getElementById` property to detect the color that the user has selected. We can then use `style.color` to apply the new color. In both cases, we refer to the HTML elements by their ID (using `getElementById`.)

Code:

```
<script type="text/javascript">
function changeColor(){
  var newColor = document.getElementById('colorPicker').value;
          document.getElementById('colorMsg').style.color = newColor;
}
</script>
<p id="colorMsg">Choose a color...</p>
<select id="colorPicker" onchange="JavaScript:changeColor()">
<option value="#000000">Black</option>
<option value="#000099">Blue</option>
<option value="#990000">Red</option>
<option value="#009900">Green</option>
</select>
```

Result:
Choose a color...

Sometimes, you may need to call some JavaScript from within a link. Normally, when you click a link, the browser loads a new page (or refreshes the same page). This might not always be desirable. For example, you might only want to dynamically update a form field when the user clicks a link.

Example: JavaScript "On Double Click"

You could just have easily used another event to trigger the same JavaScript. For example, you could run JavaScript only when the double clicks the HTML element. We can do this using the `onDblClick` event handler.

Code:
<input type="button" onDblClick="alert('Hey, remember to tell your friends about Quackit.com!');" value="Double Click Me!" />

To prevent the load from refreshing, you could use the JavaScript void() function and pass a parameter of (zero).

Example of void(0): We have a link that should only do something (i.e. display a message) upon two clicks (i.e. a double click). If you click once, nothing should happen. We can specify the double click code by using JavaScript's "ondblclick" method. To prevent the page reloading upon a single click, we can use "JavaScript:void(0);" within the anchor link.

Code:

```
<a href="JavaScript:void(0);"
ondblclick="alert('Well done!')">Double Click
Me!</a>
```

Result:
Double Click Me!

Same Example, but without void(0):

Look at what would happen if we didn't use "JavaScript:void(0);" within the anchor link...

Code:

```
<a href="" ondblclick="alert('Well
done!')">Double Click Me!</a>
```

Result:
Double Click Me!

Did you notice the page refresh as soon you clicked the link. Even if you double clicked and triggered the "ondbclick" event, the page still reloads!

Note: Depending on your browser, your browser might have redirected you to the "/javascript/tutorial/" index page. Either way, JavaScript's "void()" method will prevent this from happening.
Void(0) can become useful when you need to call another function that may have otherwise resulted in an unwanted page refresh.

Refresh code In JavaScript, you refresh the page using `location.reload`.

This code can be called automatically upon an event or simply when the user clicks on a link.

Example JavaScript Refresh code

code:

```
<input type="button" value="Reload Page" onClick =
"document.location.reload (true)">
```

You can use JavaScript to automatically open print dialogue box so that users can print the page. Although most users know they can do something like "File > Print", a "Print this page" option can be nice, and may even encourage users to print the page. Also, this code can be triggered automatically upon loading a printer friendly version.

Creating a "Print this page"

The following code creates a hyperlink and uses the Javascript print function to print the current page:

```
<a href="JavaScript:window.print();">Print this
page</a>
```

7

Dynamic Web programming

4th class

Dr. Athraa Jasim Mohammed

The second course

Sixth Lecture

2020-2021

Open a new window in JavaScript

One of the more common requests I see is how to open a new Javascript window with a certain feature/argument. The Window.open method has been available since Netscape 2.0 (Javascript 1.0), but additional window decoration features have been added to the syntax since then.

Window.open method

The syntax to open a popup is: window.open(url, name, params):

url

An URL to load into the new window.

name

A name of the new window. Each window has a window.name, and here we can specify which window to use for the popup. If there’s already a window with such name – the given URL opens in it, otherwise a new window is opened.

params

The configuration string for the new window. It contains settings, delimited by a comma. There must be no spaces in params, for instance: width:200,height=100.

Example
```
let params = `scrollbars=no, resizable=no, status=no,
location=no, toolbar=no, menubar=no, width=0,height=0, left=-
1000, top=-1000`;

open('/', 'test', params);
```

```
<html>
<body>
<p>Click the button to open a new browser window.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  window.open("https://www.yahoo.com");}
</script>
</body>
</html>
```

Write to a new window

How do I write script-generated content to another window?
To write script-generated content to another window, use the method ***winRef*.document.write()**, as returned by the `window.open()` method.

To make sure that your script's output actually shows up in the other window, use ***winRef*.document.close()** after writing the content.
As an example, consider the following function that opens a new window with the title **Console** and writes the specified `content` to the new

Examples

```
<HTML>
<HEAD>
<TITLE>Writing to Subwindow</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
   newWindow = window.open("","","status,height=200,width=300")
}
function subWrite() {
   if (newWindow.closed) {
      makeNewWindow()
   }
   newWindow.focus()
   var newContent = "<HTML><HEAD><TITLE>A New Doc</TITLE></HEAD>"
   newContent += "<BODY BGCOLOR='coral'><H1>This document is brand new.</
H1>"
   newContent += "</BODY></HTML>"
   newWindow.document.write(newContent)
   newWindow.document.close() // close layout stream
}
</SCRIPT>
</HEAD>
<BODY onLoad="makeNewWindow()">
<FORM>
<INPUT TYPE="button" VALUE="Write to Subwindow" onClick="subWrite()">
</FORM>
</BODY>
</HTML>
```

Moving and resizing windows

When you have created a window, you can use Javascript to **move** it or **resize** it.

- **Moving windows**

A window object has two methods for moving it**: `moveTo`** and **`moveBy`**. Both take two numbers as arguments.
The first function moves the window to the specified coordinates, measured in pixels from the top left corner of the screen. The first argument is the horizontal, or X, coordinat, and the second is the vertical, or Y, coordinate.
window.moveTo(*x*, *y*)
x is the horizontal coordinate to be moved to.
y is the vertical coordinate to be moved to.

Example
```
<script type="text/javascript">
function open_and_move1(){
win2=window.open("page.htm","","width=300,height=300")
win2.moveTo(0,0)
}
</script
```
The second method changes the current coordinates by the given amounts, which can be negative to move the window left or up.
Not all browsers allow you to move a window.

window.moveBy(deltaX, deltaY**)**
deltaX is the amount of pixels to move the window horizontally. Positive values are to the right, while negative values are to the left.
deltaY is the amount of pixels to move the window vertically. Positive values are down, while negative values are up.

Example
```
<script type="text/javascript">
function open_and_move2(){
   win3=window.open("page.htm","","width=600,height=500")
   win3.moveTo(screen.width/2-300,screen.height/2-250)
}
</script>
```
The center coordinates of the screen in the second example is calculated by determining the screen's dimensions, dividing that by 2, and subtracting half of either the window's width/height from it. In other words, in order to center a window,

- **Resizing windows**

Similar to the methods for moving, there are two methods for resizing a window: **`resizeTo`** and **`resizeBy`**. Like for moving, they either set the size absolutely or modify the size relatively to the current size.

Most browsers' standard security setting will not let you resize a window to less than 100 pixels in any direction.

```
window.resizeBy (100,-100);
```

Closing windows

Closing a window is simple when it works. All windows have a `close` method, so you can attempt to close any window you have a reference to.

```
myWindow.close();
```

Checking whether window has been closed

Given a reference to a window, it is possible to see whether the window has been closed.

```
    if  (myWindow.closed)  {  /* do something, e.g., open it again */  }
```

External JavaScript

You can place all your scripts into an external file (with a .js extension), then link to that file from within your HTML document. This is handy if you need to use the same scripts across multiple HTML pages, or a whole website.
To link to an external JavaScript file, you add a `src` attribute to your HTML `script` tag and specify the location of the external JavaScript file.

Linking to an external JavaScript file

<script type="text/javascript" src="external_javascript.js"></script>
Contents of your external JavaScript file
The code in **your .js file** should be no different to the code you would normally have placed in between the script tags. But remember, you don't need to create script tag again - it is already on the HTML page calling the external file! Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
   function msg(){
```

```
 alert("Hello Javatpoint");
}
```

Let’s include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
<html>
<head>
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```