[Lesson 10]

Roi Yehoshua 2018

[DAN.IT]
EDUCATION

[What we learnt last time?]

- Set

- Map

- Date and Time

[DAN.IT]
EDUCATION

[Our targets for today]

- Working with DOM

- DOM Tree

- Searching nodes

[DAN.IT]
EDUCATION

[Browser Environment]

→ Here's a bird's-eye view of what we have when JavaScript runs in a web-browser:

```
                            window


    DOM              BOM              JavaScript
  document         navigator            Object
    ...             screen              Array
                   location           Function
                    frames               ...
                    history
                XMLHttpRequest
```
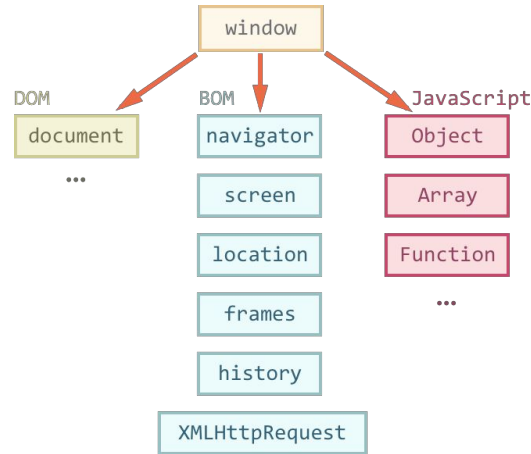
→ There's a "root" object called window. It has two roles:

  → First, it is a global object for JavaScript code

  → Second, it represents the "browser window" and provides methods to control it

[DAN.IT]
EDUCATION

4

[The window Object]

→ For instance, here we use the window as a global object:

```javascript
function sayHi() {  alert("Hello");
}

// global functions are accessible as properties of window
window.sayHi();
```

→ And here we use it as a browser window, to see the window height:

```javascript
alert(window.innerHeight); // inner window height
```

[DAN.IT]
E D U C A T I O N

[Document Object Model (DOM)]

→   The **document** object gives access to the page content

→   We can change or create anything on the page using it

→   For instance:

```
// change the background color to red
document.body.style.background = "red";

// change it back after 1 second
setTimeout(() => document.body.style.background = "", 1000);
```

→   The **DOM specification** explains the structure of a document and provides objects to manipulate it

→   It is being developed by two groups:

>   →   W3C – the documentation is at https://www.w3.org/TR/dom

>   →   WhatWG – publishing at https://dom.spec.whatwg.org

[DAN.IT]
EDUCATION

[Browser Object Model (BOM)]

→ **Browser Object Model** (BOM) are additional objects provided by the browser (host environment) to work with everything except the document

→ For instance:

    → The navigator object provides background information about the browser and the OS

    → The location object allows us to read the current URL and redirect the browser to a new one

→ Here's how we can use the location object:

```
alert(location.href); // shows current URL  if (confirm("Go to wikipedia?")) {
    location.href = "https://wikipedia.org"; // redirect the browser to another URL
}
```

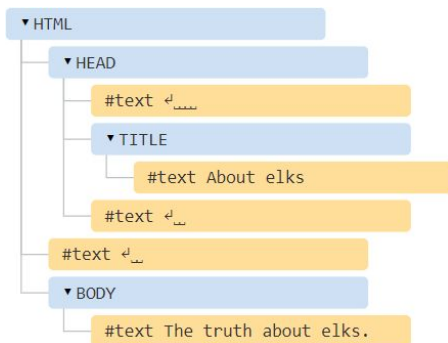→ BOM is the part of the general HTML specification

EDUCATION

[DOM Tree]

→   The backbone of an HTML document are tags

→   According to Document Object Model (DOM), every HTML tag is an object

→   Nested tags are called "children" of the enclosing one

→   The text inside a tag it is an object as well

→   All these objects are accessible using JavaScript

[DAN.IT]
E D U C A T I O N

[Example of DOM]

→ For instance, let's explore the DOM for this document:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>About elks</title>
</head>
<body>
    The truth about elks.
</body>
</html>
```

```
▼ HTML
    ▼ HEAD
        #text ↵␣␣
        ▼ TITLE
            #text About elks
        #text ↵␣
    #text ↵␣
    ▼ BODY
        #text The truth about elks.
```

→ We have a tree of elements: <html> is at the root, then <head> and <body> are its children, etc.

→ The text inside elements forms *text nodes*, labeled as #text

    → A text node contains only a string. It may not have children and is always a leaf of the tree.

    → Spaces and newlines – form text nodes and become a part of the DOM

[DAN.IT]
EDUCATION

9

[Autocorrection]

→   If the browser encounters malformed HTML, it automatically corrects it when making DOM

→   For instance, the top tag is always <html>

  →   Even if it doesn't exist in the document - the browser will create it

  →   The same goes for <body>

→   As an example, if the HTML file is a single word "Hello", the browser will wrap it into <html> and <body>, add the required <head>, and the DOM will be:

```
▼ HTML
    ▼ HEAD
    ▼ BODY
        #text Hello
```
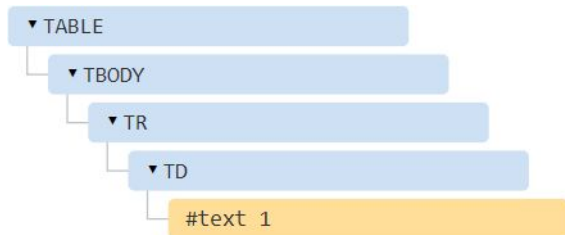
[DAN.IT]
EDUCATION
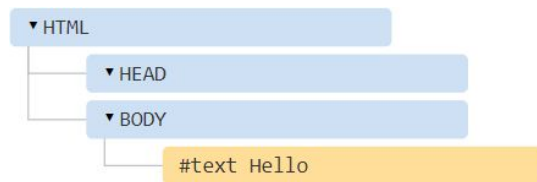
[Autocorrection]

→ An interesting "special case" is tables

→ By the DOM specification they must have <tbody>, but HTML text may (officially) omit it. Then the browser creates <tbody> in DOM automatically.

→ For the HTML:

```
<table id="table"><tr><td>1</td></tr></table>
```

→ DOM-structure will be:

▾ TABLE
    ▾ TBODY
        ▾ TR
            ▾ TD
                #text 1

[DAN.IT]
EDUCATION

11

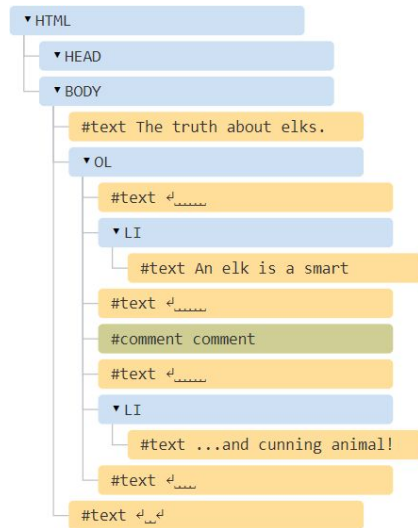# [Other Node Types]

→   Let's add more tags and a comment to the page:

```
<!DOCTYPE HTML>
<html>
<body>
    The truth about elks.
    <ol>
        <li>An elk is a smart</li>
        <!-- comment -->
        <li>...and cunning animal!</li>
    </ol>
</body>
</html>
```
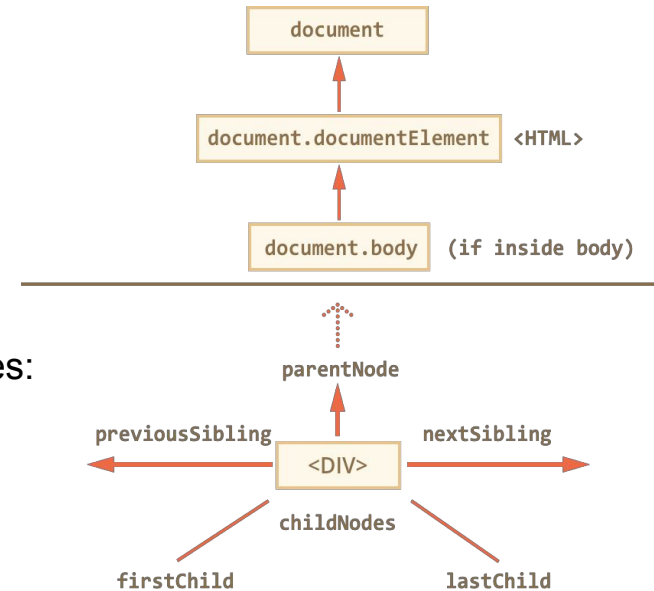
```
▼ HTML
    ▼ HEAD
    ▼ BODY
        #text The truth about elks.
        ▼ OL
            #text ⏎␣␣␣␣␣
            ▼ LI
                #text An elk is a smart
            #text ⏎␣␣␣␣␣
            #comment comment
            #text ⏎␣␣␣␣␣
            ▼ LI
                #text ...and cunning animal!
            #text ⏎␣␣␣
        #text ⏎␣⏎
```

→ Everything in HTML, even comments, becomes a part of the DOM

[DAN.IT]
EDUCATION

12

[Other Node Types]

→ There are 12 node types

→ In practice we usually work with 4 of them:

→ document – the "entry point" into DOM

→ element nodes – HTML-tags, the tree building blocks

→ text nodes – contain text

→ comments – sometimes we can put the information there, it won't be shown, but JS can read it from the DOM

[DAN.IT]
EDUCATION

Walking the DOM

→ The DOM allows to do anything with elements, but first we need to reach the corresponding DOM object, get it into a variable, and then we are able to modify it

→ All operations on the DOM start with the document object

→ From it we can access any node

→ The topmost tree nodes are available as document properties:

    → <html> = document.documentElement

    → <body> = document.body

    → <head> = document.head

document

document.documentElement <HTML>

document.body (if inside body)

parentNode

previousSibling   nextSibling

<DIV>

childNodes

firstChild   lastChild

[DAN.IT]
EDUCATION

Walking the DOM

→ For example, the following script changes the background color of the body:

```
<html>
<head>
    <title></title>
</head>
<body>
    <script>
        document.body.style.backgroundColor = "red";
    </script>
</body>
</html>
```

Walking the DOM

→ Note that a script cannot access an element that doesn't exist at the moment of running

→ In particular, if a script is inside <head>, then document.body is unavailable, because the browser did not read it yet

```
<html>
<head>
    <title></title>
    <script>
        document.body.style.backgroundColor = "red";
    </script>
</head>
<body>
</body>
</html>
```

```
⊗ Uncaught TypeError: Cannot read property 'style' of null
      at DocumentBody.html:7
```

[DAN.IT]
E D U C A T I O N

[Child Nodes]

→ **Child nodes (or children)** – elements that are direct children, i.e., they are nested exactly in the given one

   → For instance, \<head\> and \<body\> are children of \<html\> element

→ The childNodes collection provides access to all child nodes, including text nodes

→ The example below shows the children of document.body:

```html
<body>
    <div>Begin</div>
    <ul>
        <li>Information</li>
    </ul>
    <div>End</div>
    <script>
        for (let i = 0; i < document.body.childNodes.length; i++) {
            alert(document.body.childNodes[i]); // Text, DIV, Text, UL, ..., SCRIPT
        }
    </script>
    ...more stuff...
</body>
```

17

[DAN.IT]
EDUCATION

[Child Nodes]

→ Properties **firstChild** and **lastChild** give fast access to the first and last children

→ If there exist child nodes, then the following is always true:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

→ There's also a special function **elem.hasChildNodes()** to check whether there are any child nodes

[DOM Collections]

→ childNodes looks like an array, but it is rather a *collection* – a special array-like iterable object

→ There are two important consequences:

  → We can use for..of to iterate over it:

```
for (let node of document.body.childNodes) {
        alert(node); // shows all nodes from the collection
}
```

  → Array methods won't work, because it's not an array:

```
alert(document.body.childNodes.filter); // undefined (there's no filter method!)
```

→ DOM collections are read-only

  → We can't replace a child by something else, assigning childNodes[i] = ...

→ Almost all DOM collections are *live*

  → They reflect the current state of DOM

19

# Siblings and the Parent

→ **Siblings** are nodes that are children of the same parent

    → The next node of the same parent is available as **nextSibling**

    → The previous node of the same parent is available as **previousSibling**

→ The parent is available as **parentNode**

```html
<html><head></head><body><script>
    // HTML is "dense" to evade extra "blank" text nodes.

    // parent of <body> is <html>
    alert(document.body.parentNode === document.documentElement); // true

    // after <head> goes <body>  alert(document.head.nextSibling); //
    HTMLBodyElement

    // before <body> goes <head>
    alert(document.body.previousSibling); // HTMLHeadElement
</script></body></html>
```

Element-Only Navigation

→ Navigation properties listed above refer to *all* nodes

  → For instance, in childNodes we can see text nodes, element nodes, and even comment nodes if there exist

→ But for many tasks we want to manipulate only element nodes that represent tags

→ The following navigation links take only *element nodes* into account:
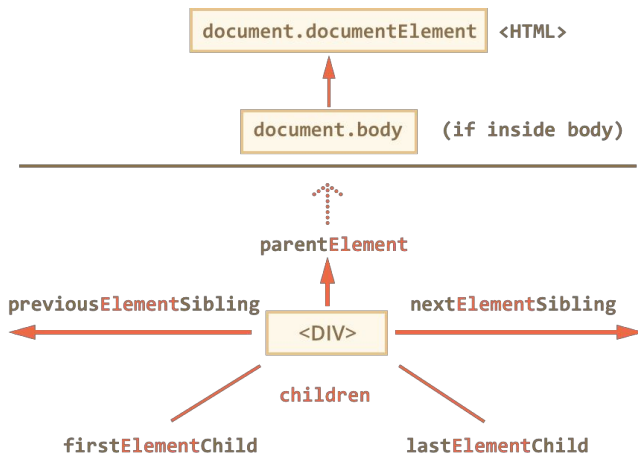
Element-Only Navigation

→ For example, the following script it shows only the child elements of document.body:

```html
<html>
<body>
    <div>Begin</div>
    <ul>
        <li>Information</li>
    </ul>
    <div>End</div>

    <script>
        for (let elem of document.body.children) {
          alert(elem); // DIV, UL, DIV, SCRIPT
        }
    </script>
</body>
</html>
```

[Exercise (1)]

→ For the page:

```html
<html>
<body>
    <div>Users:</div>
    <ul>
        <li>John</li>
        <li>Adam</li>
    </ul>
</body>
</html>
```

→ How to access:

→ The <div> DOM node?

→ The <ul> DOM node?

→ The second <li> (with Adam)?

[DAN.IT]
EDUCATION

[Exercise (2)]

→ Write the code to paint all diagonal table cells in red:

→ The result should be:

1:1	2:1	3:1	4:1
1:2	2:2	3:2	4:2
1:3	2:3	3:3	4:3
1:4	2:4	3:4	4:4

→ Hint: Use the browser inspector to examine the DOM tree structure

```html
<html>
<body>
    <table>
        <tr>
            <td>1:1</td>
            <td>2:1</td>
            <td>3:1</td>
            <td>4:1</td>
        </tr>
        <tr>
            <td>1:2</td>
            <td>2:2</td>
            <td>3:2</td>
            <td>4:2</td>
        </tr>
        <tr>
            <td>1:3</td>
            <td>2:3</td>
            <td>3:3</td>
            <td>4:3</td>
        </tr>
        <tr>
            <td>1:4</td>
            <td>2:4</td>
            <td>3:4</td>
            <td>4:4</td>
        </tr>
    </table>
</body>
</html>
```

[DAN.IT]
EDUCATION

[Searching Elements]

→ DOM navigation properties are great when elements are close to each other

→ What if they are not? How to get an arbitrary element of the page?

→ There are additional searching methods for that, which we'll see on the next slides

[DAN.IT]
EDUCATION

[getElementById]

→ If an element has the id attribute, then there's a global variable whose name is identical to that id

→ We can use this variable to access the element, like this:

```html
<div id="elem">Some element</div>

<script>
    alert(elem); // DOM-element with id="elem"
    alert(window.elem); // accessing global variable like this also works
</script>
```

→ However, if we declare another variable with the same name, it shadows the variable created by the browser

→ Also, when we look in JS and don't have HTML in view, it's not obvious where the variable comes from

26

[getElementById]

→ The better alternative is to use a special method document.getElementById(id):

```html
<div id="elem">Some element</div>

<script>
    let elem = document.getElementById("elem");
    elem.style.background = "red";
</script>
```

→ The id must be unique

  → If there are multiple elements with the same id, the behavior of the corresponding methods is unpredictable

  → The browser may return any of them at random

  → So please stick to the rule and keep id unique

[getElementsBy*]

→   There are also other methods to look for nodes:

  →   **elem.getElementsByTagName**(tag) looks for elements with the given tag and returns the collection of them

  →   **elem.getElementsByClassName**(className) returns elements that have the given CSS class

  →   **document.getElementsByName**(name) returns elements with the given name attribute

    →   Exists for historical reasons, very rarely used

```html
<div>First</div>
<div>Second</div>
<div>Third</div>

<script>
    // get all divs in the document
    let divs = document.getElementsByTagName("div");
    for (let div of divs) {
        alert(div.innerHTML); // First, Second, Third
    }
</script>
```

[DAN.IT]
EDUCATION

[QuerySelectorAll]

→ **elem.querySelectorAll**(css) returns all elements inside elem matching the given CSS selector (elem can also be the document itself)

→ That's the most often used and powerful method

→ For instance, here we look for all <li> elements that are last children:

```html
<ul>
    <li>The</li>
    <li>test</li>
</ul>
<ul>
    <li>has</li>
    <li>passed</li>
</ul>
<script>
    let elements = document.querySelectorAll('ul > li:last-child');

    for (let elem of elements) {  alert(elem.innerHTML); // "test",
        "passed"
    }
</script>
```

[DAN.IT]
EDUCATION

[QuerySelector]

→ **elem.querySelector**(css) returns the first element for the given CSS selector

  → The result is the same as elem.querySelectorAll(css)[0], but the latter is looking for all elements and picking one, thus elem.querySelector is faster and shorter to write

```html
<ul>
    <li>The</li>
    <li>test</li>
</ul>
<ul>
    <li>has</li>
    <li>passed</li>
</ul>
<script>
    let firstList = document.querySelector('ul:first-child');
    firstList.style.color = "red";
</script>
```

- The
- test

- has
- passed

[Live Collections]

→ All methods "getElementsBy*" return a *live* collection

→ Such collections always reflect the current state of the document and "auto-update" when it changes.

→ In the example below, there are two scripts:

  → The first one creates a reference to the collections of <div>. As of now, its length is 1.

  → The second script runs after the browser meets one more <div>, so its length is 2.

```
<div>First div</div>
<script>
    let divs = document.getElementsByTagName('div');
    alert(divs.length); // 1
</script>

<div>Second div</div>
<script>
    alert(divs.length); // 2
</script>
```

[DAN.IT]
EDUCATION

[Live Collections]

→   In contrast, querySelectorAll returns a *static* collection

→   It's like a fixed array of elements

→   If we use it instead, then both scripts output 1:

```
<div>First div</div>
<script>
    let divs = document.querySelectorAll('div');  alert(divs.length); //
    1
</script>

<div>Second div</div>

<script>
    alert(divs.length); // 1
</script>
```

[DAN.IT]
E D U C A T I O N

[Summary]

→ There are 6 main methods to search for nodes in DOM:

Method	Searches by…	Can call on an element?	Live?
getElementById	id	-	-
getElementsByName	name	-	√
getElementsByTagName	tag	√	√
getElementsByClassName	class	√	√
querySelector	CSS-selector	√	-
querySelectorAll	CSS-selector	√	-

→ Note that methods getElementById and getElementsByName can only be called in the context of the document: document.getElementById(...), while other methods can be called on elements too, e.g., elem.querySelectorAll(...) will search inside elem (in the DOM subtree)

[DAN.IT]
EDUCATION

Exercise (3)

→ Here's a document with a table and a form:

→ How to find?

→ The table with id="age-table"

→ All label elements inside that table (there should be 3 of them)

→ The first td in that table (with the word "Age")

→ The form with the name search

→ The first input in that form

→ The last input in that form

```html
<!DOCTYPE HTML>
<html>
<body>
    <form name="search">  Search the visitors:
        <table id="age-table">
            <tr>
                <td>Age:</td>
                <td id="age-list">
                    <label>
                        <input type="radio" name="age" value="young">less than 18
                    </label>
                    <label>
                        <input type="radio" name="age" value="mature">18-50
                    </label>
                    <label>
                        <input type="radio" name="age" value="senior">more than 50
                    </label>
                </td>
            </tr>
            <tr>
                <td>Additionally:</td>
                <td>
                    <input  type="text" name="info[0]">
                    <input  type="text" name="info[1]">
                    <input  type="text" name="info[2]">
                </td>
            </tr>
        </table>

        <input type="submit" value="Search!">
    </form>
</body>
</html>
```

34

[DAN.IT]
E D U C A T I O N

DOM Node Classes

→ Each DOM node belongs to the corresponding built-in class

→ The root of the hierarchy is EventTarget, that is inherited by Node, and other DOM nodes inherit from it

35
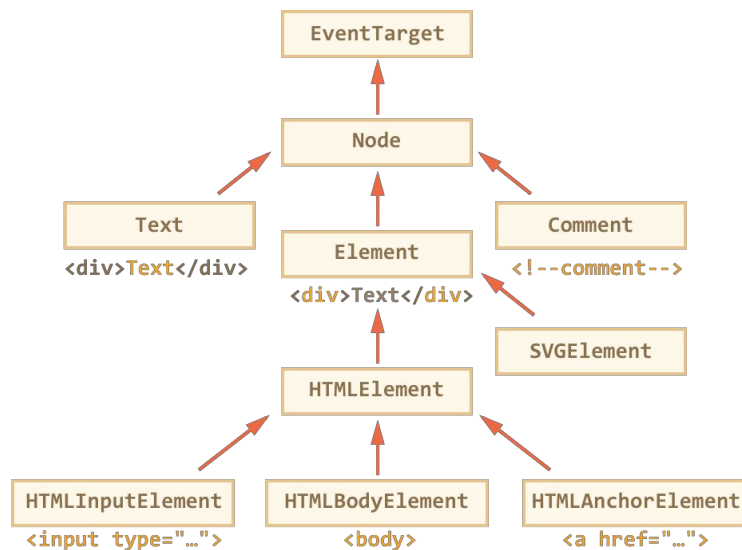
DOM Node Classes

→ To see the DOM node class name, we can recall that an object has the constructor property, which references to the class constructor:

```
alert(document.body.constructor.name); // HTMLBodyElement
```

→ We also can use instanceof to check the inheritance:

```
alert(document.body instanceof HTMLBodyElement); // true  alert(document.body
instanceof HTMLElement); // true  alert(document.body instanceof Element); //
true  alert(document.body instanceof Node); // true  alert(document.body
instanceof EventTarget); // true
```

→ To inspect a DOM element in the console, use:

→ **console.log**(elem) shows the element DOM tree

→ **console.dir**(elem) shows the element as a DOM object, good to explore its properties

[DAN.IT]
EDUCATION

DOM Node Properties

→ DOM nodes have different properties depending on their class

→ The full set of properties and methods of a given node comes as the result of the

→ inheritance hierarchy

→ For example, let's consider the DOM object for an <input> element

→ It belongs to the **HTMLInputElement** class

→ It gets properties and methods as a superposition of:

   → HTMLInputElement – this class provides input-specific properties

   → HTMLElement – provides common HTML element methods (and getters/setters)

   → Element – provides generic element methods

   → Node – provides common DOM node properties

   → EventTarget – gives the support for events (to be covered)

   → and finally it inherits from Object, so "pure object" methods like toString are also available

[DAN.IT]
E D U C A T I O N

[innerHTML]

→ The **innerHTML** property allows to get the HTML inside the element as a string

→ We can also modify it, so it's one of most powerful ways to change the page.

→ The example shows the contents of document.body and then replaces it completely:

```html
<body>
    <p>A paragraph</p>
    <div>A div</div>

    <script>
        alert(document.body.innerHTML); // read the current contents
        document.body.innerHTML = 'The new BODY!'; // replace it
    </script>
</body>
```

→ We can append "more HTML" by using elem.innerHTML+="something"

→However, we should be very careful about it, because this causes a full overwrite of the element's content

38

[textContent: pure text]

→ The **textContent** provides access to the *text* inside the element: only text, minus all <tags>

→ Compare the two:

```
<body>
    <div id="“"></div>
    <div></div>

    <script>
        document.body.children[0].innerHTML = "Hello<br/>World";
        document.body.children[1].textContent = "Hello<br/>World";
    </script>
</body>
```

Hello
World
Hello
World

→ The first <div> gets the text "as HTML": all tags become tags, so we see the line break

→ The second <div> gets the name "as text", so we literally see Hello
World

[DAN.IT]
E D U C A T I O N

The "hidden" Property

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

→ We can use it in HTML or assign using JavaScript, like this:

```
<body>
    <div>Both divs below are hidden</div>
    <div hidden>With the attribute "hidden"</div>
    <div id="elem">JavaScript assigned the property "hidden"</div>
    <script>
        elem.hidden = true;
    </script>
</body>
```

Both divs below are hidden

→ Technically, hidden works the same as style="display:none". But it's shorter to write.

[DAN.IT]
E D U C A T I O N

[More Properties]

→ DOM elements have additional properties, many of them provided by their class:

  → **value** – the value for <input>, <select> and <textarea>

  → **href** – the "href" for <a href="...">

  → **id** – the value of "id" attribute, for all elements

  → and many more…
```
<input type="text" id="elem" value="5"/>
<script>
    alert(elem.type); // text  alert(elem.id); //
    elem  alert(elem.value); // 5
</script>
```

→ Most standard HTML attributes have the corresponding DOM property, and we can access them using JavaScript

→ You can output the full list of properties of a given element using **console.dir**(elem)

[DAN.IT]
E D U C A T I O N

[Exercise (4)]

→   What does this code show?

```html
<html>
<body>
    <script>
        let body = document.body;
        body.innerHTML = "<p>My paragraph</p>";

        alert(document.body.lastChild.constructor.name); // what's here?

    </script>
</body>
</html>
```

[DAN.IT]
EDUCATION

[Exercise (5)]

→ Create a colored clock like here:

11:08:42

[DAN.IT]
EDUCATION

[Control questions]

1. What is DOM?

2. How can we walk a DOM Tree?

3. What is Parent, Child and Siblings in DOM Tree?

4. Which methods can you use to get a single DOM element?

5. What are the ways to get multiple DOM elements?

[DAN.IT]
EDUCATION

[Materials]

Core materials:
https://learn.javascript.ru/browser-environment
https://learn.javascript.ru/dom-nodes
https://learn.javascript.ru/traversing-dom
https://learn.javascript.ru/basic-dom-node-properties

Additional materials:

https://developer.mozilla.org/ru/docs/DOM/DOM_Reference/Введение

Video materials:
https://youtu.be/KEsA5wAg5Xc

[DAN.IT]
EDUCATION