Cairo University
Faculty of Engineering
Computer Engineering Department

WEB BUILDING

Browser Scripting

JavaScript

JavaScript

- A scripting language which is a lightweight programming language
- Designed to add interactivity to HTML pages
- It is usually embedded directly into HTML pages
- It is an **interpreted language** (means that scripts execute without preliminary compilation)
- Scripts in an HTML file are executed **on the client (in the browser)**

Why JavaScript?

- Add dynamic text into an HTML page .
- React to events: It can be set to execute when something happens
- Read and write HTML elements .
- Validate data .
- Detect the visitor's browser .
- Create cookies.

JavaScript in an HTML page

```
<html>
    <body>
        <script type="text/javascript">
            document.write("Hello World!");
        document.write("<h1> add HTML tags to the JavaScript </h1>");
        </script>
    </body>
</html>
```

Hello World!

add HTML tags to the JavaScript

JavaScript in an HTML page

- JavaScripts in an HTML page will be executed immediately **while** the page loads into the browser.

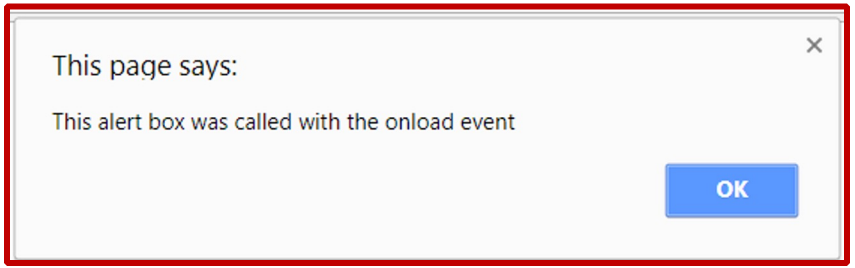
1. **In <head> :**
   - Scripts to be executed when they are called, or when an event is triggered, are placed in **functions**.
   - **Put your functions in the head section.** This way they are all in one place, and they do not interfere with page content.
2. **In <body> :**
   - If you don't want your script to be placed inside a function.
   - Or if your script should write page content.

Scripts in <head> - Example

```
<html>
<head>
    <script type="text/javascript">
      function message() {
            alert("This alert box was called with the onload event");
          }
    </script>
</head>

<body onload="message()">

</body>
</html>
```



External JavaScript

- If you want to run the same JavaScript on **several pages**, without having to re-write the same script on every page, you can write a JavaScript in an **external** file.
- Save the external JavaScript file with a .js file extension.
- The external script cannot contain the <script></script>

```
<html> <body>
    <script type="text/javascript" src="script-file.js">
    </script>
    <p>This script is in an external script file called "script-file.js".</p>
</body> </html>
```

JavaScript Statements

- JavaScript is Case Sensitive (Unlike HTML)
- The semicolon is **optional** (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement.

```
document.write ( "Hello Dolly" )
```

- **Note:** Using semicolons makes it possible to write multiple statements on one line.
  - Single-line comments start with //
  - Multi-line comments start with /* and end with */

Declaring Variables

- You can declare variables with the **var** keyword.

```
var x = 10;
var name = “XYZ”;
var y;
```

- If you assign values to variables that have not yet been declared, the variables will be **automatically declared.**

These statements:

```
x=10;
name=“XYZ";
```

Have **the same** effect as:

```
var x=10;
var name=“XYZ";
```

Re-declaring Variables

- If you re-declare a variable, it **will not lose its original value**.

```
var x=10;
var x;  // still equal 10
```

- After the execution of the statements above, the variable x will still have the value of 10.
- The value of x is not reset.

JavaScript Scope

Block Scope

Local Scope

Global Scope

Block Scope

Variables declared using **let** and **const** inside { } can not be accessed from outside the block

```
{
  let x = 2;
}
// x can NOT be used here
```

Variables declared using **var** can **NOT** have block scope

```
{
  var x = 2;
}
// x CAN be used here
```

Local Scope - Function Scope

Variables declared within a JavaScript function become local to the function. They can only be accessed from within the function

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Local Scope - Function Scope

- Variables with the same name can be used in different functions
- They are created when the function starts and deleted when the function is completed
- Each function creates a new scope
- You can declare them using let, const and var they are all quite similar

Global Scope

- A variable declared outside a function becomes Global

```
let carName = "Volvo";
// code here can use carName

function myFunction() {
// code here can also use carName
}
```

- They are declared outside any function using let, const and var
- In JS objects and functions are also variables can only be accessed in their scope

Global Scope - Automatically Global

- If you assign a value to a variable without declaration it will automatically become global

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

JavaScript Operators

- Given that **y=5**, this table explains **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

- **ators:**

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

JavaScript Operators

- Given that **x=5,** this table explains **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

- Given that **x=6 and y=3**, this table explains **logical operators:**

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

JavaScript Operators

- **+ Operator on Strings**
  - It can be used to add string variables or text values together (concatenate strings).
- To add two or more string variables together:

```
txt1 = “Good ” ;
txt2 = “Morning”;
txt3 = txt1 + txt2;
```

txt3 ← “Good Morning”

JavaScript Operators

- **+ Operator on Strings**

<u>Note:</u>

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

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

x ← "55"

JavaScript Operators

- **Conditional Operator**

  It is used to assign a value to a variable based on some condition.

Syntax:

variablename = (condition) ? value1 : value2

Example:

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

JS Conditional Statements

If Statement	If...else Statement	If...else if…else Statement

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

JS Conditional Statements

```
<html>
    <body>
            <script type="text/javascript">
                        var d = new Date();
                        var time = d.getHours();
                        if (time < 10) {      /* Checks the time on your browser if it is less than
    10 */
                document.write("<b>Good morning</b>");
                        } else {
                document.write("<b>Good day</b>");
            }
                        document.write("<br\><br\>Current time is: "+d);
     </script>
    </body>
</html>
```

Good day

Current time is: Mon Nov 02 2020 16:36:05 GMT+0200 (Eastern European Standard Time)

JS Conditional Statements

```
switch (n) {
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  otherwise
}
```

```
<script type="text/javascript">
var d = new Date();
theDay=d.getDay();
switch (theDay) {
        case 5:
          document.write("<b>Finally Friday</b>");
          break;
      case 6:
         document.write("<b>Super Saturday</b>");
         break;
      default:
         document.write("<b>I'm really looking forward to
                                    this weekend!</b>");
}
</script>
```

JS Popup Boxes

Alert box

- 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.

alert (“sometext”) ;

Confirm box

- 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. **confirm (“sometext”) ;**

Prompt box

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

prompt (“sometext” , “defaultvalue”) ;

JS Popup Boxes - Example

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

            function showAlert()
            {
                alert("Hello! I am an alert box!");
            }

            function show_confirm()
            {
                var r=confirm("Press a button!");
                if (r==true)
                  {
                  alert("You pressed OK!");
                  }
                else
                  {
                  alert("You pressed Cancel!");
                  }
            }

            function show_prompt()
            {
                var name=prompt("Please enter your name","Harry Potter");
                if (name!=null && name!="")
                  {
                  document.write("Hello " + name + "! How are you today?");
                  }
            }
        </script>
    </head>
```

```html
    <body>
        <br/>
        <br/>
        <input type="button" onclick="showAlert()" value="Show alert box" />
        <br/>
        <br/>
        <input type="button" onclick="show_confirm()" value="Show a confirm box" />
        <br/>
        <br/>
        <input type="button" onclick="show_prompt()" value="Show prompt box" />
    </body>
</html>
```

Show alert box

Show a confirm box

Show prompt box

JS Functions

- **Syntax:**

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

- **Example:**

```
function product ( a , b )
{
    return a*b;
}
```

JS Functions – return Statement 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>
<p>The script in the body section calls a function with two parameters (4 and 3).</p>
</body> </html>
```

12

The script in the body section calls a function with two parameters (4 and 3).

JS Loops-For Loop

- **Syntax:**

```
for(var=start; var<=end; var+=inc)
{
        code to be executed
}
```

```
<script type="text/javascript">
for (i = 0; i <= 5; i++)
{
  document.write("The number is " +
    i);
  document.write("<br />");
}
</script>
```

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

JS Loops-While Loop

- **Syntax:**

```
while (var<=endvalue)
 {
   code to be executed
 }
```

```
<script type="text/javascript">
i=0;
while (i<=5)
{
  document.write("The number is " + i);
  document.write("<br />");
  i++;
}
</script>
```

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

JS Loops-Do While Loop

- **Syntax:**

```
do
  {
    code to be executed
  }
while (var<=endvalue);
```

```
<script type="text/javascript">
i = 0;
do
{
  document.write("The number is " +
    i);
  document.write("<br />");
  i++;
} while (i <= 5);
</script>
```

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

JS Break and Continue Statements

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

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

```
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)  {
        if (i==3)
                break;              // what if we put continue? Will the output change?
        document.write( "The number is " + i );
                document.write("<br />");
}
</script>
```

The number is 0
The number is 1
The number is 2

JS For...In Statement

- **Syntax:**

  for (*variable* **in** *object*)
  {
  code to be executed
  }

```
<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
for ( x in mycars )
{   document.write(mycars[x] + "<br />");
}
</script>
```

Saab
Volvo
BMW

JS Events

- By using JavaScript, we have the ability to create dynamic web pages. **Events** are actions that can be detected by JavaScript.
- Every element on a web page has **certain events** which can trigger a JavaScript.
- Examples of events:
  - A mouse click
  - A web page or an image loading
  - Mousing over a hot spot on the web page
  - Selecting an input field in an HTML form
  - Submitting an HTML form

JS Events - Example

- **onSubmit event**

It is usually used to **validate** ALL **form fields** on client side before submitting it to the server.

- **Example:**

```
<form name="myForm" onsubmit="return validateForm()"
method="post">
```

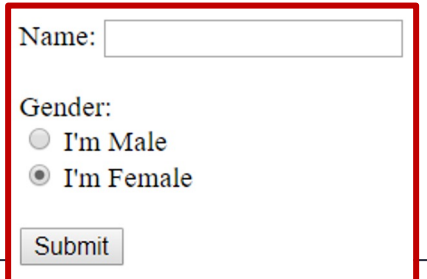
The **validateForm()** function is a user-defined function that will be called when the user clicks the submit button in the form. If the field values are not accepted, the return of validateForm() will cancel submitting the form.

JS “onsubmit” Event - Example

```
<html> <head> <script>
function validateForm() {
   var x = document.forms ["myForm"] ["fname"] . value;
   if (x == “”) {
      alert("Name must be filled out");
      return false;
   }
   var y = document.forms ["myForm"] ["rd_gender"] . value;
    // y now contains the value attribute of the checked radio button item
    // from the radio button group that has “rd_gender” name.
   alert("Name: " + x + ", Gender: " + y);
}
</script> </head>
```

JS “onsubmit” Event - Example (cont)

```
<body>
<form name="myForm" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname"> <br><br>
Gender: <br>
<label>
        <input type="radio" name="rd_gender"  value="Male"> I'm Male
</label> <br>
<label>
        <input type="radio" name="rd_gender"  value="Female" checked> I'm Female
</label> <br> <br>
<input type="submit" value="Submit">
</form>
</body> </html>
```



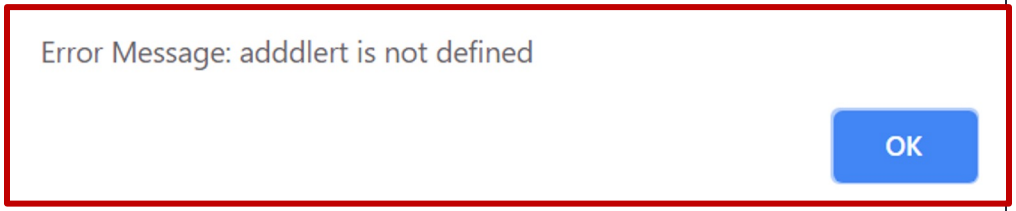
JS Try…Catch

- The try...catch statement allows you to test a block of code for errors.
- The try block contains the code to be run,
- The catch block contains the code to be executed if an error occurs.
- **Syntax:**

```
try
  {
      //Run some code here
  }
 catch(err)
{
      //Handle errors here
}
```

JS Try..Catch - Example

```
<html> <head> <script>
function message()
{   try
    {    adddlert("Welcome guest!");  //error syntax
    }
    catch ( err )
    {    alert ( "Error Message: " + err.message );
    }
}
</script> </head>
<body>
<button type="button" onclick="message()">Test</button>
</body> </html>
```



JS Throw Statement

- It allows you to **create an exception**. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.
- The exception can be a string, integer, Boolean or an object.
- **Syntax:**

```
throw ( exception )
```

JS Throw Statement - Example

```
<html> <body>
<script type="text/javascript">
var x=prompt(“Enter a number
                between 0 and 10:”,“”);
try
{   if(x>10 || x<0)
        throw “Err1”;
    else if (isNaN(x))    // isNAN() -> is not a number
        throw “Err3”;
}
catch(er)
{  if ( er == “Err1” )
       alert("Error! Wrong val");
   if ( er == “Err3”)
       alert("Error! Not a number");
}
</script>
</body>
</html>
```

JS Objects

- JS is an Object Oriented Programming (OOP) language.
- An object has **properties** and **methods**.
  - **Properties** are the values associated with an object.
  - **Methods** are the actions that can be performed on objects.

Built-in JS objects

- JS String
- JS Date
- JS Array
- JS Boolean
- Js Math
- JS RegExp

Built-in JS objects: String Object

- **Properties**

  Example:

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

The output will be: **12**

- **Methods**

  Example:

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

The output will be: **HELLO WORLD!**

Built-in JS objects: Array Object

- **Create an Array object**

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

Note:
Arrays in javascript are not a sequential allocation of memory, but objects with enumerable property names and a few extra useful methods.

- var myCars = new Array ("Saab", "Volvo", "BMW"); **// condensed array**
- var myCars=["Saab", "Volvo", "BMW"]; **// literal array**

Built-in JS objects: Array Object

- **Access an Array:**

  document.write(myCars[0]);

- **Modify Values in an Array:**

  myCars[0]="Opel";

```
<html>
<body>
<script type="text/javascript">
var parents = [“Jani”, “Tove”];
var children = [“Cecilie”, “Lone”];
var family = parents.concat( children );
document.write( family );
</script>
</body>
</html>
```

Jani,Tove,Cecilie,Lone

Built-in JS objects: Date Object

- To instantiate a date:
  - new Date() // current date and time
  - new Date(milliseconds) //milliseconds since 1970/01/01
  - new Date(dateString)
  - new Date(year, month, day, hours, minutes, seconds, milliseconds)
- <u>Examples:</u>
  - *today = **new Date()***
  - *d1 = **new Date**("October 13, 1975 11:13:00")*
  - *d2 = **new Date**(79,5,24)*
  - *d3 = **new Date**(79,5,24,11,33,0)*

Built-in JS objects: Date Object

- To set a Date object to a specific date e.g. (14th January 2010):

```
var myDate1= new Date( );
myDate1.setFullYear(2010,0,14);
```

- To set a Date object to be 5 days into the future:

```
var myDate2=new Date();
myDate2.setDate( myDate.getDate( ) + 5);
```

- *To* Compare dates:

```
if ( myDate1 > myDate2 ) {…}
```

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

JS Create Your own Object

- **Create a direct instance of an object :**

```
employee = new Object();
```

- The following code creates an instance of an object and adds <u>four properties</u> to it:

```
employee.firstname = "John";
employee.lastname = "Doe";
employee.age = 50;
employee.salary = 4500;
```

- The following code adds a <u>method called promote()</u> to the employee object:

```
employee.promote = promote;
```

- We should define the function promote() somewhere in the code.

JS Create Your own Object

- Alternatively, we can create a template of an object :

The **template** defines the structure of an object:

```
function employee ( firstname, lastname, age, salary)
{
    this.firstname = firstname;
    this.lastname = lastname;
    this.age = age;
    this.salary = salary;
    this.promote = promote;   // a method
}
```

JS Create Your own Object

- Methods are just functions attached to objects.
- We have to write the promote() function used in the previous slide:

```
function promote( increment )
 {
      this.salary += increment;
 }
```

- **You can create new instances of the object, like this:**

```
firstEmployee  = new employee ("John", "Doe" , 50, 4500);
secondEmployee = new employee ("Sally","Rally", 48, 3600);
secondEmployee.promote(500);
```

JS Browser Detection

- The **Navigator object** contains information about the visitor's browser name, version, and more.

JS Browser Detection

```
<html> <body>
<script type="text/javascript">
        document.write("Browser CodeName: " + navigator.appCodeName);
        document.write("<br /><br />");
        document.write("Browser Name: " + navigator.appName);
        document.write("<br /><br />");
        document.write("Browser Version: " + navigator.appVersion);
        document.write("<br /><br />");
        document.write("Cookies Enabled: " + navigator.cookieEnabled);
        document.write("<br /><br />");
        document.write("Platform: " + navigator.platform);
        document.write("<br /><br />");
        document.write("User-agent header: " + navigator.userAgent);
</script>
</body></html>
```

Browser CodeName: Mozilla

Browser Name: Netscape

Browser Version: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36

Cookies Enabled: true

Platform: Win32

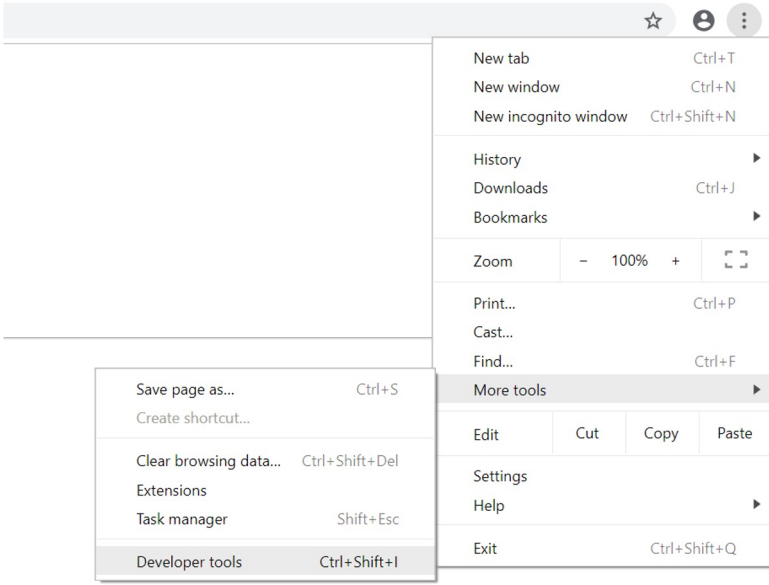
User-agent header: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36

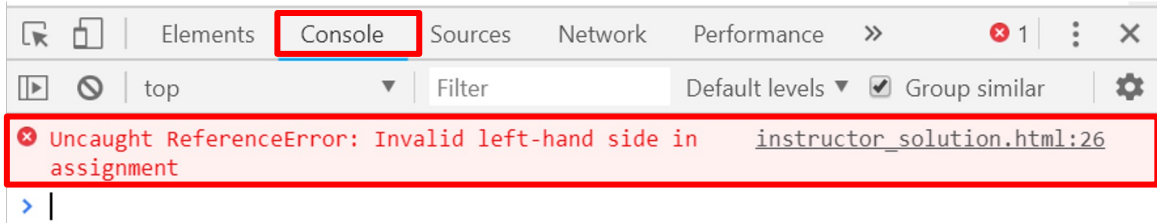
Javascript Debugging in Google Chrome

- From "**More Tools**", select "**Developer Tools**" OR press **Ctrl + Shift + I**
- From "**Console**", you can view if any errors occurs.
- If you click on the line number beside the error, you will automatically go to this line.
- Also if you write in Javascript:

console.log("hello");

This will appear in the console window (not the page)





Javascript Debugging in Google Chrome

- You open the source file to be able to debug step by step.
- Click on the "**Sources**", then double click the file that contains your script.
- You can add **breakpoint** by clicking on the **line number** as shown OR click on the line then press: **Ctrl + B**
- **Note:** if your code will be executed when a **button** is clicked, you have to click on the button to reach your breakpoint.
- When you reach the breakpoint you can execute the next line by pressing: **F10** normally.