**Day 5**

# DOM (Document Object Model)



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

* JavaScript can change all the HTML elements in the page
* JavaScript can change all the HTML attributes in the page
* JavaScript can change all the CSS styles in the page
* JavaScript can remove existing HTML elements and attributes
* JavaScript can add new HTML elements and attributes
* JavaScript can react to all existing HTML events in the page
* JavaScript can create new HTML events in the page

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 DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the 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).

**<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = "Hello World!";  
</script>  
  
</body>  
</html>**

## The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

**The innerHTML Property**

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

## Finding HTML Elements

**Method Description**

document.getElementById(id) Find an element by element id

document.getElementsByTagName(name) Find elements by tag name

document.getElementsByClassName(name) Find elements by class name

**Changing HTML Elements**

**Property Description**

element.innerHTML = new html content Change the inner HTML of an element

element.attribute = new value Change the attribute value of an HTML element

element.style.property = new style Change the style of an HTML element

Method Description

element.setAttribute(attribute, value) Change the attribute value of an HTML element

**Adding and Deleting Elements**

**Method Description**

document.createElement(element) Create an HTML element

document.removeChild(element) Remove an HTML element

document.appendChild(element) Add an HTML element

document.replaceChild(new, old) Replace an HTML element

document.write(text) Write into the HTML output stream

**Adding Events Handlers**

**Method Description**

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

**Finding HTML Objects**

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

**Later, in HTML DOM Level 3, more objects, collections, and properties were added.**

**Property Description**

document.anchors Returns all <a> elements that have a name attribute

document.applets Returns all <applet> elements (Deprecated in HTML5)

document.baseURI Returns the absolute base URI of the document

document.body Returns the <body> element

document.cookie Returns the document's cookie

document.doctype Returns the document's doctype

document.documentElement Returns the <html> element

document.documentMode Returns the mode used by the browser

document.documentURI Returns the URI of the document

document.domain Returns the domain name of the document server

document.forms Returns all <form> elements

document.head Returns the <head> element

document.images Returns all <img> elements

document.implementation Returns the DOM implementation

document.inputEncoding Returns the document's encoding (character set)

document.lastModified Returns the date and time the document was updated

document.links Returns all <area> and <a> elements that have a href attribute

document.readyState Returns the (loading) status of the document

document.referrer Returns the URI of the referrer (the linking document)

document.scripts Returns all <script> elements

document.strictErrorChecking Returns if error checking is enforced

document.title Returns the <title> element

document.URL Returns the complete URL of the document

* A form is simply an area that contains form fields. A form field is an object which the user can add information to whether it be a text area, a radio button or a dropdown menu.
* Any web form essentially works the same way – the user fills in some information and submits a button.
* The form is essential if you want to know anything about your user

e.g. have them send you a mail, store information about them in your database, and so forth.

**How does an HTML form work?**

A web form consists two parts - the HTML ‘front end’ and a back-end form processor. The HTML front end part handles the presentation while the back end handles the form submissions.   
The back end of the form is usually written in languages like PHP or ASP. But don’t worry you don’t have to know any of these languages to get a working form – you can use a formmail, which is usually accessible through your internet service provider (ISP). If not, there is several free and paid services out there and all you must do is choose one of your liking.

Here is how a form normally works, step-by-step

* A user visits your web page which contains a form.
* The users’ web browser displays the HTML form.
* The user fills out the form and clicks submits
* The browser sends the submitted form data to the web server
* A form processor script (also known as a formmail) running on the web server processes the form data
* A response page is sent back to the browser.

# Creating Your First Form

The <form> tag is used to create the actual form – it looks like this:

**<form>**

**Input elements**

**</form>**

The <form> tag contains the content of your form. This content is called form widgets, controls or fields – they all describe the same thing. This content is mostly different kinds (or states, which is actually the correct word to use) of input fields and a button here and there. Let’s have a look at a very simple form with just two input fields and a submit-button:

**<form>**

**First name: <input type="text" name="firstname" /> <br />**

**Surname: <input type="text" name="surname" /> <br />**

**<input type="submit" value="Submit now" />**

**</form>**

# Grouping with the fieldset tag

If you have a longer form, it can be very handy to group the different kinds of fields. To do this, you use the <fieldset> element. The name of the group is given by the first <legend> element, which is a child of the <fieldset> element. Once we are at it I want to introduce yet another element, the <label>. The label is used to add a caption to each piece of element you are collecting.

**<form>**

**<fieldset>**

**<legend>Contact Details</legend>**

**<label for="name">Name:</label>**

**<input id="name"><br />**

**<label for="telephone">Telephone:</label>**

**<input id="telephone"><br />**

**<label for="user-email">Email:</label>**

**<input id="user-email">**

**</fieldset>**

**<fieldset>**

**<legend>User info</legend>**

**<label for="username">Username:</label>**

**<input id="username"><br />**

**<label for="password">Password:</label>**

**<input id="password"><br />**

**</fieldset>**

**<input type="submit" value="Submit now" />**

**</form>**

Why use the fieldset? Imagine you wanted to add something to this form, perhaps favorite foods. Then you would group the different kind of foods in a new fieldset and gives clarity to your form. I have also added the attribute id to input field. This defines this elements unique identifier and we use it when we are assigning labels. Using the for-attribute we connect the input field and the label.

**Sending the Data**

Until now we have not discussed how to send the data entered by the user. You can either send it to your email address, to another page, or to your database.

Whichever option you choose, you always must define how the browser handles the user input. This is defined through the method attribute, which can have one of two values – **post** or **get**.

The easiest way to handle your data is to send it to your email. All you have to do is add the action attribute to the form element and the method attribute. The action attribute tells the browser what to do with the content of your form and the method attribute tells the browser how to handle it. Here is an example:

**<form action="mailto:your@email.com" method="post">**

**First name: <input type="text" name="firstname" />**

**Surname: <input type="text" name="surname" />**

**<input type="submit" value="Submit now" />**

**</form>**

# Form validation using HTML5

The fields in a form are there to gather information from your visitors, but no matter how much time you spend setting up your form there is going to be at least some of your visitors who gets confused filling out your form, might skip important parts of your form, type invalid values (who has ever heard of an email address with a space in it?) and the result? They click submit and you end up getting a whole lot of useless data.

And this is where validation enters – if you validate your inputfields before the data is sent to your inbox or database, then the visitor gets a chance to correct the data and this means that the chance of you getting the data you asked for, improves.

For years this have been done with JavaScript and done correctly, this works to perfection. So why change that you might ask? Because the guys behind HTML5 have introduced new, simpler ways of validating your data, that does not require several lines of JavaScript, but let the browser do the validation client-side. And the best part about it is that it is very easy to implement - all you need to do is add the right attribute here and there.

# How HTML5 validation works

The basic idea behind HTML5 validation is, that you tell the browser which fields you want validated but don’t actually do the tedious implementation yourself. As you define what state your input field is in you also asks the browser to validate the field client-side based on the type of input field.

Unfortunately, not all modern browsers understand this new way of validating forms but don’t worry. If you are using a type value the browser doesn’t recognize, it will just treat is as a regular text field and no harm has been done. That means that all the following input elements get treated the same:

**<form>**

**<input type="text" /> <br />**

**<input type="my-weird-value" /> <br />**

**<input /> <br />**

**<input type="submit" value="Submit now" />**

**</form>**

And why is this useful information when we are talking about validation? Because some of the input types discussed will not be supported by all browsers but this should not hold you from implementing the new HTML5 validation tricks.

# the required attribute

Let’s say you have a regular text input field that cannot be left blank – then you use the attribute required. HTML5 allows you to shorten you markup when using the required attribute, so instead of writing it the long way:

**<form>**

**<input type="text" required /><br />**

**<input type="submit" value="Submit now" />**

**</form>**

Using this required attribute the browser informs the user that she has to fill out this particular field before submitting the form. The required attribute do not take into consideration what kinds of data are typed into the input fields but you do have the opportunity to do this with some of the following input types. Now let's go have a look at them in the following chapters!

# Validating email addresses

**<form>**

**<input type="email" required /> <br />**

**<input type="submit" value="Submit Now!">**

**</form>**

# Validating email addresses using pattern

**<form>**

**<input pattern="/^[a-zA-Z0-9.!#$%&'\*+/=?^\_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)\*$/" required />**

**<br />**

**<input type="submit" value="Submit Now!">**

**</form>**

# Type="email" or pattern?

As both ways of validating email addresses has their pros and cons it is up to you to decide which one to use. You should not try to use them both at the same time as this might induce a clash in browsers that support both features. Using type="email" has the advantage that it is semantically correct both using the pattern attribute has the advantage that there are several easy-to-use polyfills on the web which ensures support for a greater range of audience.

# Validating URLs

**<form>**

**<input type="url" required />**

**<input type="submit" value="Submit Now!">**

**</form>**

# Validating phone numbers

**<form>**

**Your phonenumber: <input type="tel" required />**

**<input type="submit" value="Submit now" />**

**</form>**

# Validating dates

**<form>**

**<input type="date">**

**<input type="submit" value="Submit Now!">**

**</form>**

# Placeholder text

An empty form might be a bit intimidating especially if you are uncertain of what might go in to the different input fields. Therefore some developers invented the idea of a sample text inside the fields – so the user would have an idea of what to type in. This is what we nowadays call the placeholder text. Usually the placeholder text would have a light gray color in order to distinguish it from the user’s actual input.

**<form>**

**First name: <input type="text" name="firstname" placeholder="Jane"/> <br>**

**Surname: <input type="text" name="surname" placeholder="Doe"/> <br>**

**<input type="submit" value="Submit now" />**

**</form>**

# Numbers as spinboxes

**<form>**

**<input type="number" min="10" max="20" step="2" value="16" required>**

**<input type="submit" value="Submit Now!">**

**</form>**