MedTech – Mediterranean Institute of Technology
CS-Web and Mobile Development

Chp3- Client-Side JavaScript

DOM, Forms and HTML Manipulation

Dr. Lilia SFAXI
www.liliasfaxi.wix.com/liliasfaxi

Window object

Client-Side JavaScript

- Main entry point to all client-side JS features and APIs
- Represents a web browser window or frame
- Referred to with the identifier window
- It is the global object, and can be omitted
- Defines properties like:
  - location (URL currently displayed)
    - Ex: `window.location = "http://medtech.tn/"` navigates to the website of the institute
  - alert() : displays a message in a dialog box
  - setTimeout(): registers a function to be invoked after a specific amount of time
  - document: represents the content displayed in the window

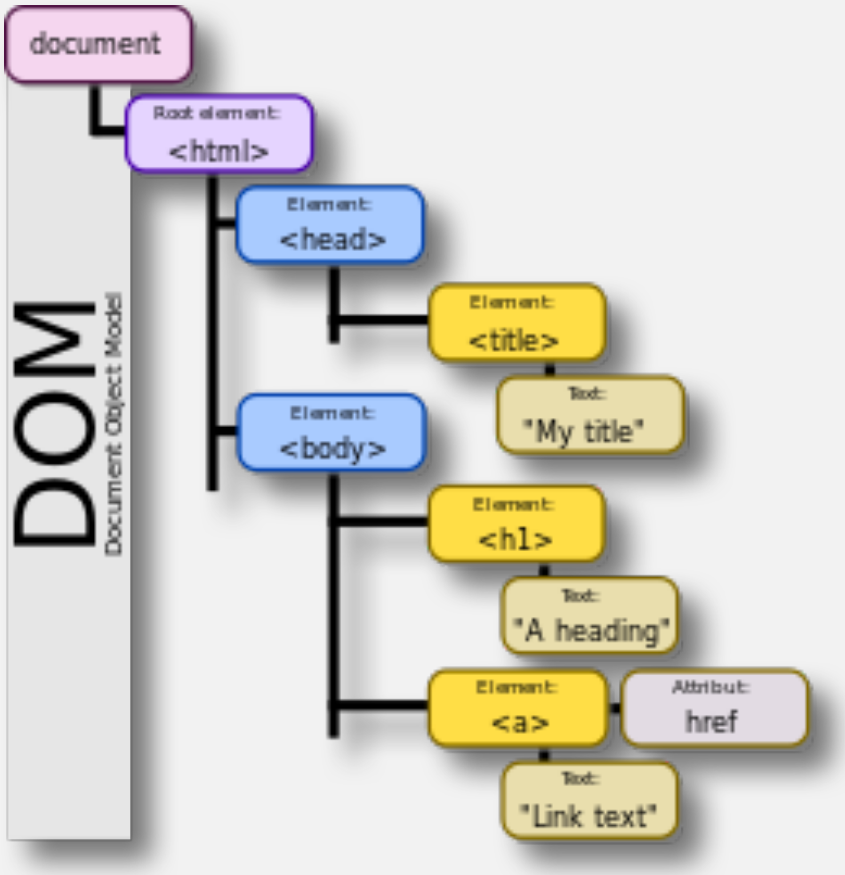
DOM: Document Object Model

Client-Side JavaScript

- W3C Standard
- Object-oriented model for representing HTML and XML documents
- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM: Document Object Model

Client-Side JavaScript




```
<html>
    <head>
        <title> My title</title>
    </head>

    <body>
        <h1> A heading </h1>
        <a href="link text">
            and a link
        </a>
    </body>
</html>
```

Client-Side JavaScript

GOING THROUGH THE DOM ELEMENTS

DOM Programming Interface

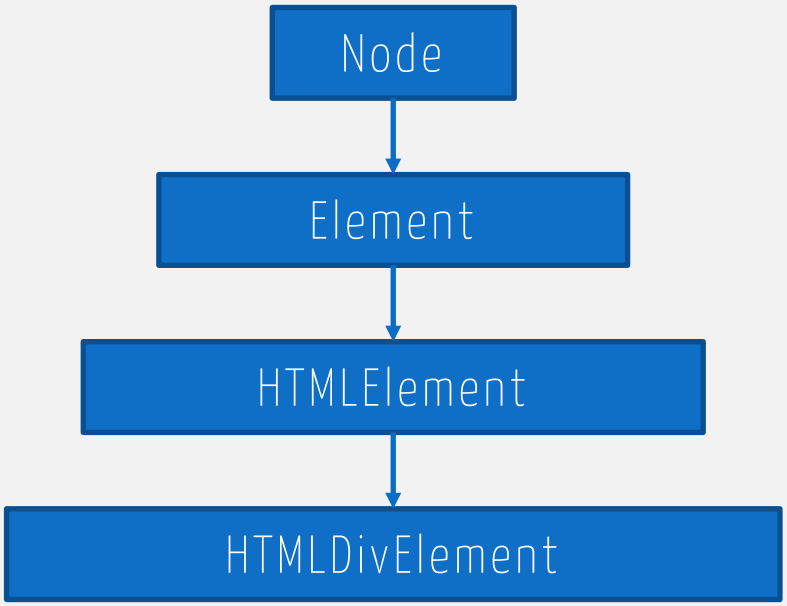
Client-Side JavaScript

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The DOM Programming Interface is the set of properties and methods of each object
  - A property is a value that you can get or set (like changing the content of an HTML element).
  - A method is an action you can do (like add or deleting an HTML element).
- The top of the DOM tree is the object *window*
- The object *document* is a child element of window
  - It represents the web page and enables to access the HTML element

DOM Types

Client-Side JavaScript

- Node: The most generic type, all DOM elements are nodes
- Element: HTML or XML Element
- HTML Element:
  - An Element in an HTML DOM is an HTML Element
  - Represents the HTML elements of the document: the tags
- All DOM elements are objects, thus having their own methods and attributes
  - Some methods and attributes are common to all DOM objects, as they are all of the same type : *Node*



Elements Handling

Client-Side JavaScript

- To access HTML elements using the DOM, several methods are offered by the object *document*
- getElementById(): accesses an element via its id
- getElementByTagName(): gets all the elements of the given tag as an array of objects
- getElementsByName(): returns an array of objects containing all the elements whose attribute *name* is equal to the given argument
- querySelector()
  - This function takes as a parameter a string representing a CSS selector
  - Returns the first element that corresponds to this selector
- querySelectorAll()
  - Returns an array of all the elements corresponding to the given selector

Elements Handling: getElementBy…

Client-Side JavaScript

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page </title>
</head>
<body>
<div id="myMenu">
    <div class="item">
        <span> Menu 1</span>
        <span> Menu 2</span>
    </div>
    <div class="Pub">
        <span> Pub 1</span>
        <span> Pub 2</span>
    </div>
</div>
<div id="content">
     <span>
     Here is my content
     </span>
</div>
A weird form to test
getElementsByName() :
<form name="myName">
    <input type="password"
name= "myName">
    <input>
    <button name="myName"></button>
</form>
<script src="test.js"></script>
</body>
</html>
```

```
var getByid=
     document.getElementById('myMenu');
var getElementByTagName =
     document.getElementsByTagName("div");
var getElementByName =
     document.getElementsByName("myName");
console.log("Result of getElementById : "  +
     getByid.textContent);
console.log("Result of getElementByTagName:");

for(var i=0;i<getElementByTagName.length;i++){
     console.log(getElementByTagName[i]);
}

console.log("Result of getElementByName : ");
for(var i=0;i<getElementByName.length;i++){
    console.log(getElementByName[i]);
}
```

Menu 1 Menu 2
Pub 1 Pub 2
Here is my content
A weird form to test getElementsByName() :

Elements Handling: querySelector

Client-Side JavaScript

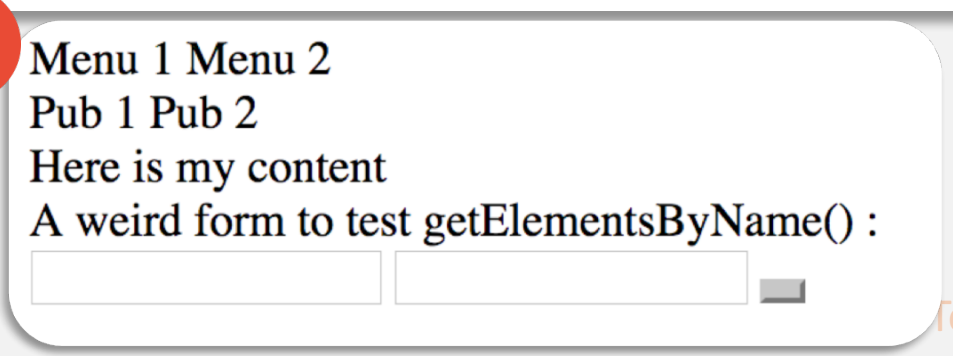
```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page </title>
</head>
<body>
<div id="myMenu">
    <div class="item">
        <span> Menu 1</span>
        <span> Menu 2</span>
    </div>
    <div class="Pub">
        <span> Pub 1</span>
        <span> Pub 2</span>
    </div>
</div>
<div id="content">
     <span>
     Here is my content
     </span>
</div>
A weird form to test
getElementsByName() :
<form name="myName">
    <input type="password"
name= "myName">
    <input>
    <button name="myName"></button>
</form>
<script src="test.js"></script>
</body>
</html>
```

```
var querySel=
     document.querySelector("head title");

var querySelAll=
     document.querySelectorAll("#myMenu .item
span");

console.log("Result of querySelector : " +
     querySel.textContent);

console.log("Result of querySelectorAll : ");
for(var i=0;i<querySelAll.length;i++){
     console.log(querySelAll[i].textContent);
}
```



Elements Handling: Attributes

Client-Side JavaScript

- To read, create or update an attribute of an element of the DOM, the object Element defines two methods:
  - getAttribute(): gets the value of an attribute
  - setAttribute(): updates the value of an attribute

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>
        My Test Page
    </title>
</head>
<body>
You will find my CV
<a id= "link"
href="www.myCv.tn">here</a>.
<script
src="test2.js"></script>
</body>
</html>
```

```
var link=document.getElementById("myLink");
var dest=link.getAttribute("href");
console.log("URL of the link before
    modification : "+ dest);
var newCVLink = prompt();
link.setAttribute("href", newCVLink);
dest = link.getAttribute("href");
console.log("URL of the link after
    modification : "+dest);
```

You will find my CV here.

Elements Handling: Attributes

Client-Side JavaScript

- Another way of accessing an attribute is to use *.attrName*
  - Not supported by all the browsers

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>
        My Test Page
    </title>
</head>
<body>
You will find my CV
<a id= "myLink"
href="www.myCv.tn">here</a>.
<script
src="test2.js"></script>
</body>
</html>
```

```
var anotherLink=
    document.getElementById("myLink");

alert("old link: "+  anotherLink.href);

anotherLink.href="newLien.tn"

alert("new link: "+  anotherLink.href);
```

You will find my CV here.

Elements Handling: innerHTML

Client-Side JavaScript

- innerHTML
  - Gets the child HTML code of a tag
  - Adds directly HTML content
- textContent (or innerText, this latter not supported in all browsers)
  - Gets only the text, without the HTML tags

```
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page</title>
</head>

<body>
<div id="div">
    <p> I am in a div with id
div</p>
</div>
<div id="div1">
    <p> I am in a div with id
div 1</p>
</div>
<script src="test3.js"></script>
</body>

</html>
```

I am in a div with id div

I am in a div with id div 1

```
var myDiv=document.getElementById("div");
var myDiv1=document.getElementById("div1");
function innerTest(){
     myDiv1.textContent="I delete everything in my
          road <b>i'm a monster 3:)</b> ";
     myDiv.innerHTML="I delete everything in my road
          <b>i'm a monster too 3:)</b> ";
}
```

Elements Handling: querySelector

Client-Side JavaScript

- querySelector() returns the first element in the document that matches a specified *CSS selector(s)* in the document.
  - To return all the matches, use the querySelectorAll() method instead.

```
function querySelTest() {
      var elem = document.querySelector("p");
      console.log(elem);
      var elems = document.querySelectorAll("p");
      console.log(elems);
}
```

Elements Handling: Nodes

Client-Side JavaScript

- There are some useful attributes that help extracting DOM elements
- **nodeType**: gets the type of DOM node, in the form of a number such as:
  - 1: Element
  - 2: Attribute
  - 3: Text
- **nodeName:** gets the name of the element node in Uppercase

Node Type	Name of the constant	Value
Element	Node.ELEMENT_NODE	1
Attribute	Node.ATTRIBUTE_NODE	2
Text Node	Node.TEXT_NODE	3
	Node.CDATA_SECTION_NODE	4
	Node.ENTITY_REFERENCE_NODE	5
	Node.ENTITYT_NODE	6
Instruction	Node.PROCESSING_INSTRUCTION_NODE	7
Comment	Node.COMMENT_NODE	8
	Node.DOCUMENT_NODE	9
	Node.TYPE_NODE	10
XML Fragment	Node.DOCUMENT_FRAGMENT_NODE	11
	Node.NOTATION_NODE	12

Elements Handling: Nodes

Client-Side JavaScript

- Other attributes to go through the DOM
  - parentNode: enables to access the parent node of an element
  - firstChild : enables to access the first child of a node
  - lastChild: enables to access the last child of a node
  - firstElementChild : enables to access the first element child of a node
  - lastElementChild : enables to access the last element child of a node
  - childNodes: returns an array containing all the child nodes
  - nextSibling: accesses the next node, sibling of the current
  - previousSibling: accesses the previous node

Elements Handling: Nodes

Client-Side JavaScript

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page </title>
</head>
<body>

<p id="para"> You will find my CV
<a id="link"
href="www.myCv.tn">here</a>. Please
call me on <i>222222</i> for
further details</p>

<div id="myMenu">

    <div id="item">
        <span>menu 1</span>
        <span>menu 2</span>
    </div>

    <div class="pub">
        <span>Pub 1</span>
        <span>Pub 2</span>
    </div>

</div>
<script src="test4.js">
</script>
</body>
</html>
```

```
var nod = document.getElementById("para");
console.log("Hi I am the node "+nod.nodeName) ;

console.log("Difference between firstChild and
firstElementChild ");

console.log("Hi, I am the first child : "
    +nod.firstChild);

console.log("Hi, I am the first Elementchild : "
    +nod.firstElementChild);

console.log(" Next Sibling:  "+nod.nextSibling);
console.log(" Next Element Sibling:  "
    +nod.nextElementSibling);
```

You will find my CV here. Please call me on *222222* for further details

menu 1 menu 2
Pub 1 Pub 2

Elements Handling: Nodes

Client-Side JavaScript

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page </title>
</head>
<body>

<p id="para"> You will find my CV
<a id="link"
href="www.myCv.tn">here</a>. Please
call me on <i>222222</i> for
further details</p>

<div id="myMenu">

    <div id="item">
        <span>menu 1</span>
        <span>menu 2</span>
    </div>

    <div class="pub">
        <span>Pub 1</span>
        <span>Pub 2</span>
    </div>

</div>
<script src="test5.js">
</script>
</body>
</html>
```

```
var nod=document.getElementById("para");
console.log("Hi, I am the node: "+nod.nodeName) ;
console.log("These are my children : ");
var children=nod.childNodes;
for (var i=0;i<children.length;i++){
    console.log("Hi, I am child number:"+ (i+1)+"
my name is: "
+children[i].nodeName+" and my content text
is:");
    if(children[i].nodeType==Node.ELEMENT_NODE){
        console.log(children[i].firstChild.data);
    }else{
        console.log(children[i].data);
    }
}
```

You will find my CV here. Please call me on *222222* for further details

menu 1 menu 2
Pub 1 Pub 2

Elements Handling: Adding Nodes

Client-Side JavaScript

- In order to add a new node in the DOM, follow the next steps:
  - Create the element to add, using the method createElement() that takes a String as a parameter, representing the name of the element to create.
  - Add the necessary attributes to the element using one of the following methods:
    - setAttribute(*attr_name, value*)
    - <obj>.<attribute> = *value*
  - Insert the element in the DOM using the method appendChild() that takes as a parameter the element to insert, and adds this element as the last child of the object calling the method.
- To add a text node, use the method createTextNode() with the corresponding text as a parameter
- The method insertBefore*(n , s)* inserts the node *n* before the node *s*

Elements Handling: Adding Nodes

Client-Side JavaScript

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title> My Test Page </title>
</head>
<body>

<p id="para"> You will find my CV
<a id="link"
href="www.myCv.tn">here</a>. </p>
<div id="myImage">
     <button onclick="addImage()">
          Add Image
     </button>
</div>
<div id="myMenu">

    <div id="item">
        <span>menu 1</span>
        <span>menu 2</span>
    </div>

    <div class="pub">
        <span>Pub 1</span>
        <span>Pub 2</span>
    </div>

</div>
<script src="test6.js">
</script>
</body>
</html>
```

```
var nod = document.getElementById("myImage");
function addImage() {
     var newNode = document.createElement("img");
     newNode.id = "jsimg";
     newNode.src = "js.jpg";
     newNode.alt = "Javascript…";
     newNode.width = "200";
     newNode.height = "200";
     nod.appendChild(newNode);
}
```

You will find my CV here.

Add Image

menu 1 menu 2

Pub 1 Pub 2

Elements Handling: Cloning Nodes

Client-Side JavaScript

- In order to clone a node in the DOM, call the method cloneNode()
- It takes a boolean as a parameter, which value is:
  - True: if the node will be cloned with its children and attributes
  - False: if the node will be cloned without them

```
function clone() {
    var trueClonedNode = nod.cloneNode(true);
    var falseClonedNode = nod.cloneNode(false);
    cloneDiv.appendChild(trueClonedNode);
    cloneDiv.appendChild(falseClonedNode);
}
```

Elements Handling: Deleting Nodes

Client-Side JavaScript

- In order to delete a node from the DOM, call the method removeChild() from the father node
  - This method takes as a parameter the node to delete
  - The return value is a reference of the deleted node

```
function deleteMe() {
    var elem = document.getElementById("item");
    elem.removeChild(elem.firstElementChild);
}
```

Elements Handling: Replacing Nodes

Client-Side JavaScript

- In order to replace a node in the DOM, call replaceChild() from the father node.
  - This method takes as a parameter the new node, followed by the old node

```
function replaceMe() {
    var nod = document.getElementById("item");
    var newnod = document.
        getElementsByClassName("pub")[0].
        cloneNode(true);
    nod.replaceChild(newnod, nod.lastElementChild);
}
```

Events Handling
Client-Side JavaScript

- HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.
- Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).
- Every DOM element has its own addEventListener method, which allows you to listen specifically on that element.

```
<button>Click me</button>
<p>No handler here.</p>
<script>
        var button = document.querySelector("button");
        button.addEventListener("click", function() {
                alert("Button clicked.");
        });
</script>
```

Events Handling
Client-Side JavaScript

- The removeEventListener method, called with arguments similar to as addEventListener, removes a handler.

```
<button>Act-once button</button>
<script>
        var button = document.querySelector("button");
        function once() {
                console.log("Done.");
                button.removeEventListener("click", once);
        }
        button.addEventListener("click", once);
</script>
```

Events Handling: Event Objects

Client-Side JavaScript

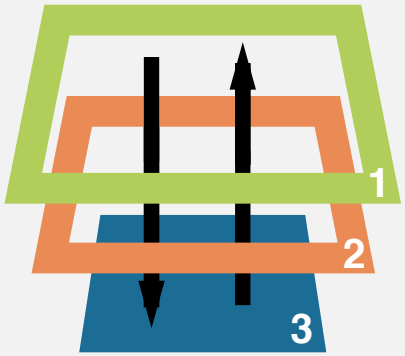
- The event object, passed as a parameter to addEventListener, gives additional information about the event

```
<button>Click me any way you want</button>
<script>
	var button = document.querySelector("button");
	button.addEventListener("mousedown", function(event) {
		if (event.which == 1)
			console.log("Left button");
		else if (event.which == 2)
			console.log("Middle button");
		else if (event.which == 3)
			console.log("Right button");
	});
</script>
```

Events Handling: Event Propagation

Client-Side JavaScript

- DOM elements can be nested inside each other. And somehow, the handler of the parent works even if you click on it's child.
  - If a button inside a paragraph is clicked, event handlers on the paragraph will also receive the click event.
- Event Propagation:
  - In all browsers, except IE <9, there are two stages of event propagation:
    - The event first goes down, from the topmost element to the innermost element: called Capturing (1->2->3)
    - It then bubbles up, to trigger on parents in nesting order: called Bubbling (3->2->1)



Events Handling: Event Propagation

Client-Side JavaScript

- All methods of event handling ignore the capturing phase.
  - By default, only the bubbling takes place
  - Using addEventListener with last argument true is only the way to catch the event at capturing.
- The event object has a target property refering to the node where they originated

```
<div class="d1"> 1
        <div class="d1"> 2
                <div class="d1"> 3 </div>
</div> </div>
<script>
        var divs= document. getElementsByTagName("div");
        for (var i = 0; i < divs.length; i++) {
                divs[i].addEventListener('click', function(event) {
                        alert("target: " + event.target.className + " - this: " + this.className)
                });
        }
</script
```

Events Handling: Event Propagation

Client-Side JavaScript

- You can stop the event propagation by running event.stopPropagation()
  - Or event.cancelBubble = true for IE <9

```
<p>A paragraph with a <button>button</button>.</p>
<script>
	var para = document.querySelector("p");
	var button = document.querySelector("button");
	para.addEventListener("mousedown", function() {
		console.log("Handler for paragraph.");
	});
	button.addEventListener("mousedown", function(event) {
		console.log("Handler for button.");
		if (event.which == 3) event.stopPropagation();
	});
</script>
```

Events Handling: Events

Client-Side JavaScript

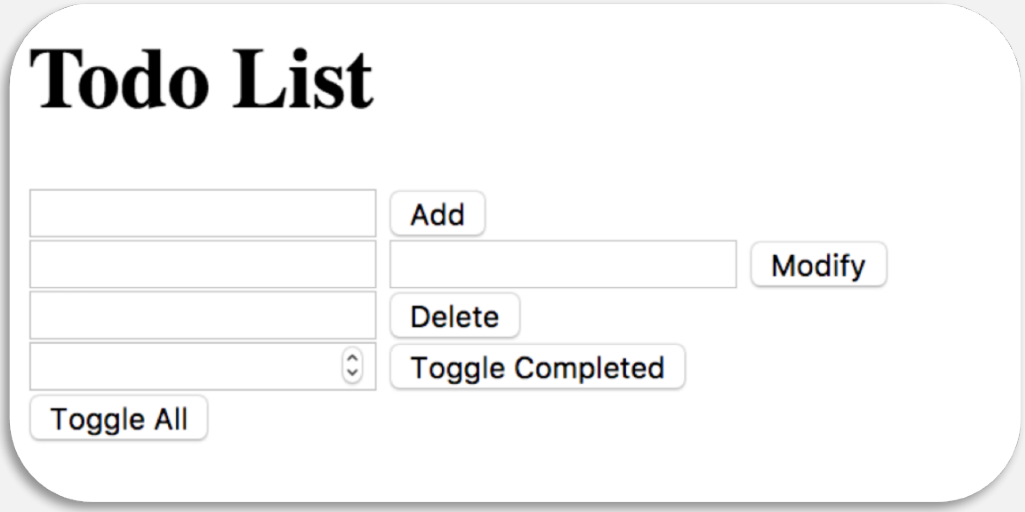
- Javascript defines various DOM events, such as:
  - onmouseover: when the mouse hovers over the element
  - onblur: when the element loses focus
  - onfocus: when the element gains focus
  - onselect: when the element is selected
  - onchange: when the content of the element is changed
  - onsubmit: when a form is submitted
  - onload: when an element ends its loading phase
  - …

JavaScript/ECMAScript

RUNNING EXAMPLE

Todo App
Running Example

- Go back to the TODO app you created the last time: we are going to add an interface to it!
- Follow these steps:
- Step1: Interface
  - Create an interface that looks like the following



Todo App
Running Example

- For now, all the display will be on the console
- Step 2:
  - Add a listener to the button add, that adds the written element to the todolist
  - Add a listener for the modify button, that modifies the element which id is given in the first input with the task in the second
  - Add a listener for the delete button, that deletes the task which id is given in the input
  - Add a listener to the toggle completed button, that does that to the task which id is given in the input
  - Add a listener to the ToggleAll button
- Step 3:
  - Now, the tasks will be displayed in an unordered list under the form. Every modification of the list is immediately impacted on the display

References

- M. Haverbeke, *Eloquent JavaScript: A modern introduction to programming*, 2014

- Textbook
  - D. Flanagan, *JavaScript: The Definitive Guide*, 6th edition, O'reilly, 2011