DOM

softserve

Agenda

- Ø DOM
  - Ø Main concept
  - Ø **DOM methods** and **properties**
  - Ø **Element search methods**
- Ø DOM navigation
  - Ø **document.body**
  - Ø **ChildNodes, children properties**
  - Ø **Node object**
  - Ø parentNode, previousSibling, nextSibling **properties**
  - Ø **navigation in tables and forms**
- Ø **DOM Nodes**
  - Ø **Nodes properties**
  - Ø **Creating, inserting, deleting nodes**

softserve

DOM

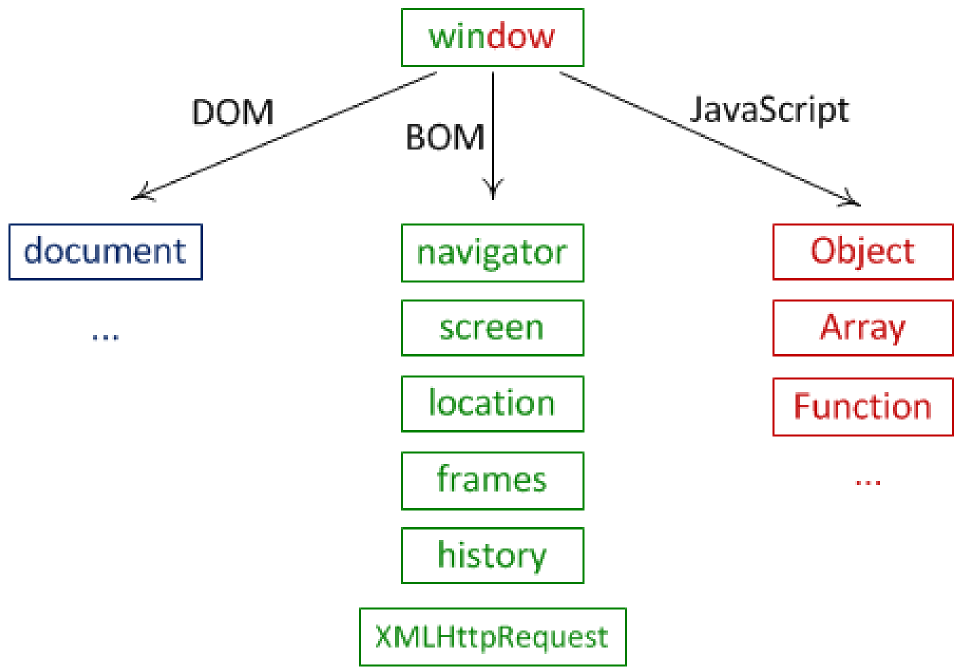
Main concept

One of the key tasks of JavaScript is user interaction and manipulation of web page elements. For JavaScript, a web page is available as a **document object model** or **DOM** for short.

The DOM describes the structure of a web page in a tree view and provides developers with a way to access and modify individual elements of a web page.

When a web **page is loaded**, the browser creates a **D**ocument **O**bject **M**odel of the page.

The browser gives access to the hierarchy of objects that we can use for development. The figure shows the **structure of the main browser objects**. At the top is a **window**, which is also called a **global object**.
Document Object Model (DOM). Available through document. Gives access to the contents of the page.
Browser Object Model (BOM). BOMs are objects for working with anything but a document.
JavaScript objects and functions. Javascript is the linking language of all this.

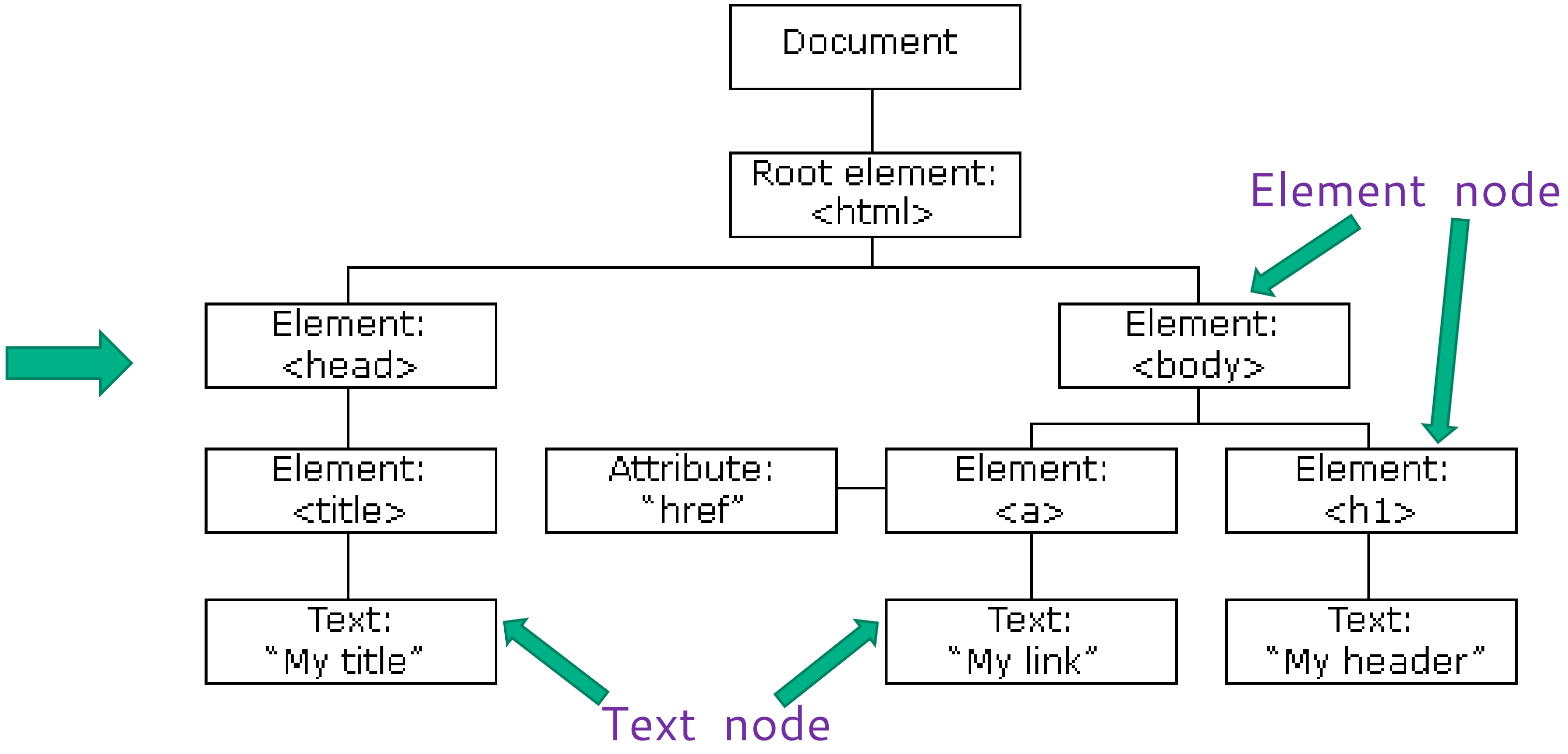


DOM tree

The DOM represents a document as a tree. The tree is made up of parent-child relationships, a parent can have one or many children nodes.

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

    <body>
        <h1>My header</h1>
        <a href="#">My link</a>
    </body>
</html>
```



Thus, all components are ordered in a DOM in a hierarchical manner, where **each component represents a separate node**. That is, each element, for example, element h1, is a node. But also the text inside the element

DOM methods and **properties**

DOM **methods** are **actions** you can perform (on HTML Elements).

DOM **properties** are **values** (of HTML Elements) that you can set or change.

Using DOM methods and properties, a user can:

- Modify HTML elements on the page
- Modify attributes
- Change CSS styles
- Delete existing HTML elements and attributes
- Add new HTML elements and attributes
- Respond to user actions against HTML elements
- Create new events

Element search methods

The following methods are used to search for HTML elements on a page:

- **document.getElementById**(value): selects an element whose id attribute is value
- **document. getElementsByTagName**(value): selects all elements for which the tag is equal to value
- **document. getElementsByClassName**(value): selects all elements that have the value class
- **document. querySelector**(value): selects the first element that matches the value css selector
- **document. querySelectorAll**(value): selects all elements that match the value css selector

! The querySelectorAll() method does not work in Internet Explorer 8 and earlier versions.

softserve

Element search methods

```
<body>
      <div class=”test2”></div>
      <p id="test" class="test2"></p>
      <p class=”test2”></p>
      <p></p>
      <input type="text" />
      <input type="radio" />
      <script>
          let id = document.getElementById("test");
          let class = document.getElementsByClassName("test2");
          let tag = document.getElementsByTagName("p");
          let x = document.querySelectorAll("div.test2");
          let y = document.querySelector("p.test2“);
      </script>
</body>
```

softserve

DOM **navigation**

softserve

DOM navigation. Object Document

The **document object** is the main "entry point" in the DOM. From it we can access any node.

document allows you to **access** frequently used **elements** of a web page through the **properties**:

- **document.documentElement**: provides access to the root <html> element
- **document.body**: provides access to the <body> element in a web page
- **document.images**: contains a collection of all image objects (img elements)
- **document.links**: contains a collection of links – elements <a> and <area> that have the href attribute defined
- **document.anchors**: provides access to a collection of <a> elements that have a name attribute defined
- **document.forms**: contains a collection of all forms on a web page

DOM navigation. document.body

There is one subtlety: **document.body** can be null. In particular, if the script is located in <head>, document.body is not available in it, because the browser has not yet read it. You cannot access an element that does not exist at the time the script is executed. The first alert will print null:

```
<head>
  <script>
    alert("document.body from HEAD: " + document.body); // null
  </script>
</head>
<body>
  <script>
    alert("document.body from BODY: " + document.body);
  </script>
</body>
```

! In the DOM, null means "does not exist" or "there is no such node."

DOM navigation. Node object

Each individual node, whether it is an html element, its attribute or text, in the DOM structure is represented by a **Node object**. This object provides a number of properties with which we can obtain information about this node:

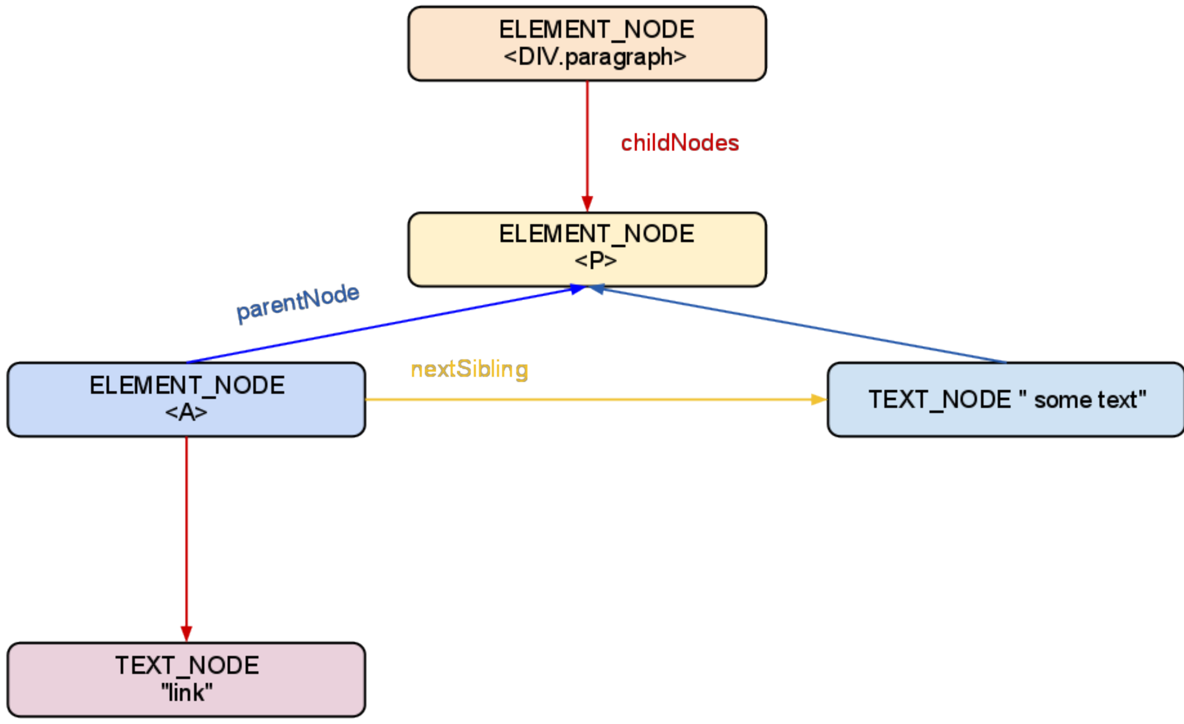
- **childNodes**: contains a collection of child nodes
- **firstChild**: returns the first child node of the current node
- **lastChild**: returns the last child node of the current node
- **previousSibling**: returns the previous item that is on par with the current
- **nextSibling**: returns the next element that is flush with the current
- **ownerDocument**: returns the root node of the document
- **parentNode**: returns the element that contains the current node
- **nodeName**: returns node name
- **nodeType**: returns node type as a number
- **nodeValue**: returns or sets the value of the node in plain text

softserve

DOM navigation. ChildNodes property

From the parent node, all children can be retrieved. There are several ways to do this. The "**childNodes**" property holds all child elements, including text ones. We will consistently display all the child nodes of "document.body":

```
<body>
    <div>
      <p>
        <a>link</a> some text
      </p>
    </div>
   <script>
     let bodyNodes = document.body.childNodes;
     for (let i = 0; i < bodyNodes.length; i++) {
        alert(bodyNodes[i]); // Text DIV Text SCRIPT
     }
   </script>
</body>
```



softserve

DOM navigation. Children property

The "**children**" property lists only child element nodes (that is, tags, text, and others are missing). We will subsequently display all the document.body child nodes:

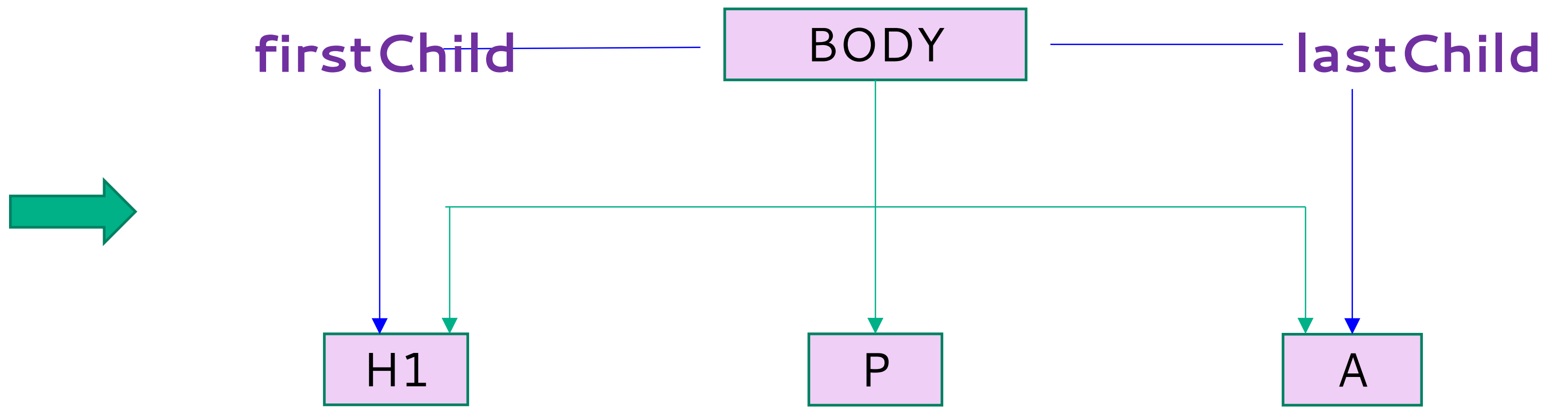
```
<body>
    <div>
      <p>
        <a>link</a> some text
      </p>
    </div>
    <!-- Comment -->
   <script>
      let bodyElements = document.body.children;
      for (let i = 0; i < bodyElements.length; i++) {
         alert(bodyElements[i]); // DIV SCRIPT
      }
   </script>
</body>
```

softserve

DOM navigation. fisrstChild and lastChild properties

You can use the **firstChild** and **lastChild** properties of the DOM node to access the first and last direct *child node* of a node, respectively. If the node doesn't have any child element, it returns *null*.

```
<body>
    <h1>Some header</h1>
    <p>Some text</p>
    <a>Some link</a>
</body>
```



The "firstChild" and "lastChild" properties are a faster and shorter way to access the first and last elements of "childNodes"

The equalities are true:

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

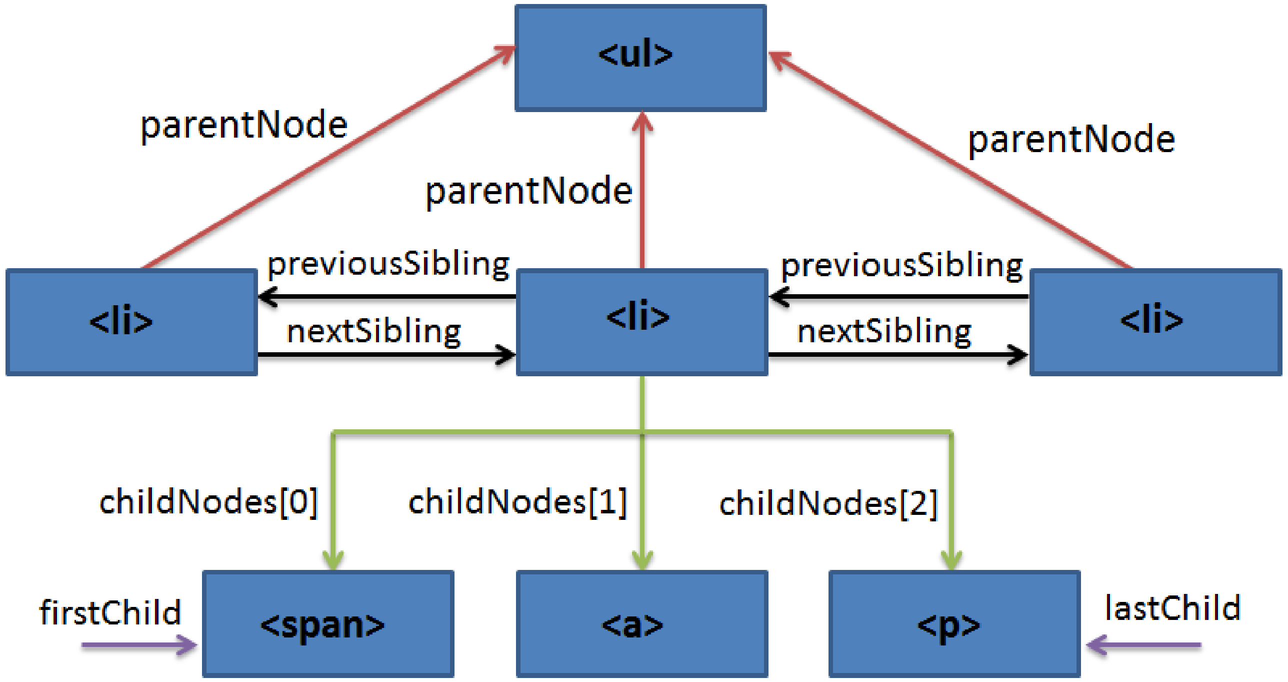
softserve

DOM navigation. ParentNode, previousSibling, nextSibling **properties**

Property "**parentNode**" refers to the parent node

The "**previousSibling**" and "**nextSibling**" properties give access to the left and right neighbors:

```
<ul>
    <li id="liElem">node</li>
    <li><span>node</span><a href="#">node</a><p>node</p></li>
    <li>node</li>
</ul>
```



softserve

DOM navigation. Tables

Tables have additional properties for more convenient navigation:

caption/tHead/tFoot - links to table elements CAPTION, THEAD, TFOOT

tBodies - a list of elements of the TBODY table, according to the specification there may be several

rows - a list of TR lines of the table/section THEAD/FFOOT/TBODY

cells - list of TD/TH cells

sectionRowIndex - line number in the current section THEAD/TBODY

rowIndex - row number in the table

cellIndex - cell number in a row

```
<body>
    <table>
       <tr><td>Monday</td><td>Tuesday</td></tr>
       <tr><td>Wednesday</td><td>Thursday</td></tr>
    </table>
    <script>
       let elem = document.body.children[0];
       alert(elem.rows[0].cells[0].innerHTML);        // "Monday"
    </script>
 </body>
```

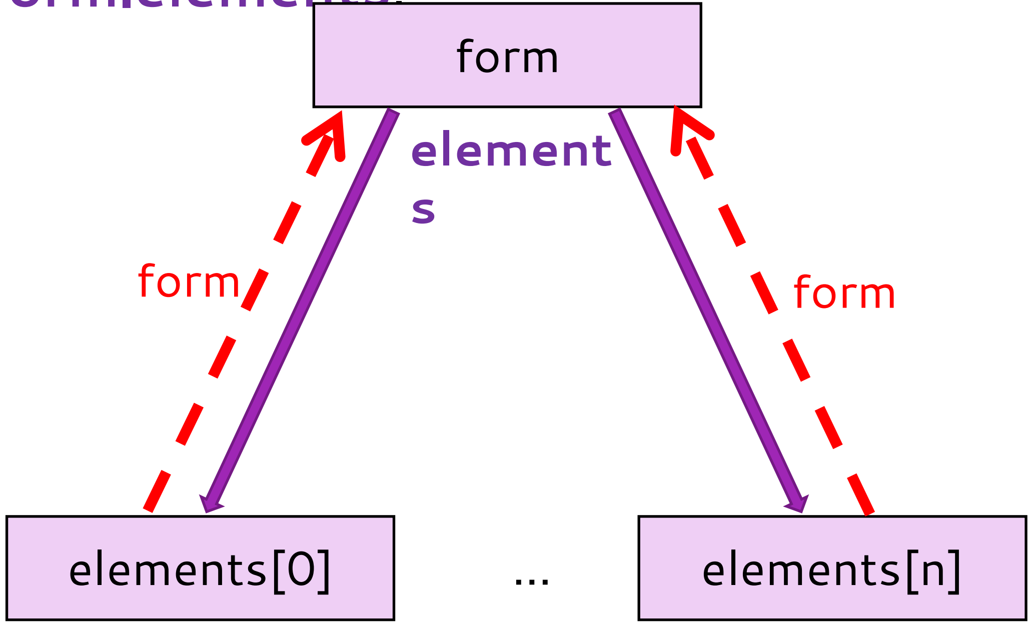
DOM navigation. Forms

One of the ways to interact with users is html forms. To get the form, we can use both the serial number in the document and its name **document.forms[index/name]**.

document.**forms**[0] – first form in a document

document.**forms**.**registerForm** – name form "registerForm" (name="registerForm")

When we have already received the form, any item is available in the named collection. **form.elements**.



```
<body>
   <form name="registerForm">
      <input name="surname" value="1" />
      <input name="email" value="2" />
   </form>
   <script>
      let myForm = document.forms.registerForm;
      let elem = myForm.elements.surname;
      alert(elem.value); // 1
   </script>
</body>
```

There may be several items with the same name. In this case, the "elements" property will return an array of elements

DOM navigation. Additional navigation properties

All modern browsers, including IE9 +, support sitelinks:

- **firstElementChild** – first child element (=children[0])
- **lastElementChild** – last child element (=children[children.length-1])
- **childElementCount** – number of child elements (=children.length)
- **nextElementSibling** – right sibling
- **previousElementSibling** – left sibling

Any nodes other than elements are simply ignored:

```
<body>
   firstElementChild: <p>Some paragraph</p>
   <!-- Comment -->
   lastElementChild: <div>Some DIV</div>
   <script>
       alert(document.body.firstElementChild.nextElementSibling); // DIV
   </script>
</body>
```

DOM
Nodes

Nodes properties. InnerHTML

The **innerHTML** property allows you to get the HTML content of an element as a string.

We can also change it. This is one of the most powerful ways to change the content on a page.

```
<body>
    <h1 id="header">I’am header</h1>
    <script>
        document.getElementById("header").innerHTML = "New header";
    </script>
</body>
```

Nodes properties. InnerHTML

Through **innerHTML** we **can add** not only text, but also **other tags**. When recording, you can record anything, and if we make a mistake, the browser will correct the incorrect HTML code:

```
<body>
   <script>
          document.body.innerHTML = '<b>Text message </b '; // forgot to close the tag
          alert( document.body.innerHTML );  // <b>Text message</b> (fixed)
   </script>
</body>
```

! If innerHTML inserts a <script> tag into the document, it becomes part of the HTML, but does not start.

softserve

Nodes properties. OuterHTML

outerHTML property holds the whole HTML Node.

You can assign to some variable HTLM Node with **outerHTML**, but, if you assign to this variable some text that can be interpreted as HTML document, it will be separate node and origin document will not be changed.

```
<body>
   <div id="hi">Hello <b>World</b></div>
   <script>
      let d = document.body.children[0];
      alert(d.outerHTML); // <div id="hi">Hello <b>World</b></div>
       d.outerHTML = "<p>New HTML!</p>";
      alert(d.tagName); // DIV
      alert(d.innerHTML); // Hello <b>World</b </b>
    </script>
</body>
```

Nodes properties. NodeType

The **nodeType** property is read only. It returns the type of a node.

The most important nodeType properties are:

Node type	Type	Example
ELEMENT_NODE	1	<p id="main">Main paragraph</p>
ATTRIBUTE_NODE	2	id = "main" (deprecated)
TEXT_NODE	3	Main paragraph
COMMENT_NODE	8	<!-- Some comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

```
<div id="first">Main block here</div>
<div id="second"></div>
<script>
  document.getElementById("second").innerHTML = document.getElementById("first").nodeType;
</script>
```

softserve

Nodes properties. nodeName, tagName

There are two properties: "**nodeName**" and "**tagName**", which contain the name (tag) of the node element. The name of the HTML tag is always uppercase. For "document.body":

alert(document.body.**nodeName**); // BODY

alert(document.body.**tagName**); // BODY

The difference is reflected in the property names, but not obvious:

- the "nodeName" property – defined for many types of DOM nodes
- the tagName property – only elements have it

With the help of "tagName" they work only with elements, and "nodeName" with all types of nodes:

```
<body><!-- Comment-->
   <script>
      // for comment
      alert(document.body.firstChild.nodeName);  // #comment
      alert(document.body.firstChild.tagName);  // undefined
      // for a document
      alert(document.nodeName);  // #document
      alert(document.tagName);  // undefined , because DOM root is not an element
   </script>
</body>
```

softserve

Nodes properties. Other useful properties

DOM nodes have type-specific properties, for example:

- **value** - value for "input", "select" or "textarea“
- **id** – identifier
- **className** - class
- **href** - link address
- a lot others

```
<body>
   <input type="text" id="my-input" class="my-class" value="my-value" />
   <script>
      let myInput = document.getElementById("my-input");
      alert(myInput.type); // text
      alert(myInput.id); // my-input
      alert(myInput.className) // my-class
      alert(myInput.value); // "my-value"
   </script>
</body>
```

softserve

Creating nodes. createElement, createTextNode

A DOM node can be created using two methods:

1) document.**createElement**(*tag*) – creates a new element with the given tag *tag*

2) document.**createTextNode**(*text*) – creates a new text node with the given text *text*

```
let newHeader = document.createElement("h1");
let newText = document.createTextNode("New text message!");
```

We can assign properties for new element:

```
let newHeader = document.createElement("h1");
newHeader.className = "newClass";
newHeader.id = "newId";
newHeader.innerHTML = "I header!";
```

softserve

Cloning nodes. cloneNode

Sometimes elements are quite complex in composition, and it is much easier to clone them than using separate calls to create from the content.

elem.**cloneNode(true) –** creates a "deep" clone of the element, with all attributes and child elements

elem.**cloneNode(false)** – creates a clone without children.

```
<div>
    <span>Big text here!</span>
</div>
<script>
    let spanElem = document.getElementsByTagName("span")[0];
    let newSpan = spanElem.cloneNode(true);  // clone node
    console.log(newSpan.innerHTML); // Big text here!
    newSpan.innerHTML = "Another text here!";
    console.log(newSpan.innerHTML); // Another text here!
</script>
```

softserve

Nodes. Insertion methods

Creating elements is not enough; you still need to add them to the web page.

This set of methods provides more ways to insert:

- node.**append**(...nodes or strings) – append nodes or strings at the end of node
- node.**prepend**(...nodes or strings) – insert nodes or strings at the beginning of node
- node.**before**(...nodes or strings) –– insert nodes or strings before node
- node.**after**(...nodes or strings) –– insert nodes or strings after node
- node.**replaceWith**(...nodes or strings) –– replaces node with the given nodes or strings.

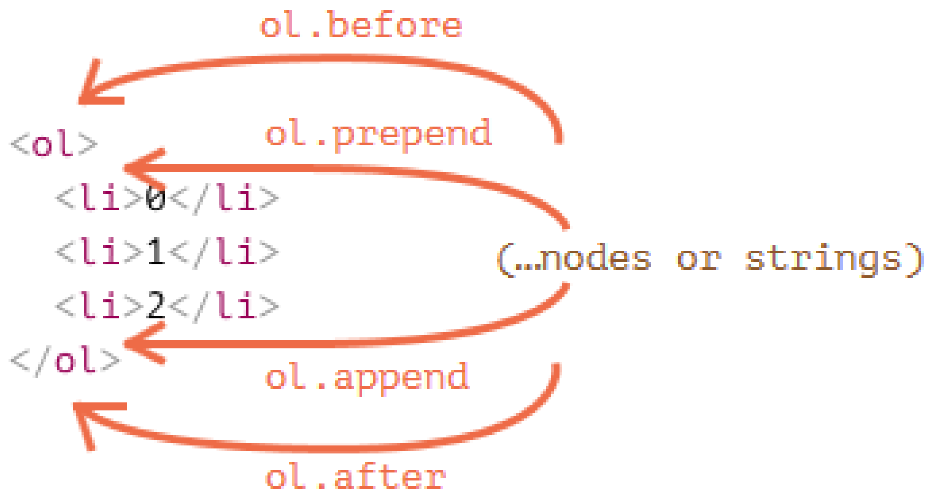
There are several other, older, insertion methods:

parentElem.**appendChild**(node) - appends node as the last child of parentElem.

parentElem.**insertBefore**(node, nextSibling) - inserts *node* before *nextSibling* into parentElem

Nodes. Insertion methods

```
<ol id="ol">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
<script>
  ol.before('before'); // insert the string "before" before <ol>
  ol.after('after'); // insert the string "after" after <ol>
  let liFirst = document.createElement('li');
  liFirst.innerHTML = 'prepend';
  ol.prepend(liFirst); // insert liFirst at the beginning of <ol>
  let liLast = document.createElement('li');
  liLast.innerHTML = 'append';
  ol.append(liLast); // insert liLast at the end of the <ol>
</script>
```



softserve

Nodes. Insertion methods. insertAdjacentHTML()

The insertAdjacentHTML() method inserts a text as HTML, into a specified position.

elem.insertAdjacentHTML(where, html)

The first parameter is a special word indicating where to insert with respect to elem.
The value must be one of the following :

"afterbegin" – after the beginning of the element (as the first child)
"afterend" – after the element
"beforebegin" – before the element
"beforeend" – before the end of the element (as the last child)

The second parameter is the HTML string that will be inserted exactly as HTML.

```
<div id="div"></div>
<script>
  div.insertAdjacentHTML('beforebegin', '<p>Hi</p>');
  div.insertAdjacentHTML('afterend', '<p>By</p>');
</script>
```

```
<p>Hi</p>
<div id="div"></div>
<p>By</p>
```

softserve

Node Attributes

DOM nodes provide access to the attributes of HTML elements. Attributes are accessed using standard methods:

- **hasAttribute(name)** – checks for an attribute
- **getAttribute(name)** – gets attribute value
- **setAttribute(name, value)** – sets attribute
- **removeAttribute(name)** – removes attribute

Unlike properties, **attributes**:

- Can only be strings.
- Their name is case insensitive (because this is HTML)
- Visible in "innerHTML" (except for older IE)
- All attributes of an element can be obtained using the "attributes" property

```
<p id="item" info="JavaScript">Some text</p>
  <script>
    alert(item.getAttribute('info') ); // (1) 'JavaScript', reading
     item.setAttribute('Example', 111); // (2), writing
    alert(item.outerHTML ); // (3), check if there is an attribute in HTML (yes)
  </script>
```

Removing and replacing nodes

There are a number of methods for removing and replacing nodes :

node.**remove**() – removes the node from the DOM tree

parentElem.**removeChild**(elem) – removes "elem" from the list of children "parentElem"

parentElem.**replaceChild**(elem, currentElem) – among children, "parentElem" replaces "currentElem" with "elem"

```
<div id="box">
   <div id="one">1</div>
   <div id="two">2</div>
   <div id="three">3</div>
</div>
<script>
   let elem = document.querySelector("#one");
   elem.remove();
   alert(document.querySelector("#box").innerHTML); // #One block disappeared
   let  parent = document.querySelector("#box");
   let  child = document.querySelector("#three");
   parent.removeChild(child);
   alert(document.querySelector("#box").innerHTML); // #Three block disappeared
</script>
```

softserve

Change element style

There are mainly two approaches used to work with the style properties of elements in JavaScript:

1) Change the **style** property
2) Changing the value of the **class** attribute

1) The **style** property represents a complex object for controlling the style and is directly mapped to the style attribute of the html element. This object contains a set of CSS properties: **element.style.CSS_property**.

```
elem.style.width="200px"    =    style="width:200px"
```

However, a number of css properties in the names have a hyphen, for example, font-size. JavaScript does not use a hyphen for these properties, but camelCase is used.

```
let root = document.documentElement;

root.style.fontSize = "14px";
```

! You must specify the units in style, as in CSS, just root.style.fontSize = 14 **– it won't work**

softserve

Change element style

2) Using the **className** property, you can set the class attribute of the html element.

```
<body class="main page">
    <script>
        alert(document.body.className); // main page
    </script>
</body>
```

If we assign something to elem.className, then this replaces the entire line with the classes. Sometimes this is what we need, but often we want to add / remove one class.

There is another property for this: elem.**classList**.

This property represents an object that implements the following methods:

- elem.**classList.add/remove**("class") – add / remove class.
- elem.**classList.toggle**("class") – add a class, if not, otherwise delete.
- elem.**classList.contains**("class") – checking for a class, returns true / false.

```
let articleDiv = document.querySelector("div.article");
articleDiv.classList.remove("article"); // delete class
articleDiv.classList.add("blueStyle"); // add class
articleDiv.classList.toggle("article"); // switch class
```

softserve

Useful links

https://www.w3schools.com/js/js_htmldom.asp
http://learn.javascript.ru/document
https://metanit.com/web/javascript/8.1.php
https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-nodes.php

THANKS

softserve