Лабораторная работа №7

Тема работы: математические функции JavaScript

Цель работы: научиться решать задачи на математические функции JavaScript

Порядок выполнения работы
Часть I

Задание 1.
Работа с % (выполнять все задания)

1. Даны переменные **a** и **b**, вводятся пользователем. Найдите остаток от деления **a** на **b**.

2. Даны переменные **a** и **b,** вводятся пользователем. Проверьте, что **a** делится без остатка на **b**. Если это так - выведите **'Делится'** и результат деления, иначе выведите **'Делится с остатком'** и остаток от деления.

Работа со степенью и корнем (выполнять все задания)

Для решения задач данного блока вам понадобятся следующие методы: Math.pow, Math.sqrt.

1. Возведите **N** в **(21-N)** степень. Результат запишите в переменную **st**. N - ваш вариант.

2. Найдите квадратный корень из **a**, вводится пользователем.

3. Дан массив. Найдите **квадратный корень** из **суммы кубов** его элементов. Для решения воспользуйтесь циклом **for**.

Работа с функциями округления (выполнять все задания)

Для решения задач данного блока вам понадобятся следующие функции: Math.round, Math.ceil, Math.floor, toFixed, toPrecision.

1. Найдите квадратный корень из **111*N**. Результат округлите до целых, до десятых, до сотых. N - ваш вариант.

2. Найдите квадратный корень из **a,** вводится пользователем. Округлите результат в **большую** и **меньшую** стороны, запишите результаты округления в **объект** с ключами **'floor'** и **'ceil'**. Выведите результат.

Нахождение максимального и минимального числа

Для решения задач данного блока вам понадобятся следующие методы: Math.max, Math.min.

1. Дан массив. Найдите **минимальное** и **максимальное** число. Выведите результат.

Работа с рандомом (выполнять все задания)

Для решения задач данного блока вам понадобятся следующие методы: Math.random.

1. Выведите на экран случайное целое число от **1** до **10*N**. N - ваш вариант.

2. Заполните массив (**N+5)-м количеством случайных** целых чисел. Выведите результат. (*Подсказка: нужно воспользоваться циклами for или while*).

Работа с модулем

Для решения задач данного блока вам понадобятся следующие методы: Math.abs.

1. Даны переменные **a** и **b,** вводятся пользователем. Найдите модуль разности **a** и **b**.

Задание 2. Потомки (по вариантам)

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

4. Дан элемент **#elem**. Найдите **первого** потомка этого элемента и сделайте его текст зеленого цвета.
5. Дан элемент **#elem**. Найдите **последнего** потомка этого элемента и сделайте его текст зеленого цвета.
6. Дан элемент **#elem**. Найдите **всех** потомков этого элемента и добавьте им в начало текст '!'.
7. Дан элемент <**div .cl**>. Найдите **первого** потомка этого элемента и сделайте его текст синего цвета.
8. Дан элемент <**div .cl**>. Найдите **последнего** потомка этого элемента и сделайте его текст синего цвета.
9. Дан элемент <**div .cl**>. Найдите **всех** потомков этого элемента и добавьте им в конец текст '!!!'.
10. Дан элемент **.ttt**. Найдите **первого** потомка этого элемента и сделайте его текст желтого цвета.

Задание 3. Соседи (по вариантам)

1. Дан элемент <**p .el**>. Найдите его соседа сверху и добавьте ему в конец текст '!'.
2. Дан элемент <**p .el**>. Найдите его соседа снизу и добавьте ему в конец текст '!'.
3. Дан элемент <**p .el**>. Найдите его соседа снизу его соседа снизу (следующий элемент за соседним) и добавьте ему в конец текст '!'.
4. Дан элемент **.el**. Найдите его соседа сверху и добавьте ему в начало текст '!'.
5. Дан элемент **.el**. Найдите его соседа снизу и добавьте ему в начало текст '!'.
6. Дан элемент **.el**. Найдите его соседа снизу его соседа снизу (следующий элемент за соседним) и добавьте ему в начало текст '!'.
7. Дан элемент **#elem**. Найдите его соседа сверху и добавьте ему в конец тот же текст, который уже есть в данном элементе.
8. Дан элемент **#elem**. Найдите его соседа снизу и добавьте ему в конец тот же текст, который уже есть в данном элементе.
9. Дан элемент **#elem**. Найдите его соседа снизу его соседа снизу (следующий элемент за соседним) и добавьте ему в конец тот же текст, который уже есть в данном элементе.

10. Дан элемент **<div .el>**. Найдите его соседа сверху и добавьте ему в конец текст '!!!'.

Задание 4. Родители (по вариантам)

1. Дан элемент **#elem**. Найдите его родителя и покрасьте его в синий цвет.
2. Дан элемент **#elem**. Найдите родителя его родителя и покрасьте его в синий цвет.
3. Дан элемент **.el**. Найдите его родителя и покрасьте его в красный цвет.
4. Дан элемент **.el**. Найдите родителя его родителя и покрасьте его в красный цвет.
5. Дан элемент **#el**. Найдите его родителя и увеличьте его размер.
6. Дан элемент **#el**. Найдите родителя его родителя и увеличьте его размер.
7. Дан элемент **.elem**. Найдите его родителя и сделайте в нем текст курсивом.
8. Дан элемент **.elem**. Найдите родителя его родителя и сделайте в нем текст жирным.
9. Дан элемент **#el1**. Найдите его родителя и измените его текст на вашу фамилию.
10. Дан элемент **#el1**. Найдите родителя его родителя и измените его текст на вашу фамилию

Задание 5. Удаление и клонирование (выполнять все задания)

- 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.

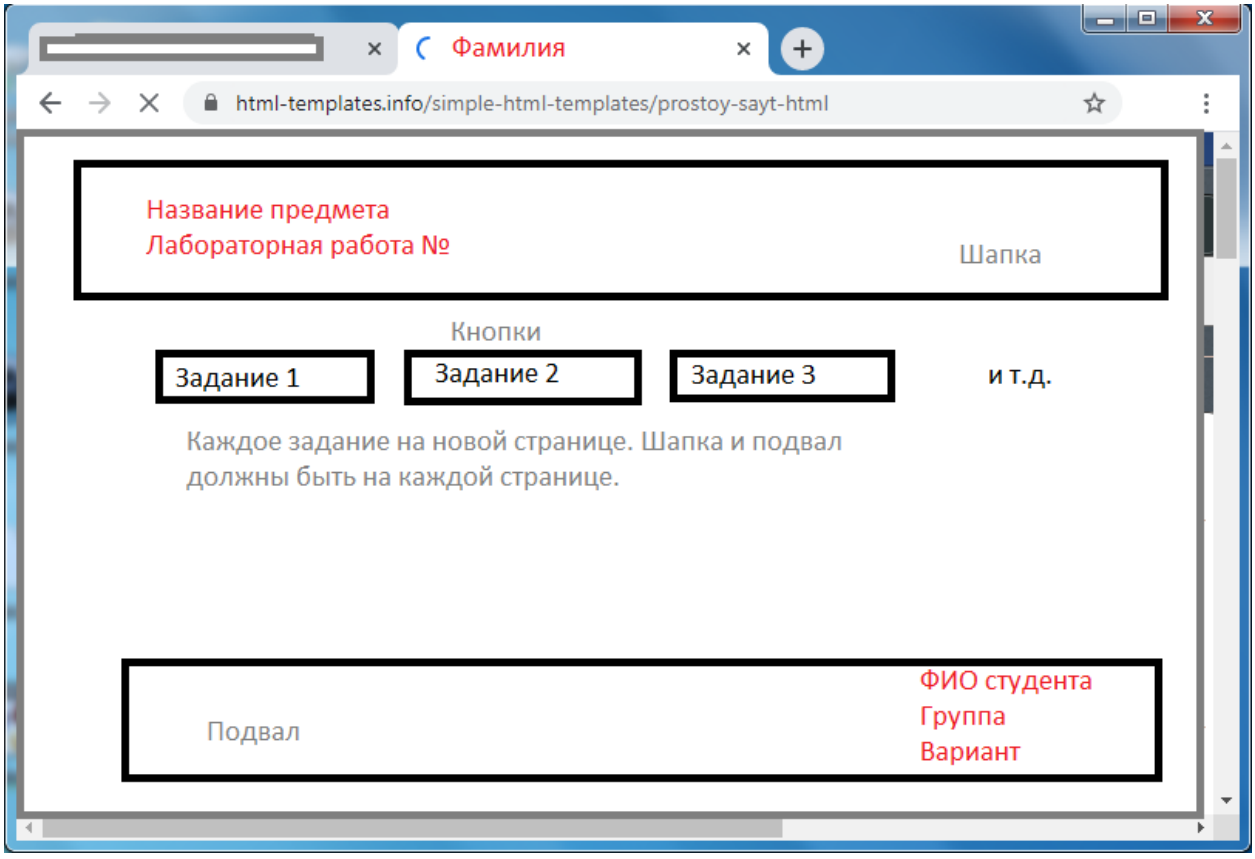
Задание 6. Практика (по вариантам)

1) Дан массив. Создайте **ol** через createElement, затем вставьте каждый элемент этого массива в отдельную **li** внутри этой **ol**, затем вставьте эту **ol** в конец **body**.
2) Дан инпут. Рядом с ним находится кнопочки "+" и "-". По нажатию на эти кнопки под нашим инпутом должен появится еще один пустой инпут либо удалиться последний соответственно.
3) Дан инпут. В него вводится текст. По потери фокуса сделайте так, чтобы каждый символ вставлялся в новый инпут. Инпутов для символов изначально не существует, они должны создаться в процессе работы скрипта.
4) Дана кнопка. Сделайте так, чтобы по нажатию на эту кнопку, скрывался (всё, что в нем находится) перед этой кнопкой.
5) 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**.
6) Дан инпут. Рядом с ним находится кнопочка "+". По нажатию на эту кнопку под нашим инпутом должен появится еще один пустой инпут.
7) 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.
8) Дана кнопка. Сделайте так, чтобы по нажатию на эту кнопку, скрывался родитель этой кнопки.
9) Дан объект. Создайте **ul** через createElement, затем вставьте каждый элемент этого объекта в отдельную **li** внутри этой **ul**, затем вставьте эту **ul** в конец **body**.
10) Дан список **ol**. Рядом с ним находится кнопочка "+". По нажатию на эту кнопку в нашем списке должен появится еще один пункт с текстом.

Структура вашей 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 CSS классов элемента, а также позволяет добавлять и удалять классы элемента, проверять наличие определенного класса среди классов элемента.

Речь идет об атрибуте 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 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, 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:

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:

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, append, appendChild, insertBefore или insertAdjacentElement .

Если записать результат работы createElement в переменную, то в этой переменной будет такой элемент, как будто бы мы получили его через querySelector или 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. Давайте разместим в нем 9 тегов 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.

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. Давайте разместим в нем 9 тегов 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 и td-шками:

```
<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. 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:

```
<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. 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**:

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:

    1
    2
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**:

let parent = document.querySelector('#parent');
console.log(parent.lastElementChild);

Результат выполнения кода:

```
null
```

Свойство children

Свойство **children** хранит в себе псевдомассив 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. Отличия: для тега 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, append, appendChild, insertBefore или 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>
```