**Лабораторная работа №3**

*Задачи на работу с DOM на JavaScript*

### Задание 1. Работа с классами

* Дан элемент **#elem**. Добавьте ему класс **www**.
* Дан элемент **#elem**. Удалите у него класс **www**.
* Дан элемент **#elem**. Проверьте наличие у него класса **www**.
* Дан элемент **#elem**. Добавьте ему класс **www**, если его нет и удалите - если есть.
* Дан элемент **#elem**. Узнайте количество его классов.
* Дан элемент **#elem**. Выведите последовательно алертом его классы.

### Задание 2. Работа с CSS

Дан элемент **#elem**. Сделайте его красного цвета, размером 30px, добавьте ему границу. Решите задачу с помощью свойства **cssText**.

### Задание 3. Свойство tagName

* Дан элемент **#elem**. По клику на него выведите название его тега.
* Дан элемент **#elem**. По клику на него выведите название его тега *в нижнем регистре*.
* Даны элементы с классом **www**. Добавьте каждому элементу в конец название его тега в нижнем регистре.

### Задание 4. Вставка элементов через appendChild

* Дан **ol**. Вставьте ему в конец **li** с текстом 'пункт'.
* Дан **ul**. Дан массив. Вставьте элементы этого массива в конец **ul** так, чтобы каждый элемент стоял в своем **li**.

### Задание 5. Привязывание событий при вставке

Дан **ul**. Дан массив. Вставьте элементы этого массива в конец **ul** так, чтобы каждый элемент стоял в своем **li**. Сделайте так, чтобы к вставляемым li было привязано следующее событие: по нажатию на li она должна вывести на экран свой текст.

### Задание 6. Вставка элементов через insertBefore

Дан элемент **ul**, а в нем li **#elem**. Вставьте перед элементом **#elem** новую **li** с текстом '!!!'.

### Задание 7. Вставка элементов через insertAdjacentHTML

* Дан элемент **#elem**. Вставьте перед ним **span** с текстом '!!!'.
* Дан элемент **#elem**. Вставьте после него **span** с текстом '!!!'.
* Дан элемент **#elem**. Вставьте ему в начало **span** с текстом '!!!'.
* Дан элемент **#elem**. Вставьте ему в конец **span** с текстом '!!!'.

### Задание 8. Потомки

* Дан элемент **#elem**. Найдите **первого** потомка этого элемента и сделайте его текст красного цвета.
* Дан элемент **#elem**. Найдите **последнего** потомка этого элемента и сделайте его текст красного цвета.
* Дан элемент **#elem**. Найдите **всех** потомков этого элемента и добавьте им в конец текст '!'.

### Задание 9. Соседи

* Дан элемент **#elem**. Найдите его соседа сверху и добавьте ему в конец текст '!'.
* Дан элемент **#elem**. Найдите его соседа снизу и добавьте ему в конец текст '!'.
* Дан элемент **#elem**. Найдите его соседа снизу его соседа снизу (следующий элемент за соседним) и добавьте ему в конец текст '!'.

### Задание 10. Родители

* Дан элемент **#elem**. Найдите его родителя и покрасьте его в красный цвет.
* Дан элемент **#elem**. Найдите родителя его родителя и покрасьте его в красный цвет.

### Задание 11. Удаление и клонирование

* Дан элемент **#parent**, внутри него дан элемент **#child**. Дана кнопка. По нажатию на эту кнопку удалите элемент **#child**.
* Дан **ol**. По нажатию на кнопку получите его последнего потомка и удалите его.
* Дан элемент. Сделайте так, чтобы по нажатию по нему этот элемент удалялся.
* Дан **ol**, а внутри него **li**. Сделайте так, чтобы по нажатию на любую **li** эта **li** удалялась.
* Дан инпут. Дана кнопка. По нажатию на кнопку клонируйте этот инпут.

### Задание 12. Практика (по вариантам)

1. Дан массив. Создайте **ul** через createElement, затем вставьте каждый элемент этого массива в отдельную **li** внутри этой **ul**, затем вставьте эту **ul** в конец **body**.
2. Дан инпут. Рядом с ним находится кнопочка "+". По нажатию на эту кнопку под нашим инпутом должен появится еще один пустой инпут.
3. Дан инпут. В него вводится число. По потери фокуса сделайте так, чтобы каждая цифра вставилась в новый инпут. Инпутов для цифр изначально не существует, они должны создаться в процессе работы скрипта.
4. Дана кнопка. Сделайте так, чтобы по нажатию на эту кнопку, скрывался родитель этой кнопки.

**Отчет по лабораторной работе**

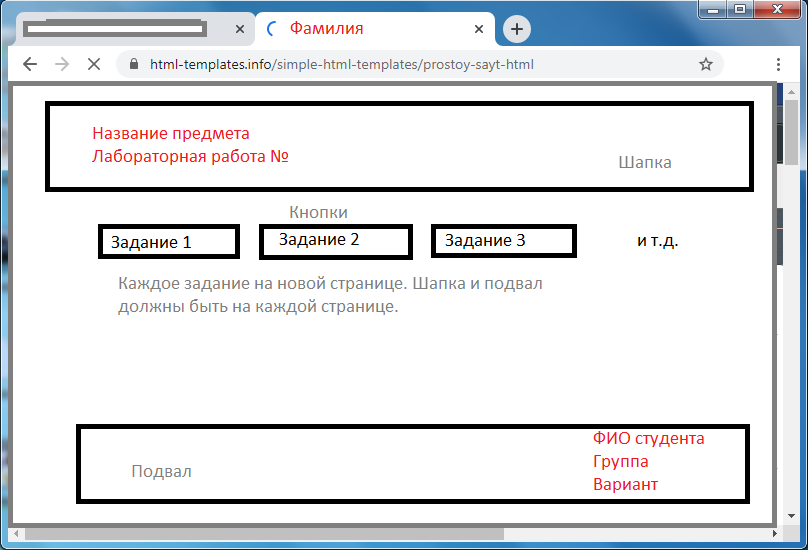
В соответствии со структурой заготовки отчета и примером оформления оформить в отчете все задания, выполняемые в ходе лабораторной работы, а также индивидуальные задания по вариантам. Файл с отчетом называть по шаблону: Фамилия\_лаб\_раб\_номер.

Отчет предоставляется в электронном виде либо лично преподавателю, либо на электронную почту для проверки. Также по результатам лабораторной работы на следующем за ней занятии проводится выборочный опрос по командам языка.

Пример страницы:

Должен быть фон страницы, использованы стили.

Стилизация у каждого студента должна быть разной. Если стилизация повторяется, она будет засчитана студенту, который сдаст ЛР раньше.



**Теория и примеры**

**Работа с classList**

Свойство **classList** содержит [псевдомассив](http://code.mu/ru/javascript/manual/page/pseudo-array/) CSS классов элемента, а также позволяет добавлять и удалять классы элемента, проверять наличие определенного класса среди классов элемента.

Речь идет об атрибуте [class](http://code.mu/ru/markup/manual/html/attr/class/), внутри которого можно писать несколько классов через пробел, например **www ggg zzz**. С помощью **classList** можно удалить, к примеру, класс **ggg**, не затронув остальные классы.

## Синтаксис

элемент.classList

## Пример 1. Количество классов

Узнаем количество классов элемента:

<p id="elem" class="www ggg zzz"></p>let elem = document.querySelector('#elem');

let length = elem.classList.length;

console.log(length);

Результат выполнения кода:

3

## Пример 2. Перебираем классы

Выведем столбец классов элемента:

<p id="elem" class="www ggg zzz"></p>let elem = document.querySelector('#elem');

let classNames = elem.classList;

for (let className of classNames) {

document.write(className + '<br>');

}

Результат выполнения кода:

www  
ggg  
zzz

# **Свойство cssText**

Свойство **cssText** позволяет задать CSS стили массово одной строкой. При этом все содержимое атрибута [style](http://code.mu/ru/markup/manual/html/attr/style/) перезаписывается.

*Пожалуйста, не злоупотребляйте этим свойством. Не стоит его использовать везде - это ведет к ошибкам - ведь cssText затирает все из атрибута style, это значит, что все ранее установленные через JavaScript стили CSS просто затрутся.*

## Синтаксис

элемент.style.cssText = 'свойство 1: значение; свойство 2: значение...'

## Пример

Давайте зададим элементу несколько стилей:

<p id="elem"></p>let elem = document.querySelector('#elem');

elem.style.cssText = 'color: red; font-size: 40px;';

## Пример

В этом примере у элемента изначально уже будут стили в атрибуте [style](http://code.mu/ru/markup/manual/html/attr/style/), но свойство cssText перезапишет его:

<p id="elem" style="background: red;"></p>let elem = document.querySelector('#elem');

elem.style.cssText = 'color: red; font-size: 20px;';

## Пример

Чтобы предыдущие стили не перезаписывались, можно сделать так:

<p id="elem" style="background: green;"></p>let elem = document.querySelector('#elem');

elem.style.cssText += 'color: red; font-size: 20px;';

# **Свойство tagName**

Свойство **tagName** содержит имя тега в верхнем регистре (большими буквами).

## Синтаксис

элемент.tagName

## Пример

Давайте получим элемент **#elem** и выведем название его тега:

<div id="elem"></div>let elem = document.getElementById('elem');

console.log(elem.tagName);

Результат выполнения кода:

'DIV'

## Пример

Давайте выведем название тега в нижнем регистре. Для этого воспользуемся методом [toLowerCase](http://code.mu/ru/javascript/manual/string/toLowerCase/):

<div id="elem"></div>let elem = document.getElementById('elem');

console.log(elem.tagName.toLowerCase());

Результат выполнения кода:

'div'

# **Метод createElement**

Метод **createElement** позволяет создать новый элемент, передав в параметре имя тега. После создания с элементом можно работать как с обычным элементом, а также его можно добавить на страницу методами [prepend](http://code.mu/ru/javascript/manual/dom/prepend/), [append](http://code.mu/ru/javascript/manual/dom/append/), [appendChild](http://code.mu/ru/javascript/manual/dom/appendChild/), [insertBefore](http://code.mu/ru/javascript/manual/dom/insertBefore/) или [insertAdjacentElement](http://code.mu/ru/javascript/manual/dom/insertAdjacentElement/).

Если записать результат работы createElement в переменную, то в этой переменной будет такой элемент, как будто бы мы получили его через [querySelector](http://code.mu/ru/javascript/manual/dom/querySelector/) или [getElementById](http://code.mu/ru/javascript/manual/dom/getElementById/).

Единственное отличие - наш элемент не будет размещен на странице. А так мы можем менять ему innerHTML, атрибуты, навешивать обработчики событий и в конце концов разместить его на странице.

## Синтаксис

document.createElement('имя тега')

## Пример

Давайте создадим абзац, установим ему текст и поместим на страницу в конец блока **#parent**:

<div id="parent">

<p>1</p>

<p>2</p>

<p>3</p>

</div>let parent = document.querySelector('#parent');

let p = document.createElement('p');

p.innerHTML = '!';

parent.appendChild(p);

Результат выполнения кода:

<div id="parent">

<p>1</p>

<p>2</p>

<p>3</p>

<p>!</p>

</div>

## Пример

Дан [ul](http://code.mu/ru/markup/manual/html/tag/ul/). Давайте разместим в нем 9 тегов [li](http://code.mu/ru/markup/manual/html/tag/li/), при этом их текстом сделаем порядковые номера:

<ul id="parent"></ul>let parent = document.querySelector('#parent');

for (let i = 1; i <= 9; i++) {

let li = document.createElement('li');

li.innerHTML = i;

parent.appendChild(li);

}

Результат выполнения кода:

<ul id="parent">

<li>1</li>

<li>2</li>

<li>3</li>

<li>4</li>

<li>5</li>

<li>6</li>

<li>7</li>

<li>8</li>

<li>9</li>

</ul>

## Пример

Давайте при вставке элементов будем привязывать к ним обработчики событий:

<ul id="parent"></ul>let parent = document.querySelector('#parent');

for (let i = 1; i <= 9; i++) {

let li = document.createElement('li');

li.innerHTML = i;

li.addEventListener('click', function() {

alert(this.innerHTML);

});

parent.appendChild(li);

}

Результат выполнения кода:

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

# **Метод appendChild**

Метод **appendChild** позволяет вставить в конец какого-либо другой элемент. Чаще всего используется после создания элемента с помощью [createElement](http://code.mu/ru/javascript/manual/dom/createElement/).

## Синтаксис

родитель.appendChild(элемент)

## Пример

Давайте создадим абзац, установим ему текст и поместим на страницу в конец блока **#parent**:

<div id="parent">

<p>1</p>

<p>2</p>

<p>3</p>

</div>let parent = document.querySelector('#parent');

let p = document.createElement('p');

p.innerHTML = '!';

parent.appendChild(p);

Результат выполнения кода:

<div id="parent">

<p>1</p>

<p>2</p>

<p>3</p>

<p>!</p>

</div>

## Пример

Дан [ul](http://code.mu/ru/markup/manual/html/tag/ul/). Давайте разместим в нем 9 тегов [li](http://code.mu/ru/markup/manual/html/tag/li/), при этом их текстом сделаем порядковые номера:

<ul id="parent"></ul>let parent = document.querySelector('#parent');

for (let i = 1; i <= 9; i++) {

let li = document.createElement('li');

li.innerHTML = i;

parent.appendChild(li);

}

Результат выполнения кода:

<ul id="parent">

<li>1</li>

<li>2</li>

<li>3</li>

<li>4</li>

<li>5</li>

<li>6</li>

<li>7</li>

<li>8</li>

<li>9</li>

</ul>

## Пример

Давайте заполним таблицу [tr-ками](http://code.mu/ru/markup/manual/html/tag/tr/) и [td-шками](http://code.mu/ru/markup/manual/html/tag/td/):

<table id="table"></table>let parent = document.querySelector('#parent');

for (let i = 1; i <= 3; i++) {

let tr = document.createElement('tr'); // создаем tr-ку

// Заполняем tr-ку td-шками:

for (let j = 1; j <= 3; j++) {

let td = document.createElement('td'); // создаем td-шку

td.innerHTML = j; // пишем в нее текст

tr.appendChild(td); // добавляем созданную td-шку в конец tr-ки

}

table.appendChild(tr); // добавляем созданную tr-ку в конец таблицы

}

Результат выполнения кода:

<table id="table">

<tr>

<td>1</td>

<td>2</td>

<td>3</td>

</tr>

<tr>

<td>1</td>

<td>2</td>

<td>3</td>

</tr>

<tr>

<td>1</td>

<td>2</td>

<td>3</td>

</tr>

</table>

# **Метод insertBefore**

Метод **insertBefore** позволяет вставить элемент перед другим элементом. Чаще всего используется после создания элемента с помощью [createElement](http://code.mu/ru/javascript/manual/dom/createElement/). Метод применяется к родителю того элемента, перед которым произойдет вставка.

## Синтаксис

родитель.insertBefore(элемент, перед кем вставить)

## Пример

Создадим абзац и поместим его перед вторым абзацем:

<div id="parent">

<p>elem 1</p>

<p id="before">elem 2</p>

<p>elem 3</p>

</div>let parent = document.querySelector('#parent');

let before = document.querySelector('#before');

let p = document.createElement('p');

p.innerHTML = '!';

parent.insertBefore(p, before);

Результат выполнения кода:

<div id="parent">

<p>elem 1</p>

<p>!</p>

<p>elem 2</p>

<p>elem 3</p>

</div>

## Пример

Добавим абзац в начало элемента **#parent**. Для этого вставим наш абзац перед первым потомком **#parent**. Этого потомка можно найти с помощью [firstElementChild](http://code.mu/ru/javascript/manual/dom/firstElementChild/):

<div id="parent">

<p>elem 1</p>

<p>elem 2</p>

</div>let parent = document.querySelector('#parent');

let p = document.createElement('p');

p.innerHTML = '!';

parent.insertBefore(p, parent.firstElementChild);

Результат выполнения кода:

<div id="parent">

<p>!</p>

<p>elem 1</p>

<p>elem 2</p>

</div>

## Пример

При передаче вторым параметром null метод **insertBefore** срабатывает как [appendChild](http://code.mu/ru/javascript/manual/dom/appendChild/). В то же время, если в элементе нет дочерних элементов, **firstElementChild** возвращает null. Следовательно, добавлять в начало элемента можно даже тогда, когда в нем нет дочерних элементов:

<div id="parent"></div>let parent = document.querySelector('#parent');

let p = document.createElement('p');

p.innerHTML = '!';

parent.insertBefore(p, parent.firstChild);

Результат выполнения кода:

<div id="parent">

<p>!</p>

</div>

# **Метод insertAdjacentHTML**

Метод **insertAdjacentHTML** позволяет вставить строку HTML кода в любое место страницы. Код вставляется относительно *опорного элемента*. Можно сделать вставку перед опорным элементом (способ вставки **beforeBegin**), после него (способ вставки **afterEnd**), а также в начало (способ вставки **afterBegin**) или в конец (способ вставки **beforeEnd**) опорного элемента.

## Синтаксис

опорный элемент.insertAdjacentHTML(способ вставки, код для вставки)

## Пример . Способ beforeBegin

Пусть опорный элемент - это элемент **#target**. Вставим перед ним новый абзац:

<div id="target">

<p>elem</p>

</div>let target = document.querySelector('#target');

target.insertAdjacentHTML('beforeBegin', '<p>!</p>');

Результат выполнения кода:

<p>!</p>

<div id="target">

<p>elem</p>

</div>

## Пример . Способ afterEnd

А теперь вставим новый абзац после опорного элемента:

<div id="target">

<p>elem</p>

</div>let target = document.querySelector('#target');

target.insertAdjacentHTML('afterEnd', '<p>!</p>');

Результат выполнения кода:

<div id="target">

<p>elem</p>

</div>

<p>!</p>

## Пример . Способ afterBegin

Вставим новый абзац в начало опорного элемента:

<div id="target">

<p>elem</p>

</div>let target = document.querySelector('#target');

target.insertAdjacentHTML('afterBegin', '<p>!</p>');

Результат выполнения кода:

<div id="target">

<p>!</p>

<p>elem</p>

</div>

## Пример . Способ beforeEnd

Вставим новый абзац в конец опорного элемента:

<div id="target">

<p>elem</p>

</div>let target = document.querySelector('#target');

target.insertAdjacentHTML('beforeEnd', '<p>!</p>');

Результат выполнения кода:

<div id="target">

<p>elem</p>

<p>!</p>

</div>

# **Свойство firstElementChild**

Свойство **firstElementChild** содержит первый дочерний элемент. Дочерними элементами считаются все теги, которые непосредственно расположены внутри блока. Если у элемента нет дочерних элементов - возвращается **null**.

## Синтаксис

элемент.firstElementChild

## Пример

Получим содержимое первого потомка элемента:

<div id="parent">

<p>1</p>

<p>2</p>

</div>let parent = document.querySelector('#parent');

let text = parent.firstElementChild.innerHTML;

console.log(text);

Результат выполнения кода:

'1'

## Пример

А теперь у элемента нет дочерних элементов и поэтому выведется **null**:

<div id="parent"></div>let parent = document.querySelector('#parent');

console.log(parent.firstElementChild);

Результат выполнения кода:

null

# **Свойство lastElementChild**

Свойство **lastElementChild** хранит в себе последний дочерний элемент. Дочерними элементами считаются все теги, которые непосредственно расположены внутри блока. Если у элемента нет дочерних элементов - возвращается **null**.

## Синтаксис

элемент.lastElementChildChild

## Пример

Получим содержимое последнего потомка элемента:

<div id="parent">

<p>1</p>

<p>2</p>

</div>let parent = document.querySelector('#parent');

let text = parent.lastElementChild.innerHTML;

console.log(text);

Результат выполнения кода:

'2'

## Пример

А теперь у элемента нет дочерних элементов и поэтому выведется **null**:

<div id="parent"></div>let parent = document.querySelector('#parent');

console.log(parent.lastElementChild);

Результат выполнения кода:

null

# **Свойство children**

Свойство **children** хранит в себе [псевдомассив](http://code.mu/ru/javascript/manual/page/pseudo-array/) дочерних элементов. Дочерними элементами считаются все теги, которые непосредственно расположены внутри блока.

## Синтаксис

элемент.children

## Пример

Давайте переберем в цикле всех потомков элемента и выведем их содержимое:

<div id="parent">

<p>1</p>

<p>2</p>

<p>3</p>

<p>4</p>

<p>5</p>

</div>let parent = document.querySelector('#parent');

let elems = parent.children;

for (let elem of elems) {

console.log(elem.innerHTML);

}

# **Свойство previousElementSibling**

Свойство **previousElementSibling** содержит предыдущий элемент, находящийся в этом же родителе. Если такого элемента нет - возвращается **null**.

## Синтаксис

элемент.previousElementSibling

## Пример

Дан элемент **#elem**. Давайте выведем текст его соседа сверху:

<p>sibling</p>

<p id="elem">elem</p>let elem = document.querySelector('#elem');

let text = elem.previousElementSibling.innerHTML;

console.log(text);

Результат выполнения кода:

'sibling'

## Пример

Если соседа сверху нет или он расположен не в родителе нашего элемента, также возвращается null:

<p>sibling</p>

<div>

<p id="elem">elem</p>

</div>let elem = document.querySelector('#elem');

console.log(elem.previousElementSibling);

Результат выполнения кода:

null

# **Свойство parentElement**

Свойство **parentElement** содержит родительский элемент.

## Синтаксис

элемент.parentElement

## Пример

Давайте получим родителя элемента **#elem** и выведем на экран его **id**:

<div id="parent">

<p id="elem"></p>

</div>let elem = document.querySelector('#elem');

let id = elem.parentElement.id;

console.log(id);

Результат выполнения кода:

'parent'

# **Свойство parentNode**

Свойство **parentNode** содержит родительский элемент.

Существует также почти идентичное свойство [parentElement](http://code.mu/ru/javascript/manual/dom/parentElement/). Отличия: для тега [html](http://code.mu/ru/markup/manual/html/tag/html/) свойство parentNode возвращает **document**, а parentElement возвращает null.

## Синтаксис

элемент.parentNode

## Пример

Давайте получим родителя элемента **#elem** и выведем на экран его **id**:

<div id="parent">

<p id="elem"></p>

</div>let elem = document.querySelector('#elem');

let id = elem.parentNode.id;

console.log(id);

Результат выполнения кода:

'parent'

# **Метод cloneNode**

Метод **cloneNode** позволяет клонировать элемент и получить его точную копию. Эту копию затем можно вставить на страницу с помощью методов [prepend](http://code.mu/ru/javascript/manual/dom/prepend/), [append](http://code.mu/ru/javascript/manual/dom/append/), [appendChild](http://code.mu/ru/javascript/manual/dom/appendChild/), [insertBefore](http://code.mu/ru/javascript/manual/dom/insertBefore/) или [insertAdjacentElement](http://code.mu/ru/javascript/manual/dom/insertAdjacentElement/).

В параметре метод получает true либо false. Если передан true, то элемент клонируется полностью, вместе со всем атрибутами и дочерними элементами, а если false - только сам элемент (без дочерних элементов).

## Синтаксис

элемент.cloneNode(true или false);

## Пример

Сделаем копию блока с классом **elem** и вставим его в конец блока **#parent**:

<div id="parent">

<div class="elem">

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

</div>let parent = document.getElementById('parent');

let elem = parent.querySelector('.elem');

let clone = elem.cloneNode(true);

parent.appendChild(clone);

Результат выполнения кода:

<div id="parent">

<div class="elem">

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

<div class="elem">

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

</div>

## Пример

С полученным клоном можно работать как с обычным элементом:

<div id="parent">

<div class="elem">

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

</div>let parent = document.getElementById('parent');

let elem = parent.querySelector('.elem');

let clone = elem.cloneNode(true);

clone.children[0].innerHTML = 'Новое содержимое первого абзаца';

clone.children[1].innerHTML = 'Новое содержимое второго абзаца';

parent.appendChild(clone);

Результат выполнения кода:

<div id="parent">

<div class="elem">

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

<div class="elem">

<p>Новое содержимое первого абзаца</p>

<p>Новое содержимое второго абзаца</p>

</div>

</div>