

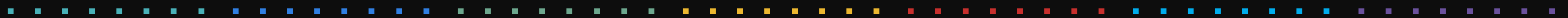
WMNFE2010 | FRONT-END DEVELOPMENT

JavaScript #8

DOM Navigation & Manipulation



ΠΕΡΙΕΧΟΜΕΝΑ



Περιεχόμενα

```
screen-reader-text:hover,
screen-reader-text:active,
screen-reader-text:focus {
  background-color: #f1f1f1;
  border-radius: 3px;
  box-shadow: 0 0 2px 2px rgb(
clip: auto !important;
color: #21759b;
display: block;
font-size: 14px;
font-size: 0.875rem;
font-weight: bold;
height: auto;
left: 5px;
line-height: normal;
padding: 15px 23px 14px;
text-decoration: none;
top: 5px;
width: auto;
z-index: 100000; /* Above w
```

Nikos Bilalis - n.bilalis@sae.edu

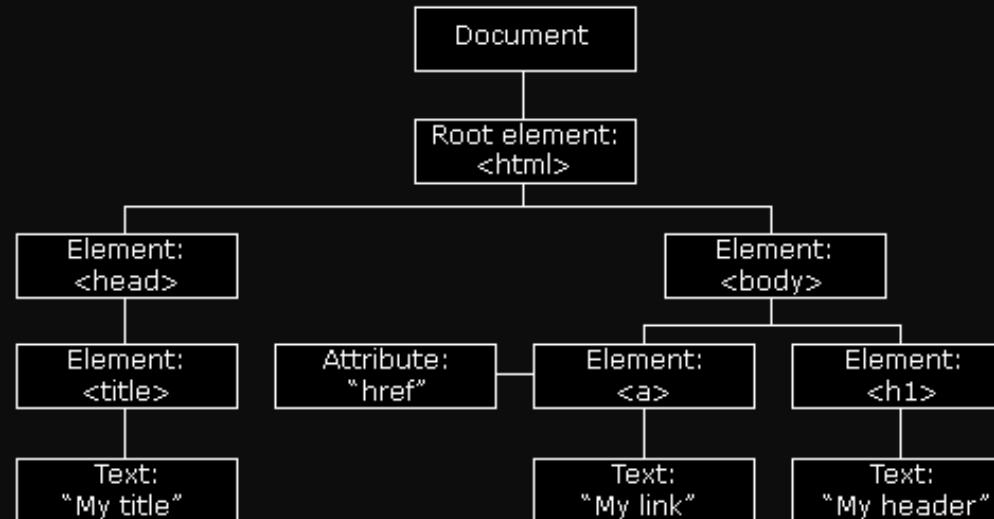
- **DOM** Navigation
 - Accessing child nodes
 - Accessing parent nodes
 - Accessing sibling nodes
- **DOM** Manipulation
 - Adding new elements
 - Accessing / setting HTML content
 - Removing existing elements
 - Replacing existing elements
 - Accessing / setting attributes

DOM

DOM

HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **Document Object Model** of the page, as a tree of elements:



DOM

HTML DOM API

The **HTML DOM API** is made up of the interfaces that define the functionality of each of the elements in **HTML**, as well as any supporting types and interfaces they rely upon.

This **API** allows JavaScript to:

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

DOM

Types of DOM Nodes

The **DOM** tree is consists of different types of nodes, such as elements, text, comments, etc. Every node has a `nodeType` property that you can use to find out what type of node you are dealing with.

| Constant | Value | Description |
|---------------------------|-------|---|
| ELEMENT_NODE | 1 | An element node such as <p> or |
| TEXT_NODE | 3 | The actual text of element. |
| COMMENT_NODE | 8 | A comment node i.e. <!-- some comment --> |
| DOCUMENT_NODE | 9 | document node i.e. the parent of <html> element. |
| DOCUMENT_TYPE_NODE | 10 | A document type node e.g. <!DOCTYPE html> for HTML5 documents. |

DOM NAVIGATION

DOM Navigation

Accessing child nodes

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**.

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const main = document.getElementById("main");
  console.log(main.firstChild.nodeName); // Prints: #text

  const hint = document.getElementById("hint");
  console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

DOM Navigation

Text / Comment nodes

First-child node of a **DIV** element might return **#text** instead of an element.

That is because whitespace, such as spaces, tabs, newlines, etc. are valid characters and they form **#text** nodes and become a part of the **DOM** tree.

To avoid the issue with **firstChild** and **lastChild** returning **#text** or **#comment** nodes, you could alternatively use the **firstElementChild** and **lastElementChild** properties to return only the first and last element node, respectively.

DOM Navigation

Παράδειγμα

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const main = document.getElementById("main");
  alert(main.firstChild.nodeName); // Outputs: H1
  main.firstChild.style.color = "red";

  const hint = document.getElementById("hint");
  alert(hint.firstChild.nodeName); // Outputs: SPAN
  hint.firstChild.style.color = "blue";
</script>
```

DOM Navigation

All Children

Similarly, you can use the **childNodes** property to access all child nodes of a given element, where the first child node is assigned index **0**.

The **childNodes** returns all child nodes, including non-element nodes like text and comment nodes.

To get a collection of only elements, use **children** property instead.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const main = document.getElementById("main");

  // First check that the element has child nodes
  if(main.childNodes()) {
    const nodes = main.childNodes;

    // Loop through node list and display node name
    for (const node in nodes) {
      alert(node.nodeName);
    }
  }
</script>
```

DOM Navigation

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const main = document.getElementById("main");

  // First check that the element has child nodes
  if(main.hasChildNodes()) {
    const nodes = main.children;

    // Loop through node list and display node name
    for (const node in nodes) {
      alert(node.nodeName);
    }
  }
</script>
```

DOM Navigation

Accessing parent nodes

You can use the **parentNode** property to access the parent of the specified node in the **DOM** tree.

The **parentNode** will always return **null** for document node, since it doesn't have a parent.

However, if you want to get only element nodes you can use the **parentElement**.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const hint = document.getElementById("hint");
  alert(hint.parentNode.nodeName); // Outputs: DIV
  alert(document.documentElement.parentNode.nodeName); // Outputs: #document
  alert(document.parentNode); // Outputs: null
</script>
```


Παράδειγμα #2

DOM Navigation

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const hint = document.getElementById("hint");
  alert(hint.parentNode.nodeName); // Outputs: DIV
  hint.parentNode.style.backgroundColor = "yellow";
</script>
```

DOM Navigation

Accessing sibling nodes

You can use the **previousSibling** and **nextSibling** properties to access the previous and next node in the **DOM tree**, respectively.

Alternatively, you can use the **previousElementSibling** and **nextElementSibling** to get the previous and next sibling element skipping any whitespace text nodes.

All these properties returns **null** if there is no such sibling.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p><hr>
</div>

<script>
  const title = document.getElementById("title");
  alert(title.previousSibling.nodeName); // Outputs: #text

  const hint = document.getElementById("hint");
  alert(hint.nextSibling.nodeName); // Outputs: HR
</script>
```

Παράδειγμα #2

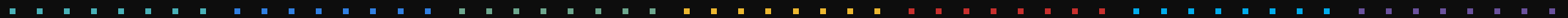
DOM Navigation

```
<div id="main">
  <h1 id="title">My Heading</h1>
  <p id="hint"><span>This is some text.</span></p>
</div>

<script>
  const hint = document.getElementById("hint");
  alert(hint.previousElementSibling.nodeName); // Outputs: H1
  alert(hint.previousElementSibling.textContent); // Outputs: My Heading

  const title = document.getElementById("title");
  alert(title.nextElementSibling.nodeName); // Outputs: P
  alert(title.nextElementSibling.textContent); // Outputs: This is some text.
</script>
```

DOM MANIPULATION



DOM Manipulation

Adding new elements

You can explicitly create new element in an **HTML** document, using the **document.createElement()** method. This method creates a new element, but it doesn't add it to the **DOM**; you'll have to do that in a separate step.

The **appendChild()** method adds the new element at the end of any other children of a specified parent node.

However, if you want to add the new element at the beginning of any other children you can use the **insertBefore()** method.

DOM Manipulation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Creating a new div element
  const newDiv = document.createElement("div");

  // Creating a text node
  const newContent = document.createTextNode("Hi, how are you doing?");

  // Adding the text node to the newly created div
  newDiv.appendChild(newContent);

  // Adding the newly created element and its content into the DOM
  const currentDiv = document.getElementById("main");
  document.body.appendChild(newDiv, currentDiv);
</script>
```

DOM Manipulation

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Creating a new div element
  const newDiv = document.createElement("div");

  // Creating a text node
  const newContent = document.createTextNode("Hi, how are you doing?");

  // Adding the text node to the newly created div
  newDiv.appendChild(newContent);

  // Adding the newly created element and its content into the DOM
  const currentDiv = document.getElementById("main");
  document.body.insertBefore(newDiv, currentDiv);
</script>
```


DOM Manipulation

Getting / setting HTML content

You can also get or set the contents of the **HTML** elements easily with the **innerHTML** property. This property sets or gets the **HTML** markup contained within the element i.e. content between its opening and closing tags.

When inserting **HTML** into a page, be careful not to use user input that hasn't been escaped, to prevent **XSS** attacks.

Mote often than not, using **textContent** is a better choice than **innerHTML**, since works with plain text and not HTML.

DOM Manipulation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Getting inner HTML contents
  const contents = document.getElementById("main").innerHTML;
  alert(contents); // Outputs inner html contents

  // Setting inner HTML contents
  const mainDiv = document.getElementById("main");
  mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
</script>
```

DOM Manipulation

Removing existing elements

Similarly, you can use the **removeChild()** method to remove a child node from the **DOM**. This method also returns the removed node.

It is also possible to remove the child element without exactly knowing the parent element. Simply find the child element and use the **parentNode** property to find its parent element.

This property returns the parent of the specified node in the **DOM** tree.

DOM Manipulation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const parentElem = document.getElementById("main");
  const childElem = document.getElementById("hint");
  parentElem.removeChild(childElem);
</script>
```

Παράδειγμα #2

DOM Manipulation

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const childElem = document.getElementById("hint");
  childElem.parentNode.removeChild(childElem);
</script>
```

DOM Manipulation

Replacing existing elements

You can also replace an element in **HTML DOM** with another using the **replaceChild()** method.

This method accepts two parameters: the node to insert and the node to be replaced.

```
parentNode.replaceChild(newChild, oldChild);
```

DOM Manipulation

Παράδειγμα

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const parentElem = document.getElementById("main");
  const oldPara = document.getElementById("hint");

  // Creating new element
  const newPara = document.createElement("p");
  const newContent = document.createTextNode("This is a new paragraph.");
  newPara.appendChild(newContent);

  // Replacing old paragraph with newly created paragraph
  parentElem.replaceChild(newPara, oldPara);
</script>
```

DOM Manipulation

Accessing / setting attributes

You can access or set the attributes of an element using the **getAttribute()** and **setAttribute()** methods.

The **getAttribute()** method of the Element interface returns the value of a specified attribute on the element.

If the given attribute does not exist, the value returned will either be **null** or "" (the empty string).

```
let attribute = element.getAttribute(attributeName);
```

Its complementary methods are: **setAttribute()**, **hasAttribute()** and **removeAttribute()**.

DOM Manipulation

Element properties

An easier way to access the attributes of an element, is directly as properties of the element.

Most attributes have a corresponding attribute with the exact same name.

If the attribute's name includes *hyphens*, then the property name is the *camel-cased* version of that name.

The **class** attribute is a special case. Since it is a reserved word, the property name is **className**.

Παράδειγμα

DOM Manipulation

```
<div id="main" class="main">
  <h1>Hello World!</h1>
  <p>
    <a href="http://www.example.com">
      
    </a>
  </p>
</div>

<script>
  const div = document.getElementById('main');
  const a = div.querySelector('a');
  const img = div.querySelector('img');

  console.log(div.className);
  console.log(a.href);
  console.log(img.src);

  a.target = '_blank';
  img.alt = 'Affiliate link';
</script>
```

DOM Manipulation

style property

All rules in the **style** attribute, like **property: value;**, become "*sub-properties*" of the **style** property.

```
el.style.color = 'red';
el.style.backgroundColor = 'blue';
```

The same rule, for names with *hyphens*, applies here too.

DOM Manipulation

classList property

Although it is possible to alter styles through the **style** property, or access the **className** property directly, it is more convenient to use the **classList** property.

This property is a **DOMTokenList** object, which is a list of space-separated **CSS** classes.

You can add, remove or toggle classes using the **add()**, **remove()** and **toggle()** methods respectively, without having to worry about the existence / duplicity of class names, or splitting and concatenating them.

Παράδειγμα

DOM Manipulation

```
<div id="main" class="black-bg">
  <h1>Hello World!</h1>
</div>

<script>
  const el = document.getElementById('main');
  el.classList.remove('black-bg');
  el.classList.add('main');

  // Poor man's version of dark mode.
  // The class name is added to the list,
  // when the second argument is `true` / `truthy`
  // otherwise it is removed, if present.
  el.classList.toggle('dark', (new Date()).getHours() > 18);
</script>
```


Χρήσιμα links

 JavaScript HTML DOM


https://www.w3schools.com/js/js_htmldom.asp

 JavaScript DOM Navigation - Tutorial Republic

<https://www.tutorialrepublic.com/javascript-tutorial/j...>

 Node.textContent - Web APIs | MDN


<https://developer.mozilla.org/en-US/docs/Web/API/N...>

 What's Best: innerText vs. innerHTML vs. textContent | by Annie Mester | Better Programming

<https://betterprogramming.pub/whats-best-innertext-...>

 Element.getAttribute() - Web APIs | MDN

<https://developer.mozilla.org/en-US/docs/Web/API/El...>

 Element.classList - Web APIs | MDN

<https://developer.mozilla.org/en-US/docs/Web/API/El...>

Extra info

🔗 How to Manipulate the DOM - the Ultimate Beginner's Guide

<https://www.freecodecamp.org/news/how-to-manip...>

🔗 What is the Difference Between textContents, innerText, and innerHtml?

<https://www.microfocus.com/documentation/silk-test...>

THANK YOU!