Web technology – JavaScript

functions

dotNET academy

Document Object Model

When you open a web page in your browser, the browser is shown the HTML, and it starts to 'parse'. The result is the construction of a model of the structure of the page. And the browser uses this model to draw the page on the screen.

Document Object Model

Browsers make the DOM available as an API with which you as a programmer can interact with

- View page content
- Interact with elements
- To customize elements within a page

In other words: DOM is the internal representation of the browser

https://dom.spec.whatwg.org/ = dom working group (invented by Apple, Mozilla, Opra)

Dom & JavaScript

With JavaScript, we can interact with the DOM to:

- Page structures to inspect
- CSS styling edit
- Apply event listeners
- Edit nodes within a page
- Node attributes to change

Document & Window & DOM ready event

Document & Window are the 2 objects that the DOM API gives you the most you will participate with.

You have to take into account that you can only start editing the DOM when it has been created. You can do this by attaching an event listener to the document object for the DOMContentLoaded event:

Document.addEventListener()

```
document.addEventListener("DOMContentLoaded",event=>{
    console.log("DOM ready");
})
```

Introductie

- Sequential code execution:
    - Code 1 – Code B – Code C - ...
- Now you have to respond to "keydown" of a key, mouse click, touch screen, ... :

```
<form>

    Naam: <input type="text">
    <input type="button" onclick="DoeIets()">
    <input type="reset">

</form>
```

Important events

Event	Omschrijving
Mouseover	Muisaanwijzer boven element
Mouseout	Muisaanwijzer verlaat element
Mousedown	Muisaanwijzer wordt ingedrukt
Mouseup	Muisknop wordt losgelaten
Dblclick	Dubbelklik
Mousemove	Muisaanwijzer beweegt
Click	muisklik
Keydown	Toets wordt ingedrukt
Keyup	Toets wordt losgelaten
Submit	Formulier wordt verstuurd
Reset	Formulier wordt gereset
Focus	Een element krijgt de focus
Blur	Een element verliest de focus
Load	Een element is geladen
Unload	Een element is niet meer geladen
Resize	De afmetingen van een element worden veranderd

```
9   <script>
10
11      function DoeIets(){
12
13          alert("Doe Iets");
14      }
15
16  </script>
17      </head>
18
19      <body>
20
21          <form onsubmit="DoeIets()" onreset="DoeIets()">
22
23              <input type="button" onclick="DoeIets()">
24              <input type="text" onfocus="DoeIets()">
25              <input type="submit">
26              <input type="reset">
27
28          </form>
```

Inline event-handling

Event can be handled with an event handler named the event and a prefix "on"

For example: onmouseover, onload, ...

The event handler is placed as an attribute in the HTML tag

Disadvantage: The event handlers are embedded, so you have a mixture of HTML and JavaScript code (difficult to maintain with larger applications)

```
<a href="#" onmouseover="DoeIets()">Doe Iets</a>
```

Register functions as *"handlers"* for specific events.

```
<script>

    document.addEventListener("click",function(arg){
        console.log(arg);
    })

    document.addEventListener("dblclick",(arg)=>{
        console.log("dblclick");
        console.log(arg);
    });

</script>


</body>
```

Remove event handlers

```
<script>

    function DoeIets(arg){
        console.log(arg);
    }
   document.addEventListener("click",DoeIets)

   document.addEventListener("dblclick",(arg)=>{
       console.log("dblclick");
       console.log(arg);

       document.removeEventListener("click",DoeIets);



   });

</script>
```

Event Object

Each event handler function bevta an argument = Event Object

This item contains information about the event

```
document.addEventListener("mouseover",function(arg){

 console.log(arg);

});
```

getElementById

Returns a reference to element by an ID. The ID is a string that is given to identify the element

```
    function test(){
        let element = document.getElementById("myText")
        console.log(element)
        if(element.value==""){
            element.value = "Gelieve tekst veld in te vullen"
        }
    }

</script>

<input type="text" id="myText" value="">
<button onclick="test()">click me</button>
```

Methods to find elements

- getElementsByTagName

```
var header = document.getElementsByTagName('h1')[0];
```

- getElementsByClassName

```
var linkHeader = document.getElementsByClassName('linkheader')[0];
```

- getElementById

```
var kar = document.getElementById("WinkelKarTable");
```

- querySelector (op basis van css identificaties)

```
document.querySelector('#BannerCheckbox')
```

Window object

The window object represents the window that contains the DOM document.

window.document points to the document object loaded into window (you don't need to encrypt window.document, but you can call document directly).

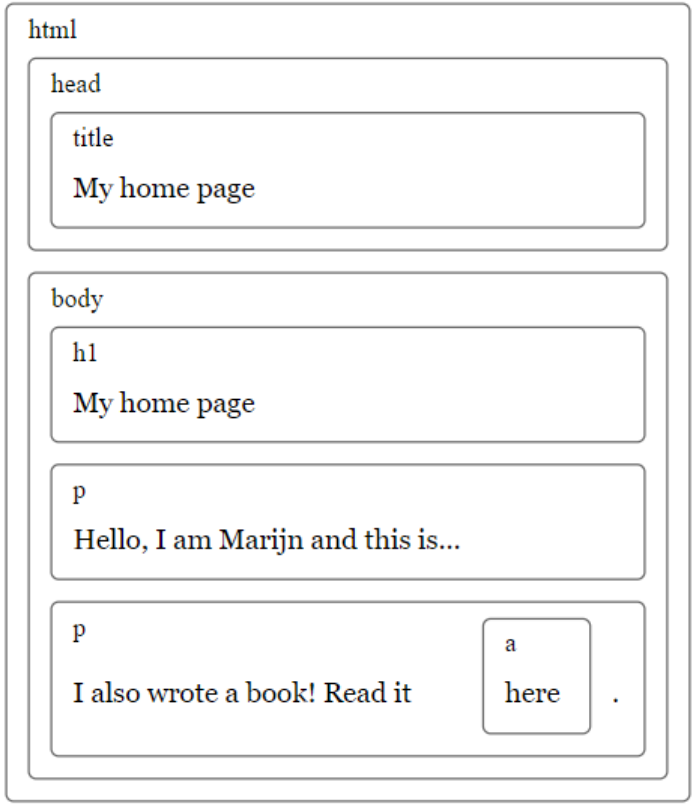
Important properties of the window object :

- Console : om te loggen / debuggen: vb. console.log(“test”)
- Document
- History
- Location: https://developer.mozilla.org/en-US/docs/Web/API/Location
- localStorage
- SessionStorage

Document object

The document object represents the DOM tree

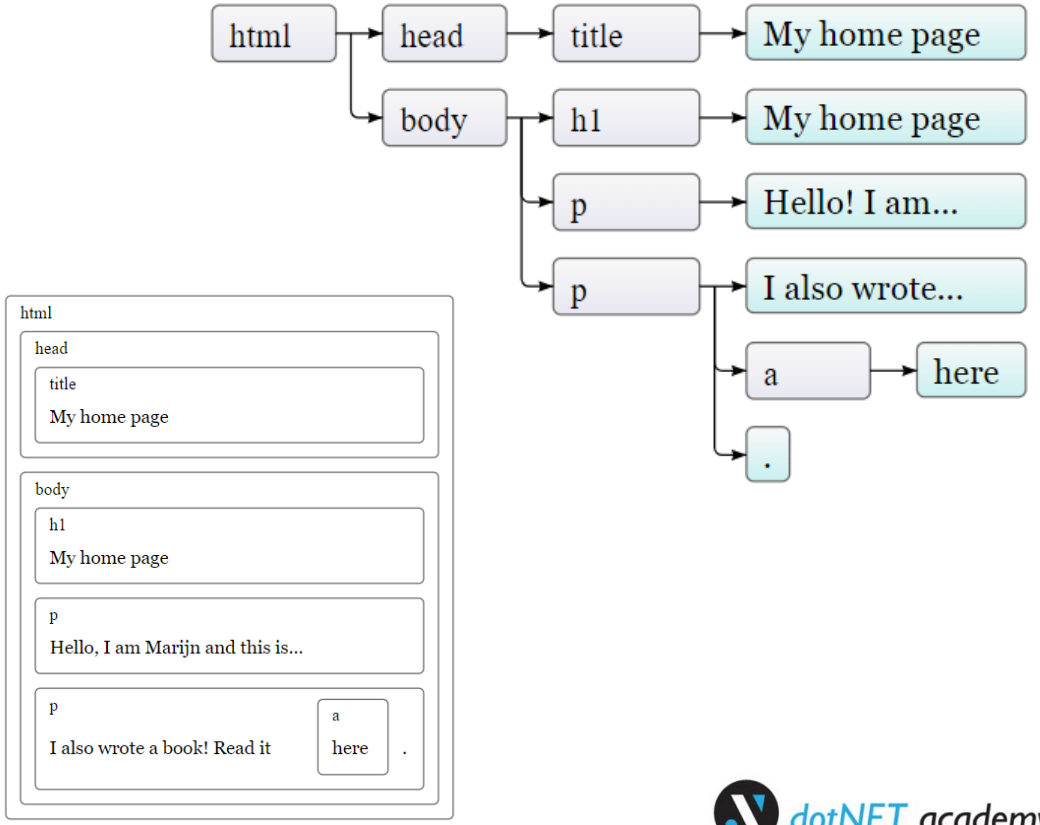
```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```



Document object

For each box, there is an object to "interact" with.

There is a global variable that gives us access to these objects, namely "document".



Document object

Normally you would have to encode window.document to load something from the DOM tree, but since the window object is a global object, you can just use document.

For example: document.title, or document. URL.

Document Object

- From the document object you can use the body and head element nodes: https://developer.mozilla.org/en-US/docs/Web/API/Element
- E.g. document.documentElement (the whole document node)

```
document.documentElement
<html prefix="og: http://ogp.me/ns#" lang="en" class="gr__
▶<head>…</head>
▶<body data-gr-c-s-loaded="true">…</body>
</html>
```

- document.body : the body:

```
document.body
▶<body data-gr-c-s-loaded="true">…</body>
```

- document.head : the head

```
document.head
▶<head>…</head>
```

Collecties van element nodes

You can request a list of elements by

Document.links : lists all links

Document.images : lists all images

Document.forms: lists all forms

Selectors

Selectors

- `document.getElementById()` accepts an `id` attribute value and returns the element (only one, as `id` cannot be repeated in the page according to the HTML standard)
- `document.getElementsByTagName()` accepts a `tag` element name and returns all the elements that match it
- `document.getElementsByClassName()` accepts a `class` attribute name and returns all the elements that match it
- `document.querySelector()`
- `document.querySelectorAll()`

Node Types

There are different types of nodes

- Document: the document node, or the root of the tree
- Element : is an HTML tag
- Attr: an attribute of a tag
- Text: the text content of an element or of an attr node
- Comment: an HTML comment

Moving through the DOM

The DOM is a tree of different elements, where the document node is the root and thus points to the HTML element node, and it points to its child elements head and body, and so you move further through the tree trick.

Good to know is that each element has 1 parent:
Node.parentNode (most commonly used) or Node.parentElement

To see if a node has children, we use node.hasChildnodes() that returns a bool value.

And then node.childNodes can get the elements.

To have the first child element you can use node.firstElementChild and for the latter the node.lastElementChild

Editeren van de DOM tree

There are several methods to edit the DOM:

`document.createElement()`: creates a new Element Node
`document.createTextNode()`: creates a new Text Node

You can add new elements to others:

`document.appendChild()`

```
document.addEventListener("DOMContentLoaded", event => {

    const div = document.createElement('div')
    div.appendChild(document.createTextNode('Hello world!'))
    document.body.appendChild(div);

    // Append an item in a list
    var node = document.createElement("li"); // Create a <li> node
    var textnode = document.createTextNode("Water"); // Create a text node
    node.appendChild(textnode); // Append the text to <li>
    document.getElementById("myList").appendChild(node); // Append <li> to <ul> with id="myList"
});

</script>

<ul id="myList">

</ul>
```

`first.removeChild(second)` removes the child node “second” from the node “first”.

`document.insertBefore(newNode, existingNode)` inserts “newNode” as a sibling of “existingNode”, placing it before that in the DOM tree structure.

`element.appendChild(newChild)` alters the tree under “element”, adding a new child Node “newChild” to it, after all the other children.

`element.prepend(newChild)` alters the tree under “element”, adding a new child Node “newChild” to it, before other child elements. You can pass one or more child Nodes, or even a string which will be interpreted as a Text node.

`element.replaceChild(newChild, existingChild)` alters the tree under “element”, replacing “existingChild” with a new Node “newChild”.

`element.insertAdjacentElement(position, newElement)` inserts “newElement” in the DOM, positioned relatively to “element” depending on “position” parameter value. See the possible values.

`element.textContent = 'something'` changes the content of a Text node to “something”.

removeChild

Removing the parent's first child

```
<script>
document.addEventListener("DOMContentLoaded", event => {

    const node = document.querySelectorAll("li")
    console.log(node)

    const ul = document.getElementsByTagName("ul")[0]
    console.log(ul);

    ul.removeChild(node[0]);

});
</script>
    <ul>
        <li>een</li>
        <li>twee</li>
        <li>drie</li>
        <li>vier</li>
        <li>vijf</li>
    </ul>
```

insertBefore

```
document.addEventListener("DOMContentLoaded", event => {


    const newItem = document.createElement("li");
    const textnode = document.createTextNode("nul");
    newItem.appendChild(textnode);
    const list = document.getElementsByTagName("ul")[0]
    list.insertBefore(newItem, list.childNodes[2]);

});
</script>
    <ul>
        <li>een</li>
        <li>twee</li>
        <li>drie</li>
        <li>vier</li>
        <li>vijf</li>
    </ul>
```

Add a new classname

```
function start(){
    document.getElementsByTagName("a")[0].addEventListener("click",function(){

          document.querySelector("img").className += " doAnim";
        const el = document.querySelector("div");
        el.classList.add("doAnim");

        el.addEventListener("webkitAnimationStart", function(event){
            console.log("start");
        });

        el.addEventListener("webkitAnimationEnd", function(event){
           this.classList.remove("doAnim");
        });



    });
```

innerHTML

The innerHTML property converts or returns the HTML content (= inner HTML) of an element

```
<script>
    function ChangeText() {
        document.getElementById("ptag").innerHTML = "verandering tekst";
    }
</script>
</head>

<body>

    <p id="ptag">
        Dit is tekst
    </p>

    <button onclick="ChangeText()">
        Change Text
    </button>
```

this

By using this we can write a function that can be used for multiple elements.

We now check within the function which element the function has called.

This refers to the owner of the function that has it performed

```
<script>

    function DoeIets(obj){

        console.log(obj.id);
        var id = obj.id;
        var element = document.getElementById(id);
        element.innerHTML = "ok";
    }

</script>
    </head>

    <body>

        <div id="d1" onclick="DoeIets(this)">1
        </div>

        <div id="d2" onclick="DoeIets(this)">2
        </div>
    </body>
```

Praktisch voorbeeld