Chapter 8 - Events & other DOM properties

Console.dir function

Console.log shows the element DOM tree
Console.dir shows the element as an object with its properties

tagName / nodeName

Used to read tag name of an element
tagName → only exists for Element nodes
nodeName → defined for any node (text, comment etc.)

innerHTML and outerHTML

The innerHTML property allows to get the HTML inside the element as a string.
↳ Valid for element nodes only

The outerHTML property contains the full HTML.
innerHTML + the element itself.

innerHTML is valid only for element nodes. For other node types we can use nodeValue or data.

textContent

Provides access to the text inside the element: only text, minus all tags.

The hidden property

The "hidden" attribute and the DOM property specifies whether the element is visible or not.

```
< div hidden > I am hidden </div>

< div id = "element" > I can be hidden </div>

< Script >
  element. hidden = true ;
</ Script >
```

Attribute methods

1> elem. hasAttribute (name) → Method to check for existence of an attribute

2> elem. get Attribute (name) → Method used to get the value of an attribute

3> elem. Set Attribute (name, value) → Method used to set the value of an attribute.

4> elem. removeAttribute (name) → Method to remove the attribute from elem.

5> elem. attributes → Method to get the collection of all attributes

data -* attributes
We can always create custom attributes but the ones starting with "data-" are reserved for programmers use. They are available in a property named dataset.

If an element has an attribute named "data-one", its available as element.dataset.one

Insertion methods

We looked at some ways to insert elements in the DOM. Here is another way:

```
let div = document.createElement('div') //create
div.className = "alert"    // set class
div.innerHTML = "<span> hello </span>
document.body.append(div)
```

Here are some more insertion methods:

1> node.append(e) → append at the end of node

2> node.prepend(e) → Insert at the beginning of node

3> node.before(e) → Insert before node

4> node.after(e) → Insert after node

5> node.replaceWith(e) → replaces node with the given node.

Quick Quiz: Try out all these methods with your own webpage.

insert Adjacent HTML / Text / Element
This method is used to insert HTML. The first parameter is a code word, specifying where to insert. Must be one of the following:

1. "beforebegin" - Insert HTML immedialily before element

2. "afterbegin" - Insert HTML into element at the beginning

3. "beforeend" - Insert HTML into element at the end.

4. "afterend" - Insert HTML immedialily after element.

The second parameter is an HTML String

Example :

```
< div id= "div"> </div>
< script >
   div. insert Adjacent HTML ('beforebegin', '<p> Hello </p>');
   div. insert Adjacent HTML ('afterend', '<p> Bye </p>');
</Script>
```

The output would be :

```
<p> Hello </p>
< div id = "div" > </div>
<p> Bye </p>
```

Node removal
To remove a node, there's a method node.remove()

```
let id1 = document.getElementById("id1")

id1.remove()
```

ClassName and classList
If we assign something to elem.className, it replaces the whole string of classes.

Often we want to add/remove/toggle a single class.

1> elem.classList.add/remove("class") - Adds/removes a class

2> elem.classList.toggle("class") - Adds the class if it doesnt exist, otherwise removes it.

3> elem.classList.contains("class") - Checks for the given class, returns true/false

SetTimeout and setInterval
SetTimeout allows us to run a function once after the interval of time.

Syntax of setTimeout is as follows:

```
let timerId = setTimeout(fuction, <delay>, <arg1>, <arg2>)
```

$\downarrow$ returns a timerId

$\downarrow$ in ms

Clear Timeout is used to cancel the execution (in case we change our mind). For example:

```
let timerId = setTimeout (() => alert("never"), 1000);

Clear Timeout (timer Id)
```

↳ cancel the execution

set Interval method has a similar syntax as set Timeout :

```
let timerId = set Interval (function, <delay>, <arg1>,<arg2>)
```

All arguments have the same meaning. But unlike SetTimeout, it runs the function not only once, but regularly after the given interval of time.

To stop further calls, we can use clearInterval (timerId)

Browser Events
An event is a signal that something has happened. All the DOM nodes generate such signals.

Some important DOM events are:

Mouse events : click, contextmenu (right click), mouseover/ mouseout, mousedown/mouseup, mousemove

Keyboard events : keydown and keyup

form element events : submit, focus etc.

Document events : DOMContentLoaded

Handling Events
Events can be handled through HTML attributes

```
<input value = " Hey" onclick = "alert('hey')" type = "button">
```

↳ can be another JS function

Events can also be handled through the onclick property

```
elem.onclick = function() {
                  alert ("yes")
               };
```

Note : Adding a handler with JavaScript overwrites the existing handler

addEventListener and removeEventListener
addEventListener is used to assign multiple handlers to an event.

```
element.addEventListener(event, handler)

element.removeEventListener(event, handler)
```

↳ handler must be the same function object for this to work

The Event Object

When an event happens, the browser creates an event object, puts details into it and passes it as an argument to the handler

```
elem. onclick = function (event) {
                 ...
               }
```

event. type : Event type

(target) event. currentTarget : Element that handled the event

event. clientX / event. clientY : Coordinates of the cursor