**Web-дизайн**

Лабораторная работа №9

**Взаимодействие с веб-страницей в JavaScript**

**Таймеры**

**Функция setTimeout**

Для одноразового выполнения действий через промежуток времени предназначена функция **setTimeout**(). Она может принимать два параметра:

var timerId = setTimeout(someFunction, period)

Параметр **period** указывает на промежуток, через который будет выполняться функция из параметра **someFunction**. А в качестве результата функция возвращает *id* таймера.

function timerFunction() {

    document.write("выполнение функции setTimeout");

}

setTimeout(timerFunction, 3000); // выполнится через 3 секунды

Для остановки таймера применяется функция **clearTimeout**().

**Рекурсия с помощью setTimeout**

Хотя метод **setTimeout** не предназначен для создания итерационных таймеров, однако их все равно можно делать с ее помощью, если воспользоваться рекурсией.

function *timer*() {

var elem = document.getElementById('test');

elem.value = parseInt(elem.value)+1;

window.**setTimeout**(*timer*, 1000); //рекурсия

}

*timer*(); // запуск рекурсии таймера

**Функция setInterval**

Функции **setInterval**() и **clearInterval**() работают аналогично функциям *setTimeout*() и *clearTimeout*(), но **setInterval**() постоянно выполняет функцию через промежуток времени.

<div id="time"></div>

<script>

function updateTime() {

   document.getElementById("time").innerHTML = new Date().toTimeString();

}

setInterval(updateTime, 1000); // выполняет каждую секунду

</script>

**Функция requestAnimationFrame()**

Метод **requestAnimationFrame**() – это специализированная функция, созданная для эффективного запуска анимации в браузере. По сути, это современная версия *setInterval*() – она выполняет указанный блок кода до того, как браузер перерисовывает изображение, позволяя запускать анимацию с подходящей частотой кадров независимо от среды, в которой она выполняется.

Он был создан в ответ на проблемы с *setInterval*(), который, например, не работает с частотой кадров, оптимизированной для устройства, иногда пропускает кадры, продолжает работать, даже если вкладка не является активной вкладкой или анимация прокручивается со страницы и т. д.

**Работа с DOM**

Для JavaScript веб-страница доступна в виде **объектной модели документа** (*document object model*, **DOM)**. **DOM** описывает структуру веб-станицы в виде древовидного представления и предоставляет разработчикам способ получить доступ к элементам станицы.

**Объект document**

Для работы со структурой *DOM* предназначен объект **document** (из объекта *window*). Объект **document** предоставляет ряд свойств и методов для управления элементами страницы.

Для поиска элементов на странице применяются следующие методы:

* *getElementById* (value): выбирает элемент, у которого атрибут *id* равен *value;*
* *getElementsByTagName* (value): выбирает все элементы, у которых тег равен *value;*
* *getElementsByClassName* (value): выбирает все элементы, которые имеют класс *value*;
* *querySelector* (value): выбирает элемент, который соответствует селектору *value;*
* *querySelectorAll* (value): выбирает все элементы, соответствующие селектору *value.*

<div>

<h3 id="header"> Block Header </h3>

<p> Text </p>

</div>

<script>

var headerElement = document.**getElementById** ("header");

document.write("Заголовок: " + headerElement.innerText);

</script>

Метода *getElementsByTagName* получает группу тегов по их имени. Пример:

<div>

<h3> Заголовок </h3>

<p> Первый абзац </p>

<p> Второй абзац </p>

</div>

<script>

var pElements = document.**getElementsByTagName**("p");//находит массив тегов

for (var i = 0; i < pElements.length; i++) {

document.write("Текст: " + pElements**[i]**.innerText + "<br>");

}

</script>

**Свойства объекта document**

Кроме ранее рассмотренных методов объект **document** позволяет обратиться к определенным элементам веб-страницы через свойства:

* *documentElement*: предоставляет доступ к корневому элементу <*html*>;
* *body*: предоставляет доступ к элементу <*body*> на веб-странице;
* *images*: содержит коллекцию всех объектов изображений (элементов *img*);
* и т.д.

<img src="picure1.png" alt="Картинка 1">

<img src="picure2.png" alt="Картинка 2">

<img src="picure3.png" alt="Картинка 3">

<script>

var images = document.**images**; // получим все изображения

images[0].src="picture4.jpg"; // изменим первое изображение

for(var i=0; i<images.length;i++){ // перебирем все изображения

    document.write("<br>" + images[i].src + " " + images[i].alt);

}

</script>

**Объект Node**

Каждый отдельный узел в структуре *DOM* представлен объектом **Node**. Этот объект предоставляет ряд свойств, с помощью которых можно получить информацию о данном узле:

* *childNodes*: содержит коллекцию дочерних узлов;
* *firstChild*: возвращает первый дочерний узел текущего узла;
* *lastChild*: возвращает последний дочерний узел текущего узла;
* *previousSibling*: возвращает предыдущий элемент, который находится на одном уровне;
* *nextSibling*: возвращает следующий элемент, который находится на одном уровне;
* *parentNode*: возвращает элемент, который содержит текущий узел (родительский узел);
* *nodeName*: возвращает имя узла;
* *nodeType*: возвращает тип узла в виде числа;
* *nodeValue*: возвращает или устанавливает значение узла в виде простого текста;

<div class="article">

<h3>Заголовок статьи</h3>

<p>Абзац</p>

</div>

<script>

var articleDiv = document.querySelector("div.article"); // выбираем div

var nodes = articleDiv.childNodes; // получаем все дочерние узлы div-а

for(var i=0; i<nodes.length;i++){ // перебираем полученные узлы

var type = "";

if(nodes[i].nodeType===1) type="элемент";

else if(nodes[i].nodeType===2) type="атрибут";

else if(nodes[i].nodeType===3) type="текст";

console.log(nodes[i].nodeName + " : " + type);

}

</script>

В данном примере, несмотря на то, что в блоке *div* на странице только три узла: *h3* и 2 параграфа, но консоль отобразит нам пять узлов. Дело в том, что пробелы (переносы строки) между узлами также считаются за отдельные текстовые узлы.

**Работа с элементами веб-станицы**

**Добавление элемента**

Для создания элементов объект **document** имеет следующие методы:

* *createElement* (elementName): создает элемент, тег которого передается в качестве параметра. Возвращает созданный элемент.
* *createTextNode* (text): создает и возвращает текстовый узел. В качестве параметра передается текст узла.

var elem = document.createElement("div");

var elemText = document.createTextNode("Привет мир");

Однако одного создания элементов недостаточно, их еще надо добавить на веб-страницу.

Для добавления элементов можно использовать один из методов объекта **Node**:

* *appendChild* (newNode): добавляет узел *newNode* в конец коллекции дочерних узлов;
* *insertBefore* (newNode, referenceNode): добавляет *newNode* перед узлом *referenceNode*.

<div class="article">

<h3>Заголовок</h3>

</div>

<script>

var articleDiv = document.querySelector("div.article");

var elem = document.createElement("h2"); // создаем элемент <h2>

var elemText = document.createTextNode("Привет мир"); // создаем текст

elem.**appendChild**(elemText); // добавляем текст внутрь элемента <h2>

articleDiv.**appendChild**(elem); // добавляем элемент <h2> в блок <div>

</script>

Необязательно для определения текста внутри элемента создавать дополнительный текстовый узел, так как можно использовать свойство **textContent**:

elem.textContent = "Привет мир";

**Копирование элемента**

Метод **cloneNode**() предназначен для копирования узлов у объекта **Node**:

var articleDiv = document.querySelector("div.article");

var newArticleDiv = articleDiv.**cloneNode**(true); // клонируем элемент

document.body.appendChild(newArticleDiv); // добавляем в конец элемента body

В метод **cloneNode**() в качестве параметра передается логическое значение: если *true*, то элемент будет копироваться с дочерними узлами; если *false* – без дочерних узлов.

**Удаление элемента**

Метод **removeChild**() объекта **Node** удаляет один из дочерних узлов:

var articleDiv = document.querySelector("div.article");

var remNode = document.querySelectorAll("div.article p")[0]; // находим узел

articleDiv.**removeChild**(remNode); // удаляем узел

**Замена элемента**

Для замены элемента применяется метод **replaceChild**(newNode, oldNode) объекта **Node**.

Этот метод в качестве первого параметра принимает новый элемент *newNode*, который заменяет старый элемент *oldNode*, передаваемый в качестве второго параметра.

var articleDiv = document.querySelector("div.article");

var oldNode = document.querySelectorAll("div.article p")[0]; // находим узел

var newNode = document.createElement("h2"); // создаем элемент

var elemText = document.createTextNode("Привет мир"); // создаем текст

newNode.appendChild(elemText); // добавляем текст в элемент

articleDiv.replaceChild(newNode, oldNode); // заменяем старый узел новым

**Объект Element**

Кроме методов и свойств объекта *Node* можно использовать свойства и методы объектов **Element**. Объекты **Element** представляют только *html*-элементы. По сути те же самые узлы *Node*, у которых тип узла (*nodeType*) равно **1** (теги).

Важное свойство объекта **Element** – свойство *tagName*, которое возвращает тег элемента:

<div class="article">

<h3>Заголовок статьи</h3>

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

<script>

function getChildren(elem){

    for(var i in elem.childNodes){

        if(elem.childNodes[i].nodeType===1){

            console.log(elem.childNodes[i].**tagName**); // выводим имя тега

            getChildren(elem.childNodes[i]); // рекурсивный вызов функции

        }

    }

}

var root = document.documentElement;

console.log(root.**tagName**); // Корневой тег HTML

getChildren(root);

</script>

**Свойства innerText и innerHTML**

Для получения или установки текстового содержимого элемента можно использовать свойство **innerText**, а для получения или установки кода *html* - свойство **innerHTML**:

var articleDiv = document.querySelector("div.article");

console.log(articleDiv.innerText); // включает только текст

console.log(articleDiv.innerHTML); // включает и теги

Надо отметить, что свойство **innerText** во многом аналогично свойству **textContent**.

**Свойство outerHTML**

Кроме свойства *innerHTML* существует также свойство **outerHTML**, которое перезаписывает не только внутренний текст тега, но и сам тег. Пример:

<p id="test">Абзац</p>

<script>

var elem = document.getElementById('test');

elem.outerHTML = '<b>Полужирный текст</b>'; // <p> заменится на <b>

</script>

**Методы объекта Element**

Среди методов объекта **Element** можно отметить методы управления атрибутами:

* *getAttribute* (attr): возвращает значение атрибута *attr;*
* *setAttribute* (attr, value): устанавливает для атрибута *attr* значение *value*;
* *removeAttribute* (attr): удаляет атрибут *attr* и его значение;
* *hasAttribute* (attr): проверяет наличие или отсутствие атрибута *attr*.

**Работа с атрибутами**

Доступ к атрибутам можно получить и без использования специальных функций. Можно получить их значения, написав через точку. Пример:

<input type="text" value="!" id="test">

<script>

var elem = document.getElementById('test');

alert(elem.**value**); //выведет '!'

alert(elem.**id**); //выведет 'test'

alert(elem.**type**); //выведет 'text'

</script>

Можно не только считывать значения атрибутов, но и изменять их:

var elem = document.getElementById('test');

elem.value = 'www'; //присвоим новое значение атрибуту value

**Атрибуты data-**

В языке HTML5 разрешено добавлять свои атрибуты тегам, при этом они должны начинаться с **data-**, а затем должно идти любое имя атрибута.

Если вы хотите обратиться к таким атрибутам как с свойствам объекта, то это делается при помощи специального свойства **dataset** и указания имени атрибута (без *data-*):

<div id="elem" data-price="1000" data-product-number="5">

Товар

</div>

<script>

var elem = document.getElementById('elem');

alert(elem.**dataset**.price); //выведет 1000

alert(elem.**dataset**.productNumber); //выведет 5

</script>

К таким атрибутам можно также обращаться с помощью методов типа *getAttribute*. В этом случае следует писать полное название атрибута.

**Изменение стиля элементов**

Для работы со стилевыми свойствами элементов применяются два подхода:

* Изменение свойства *style*.
* Изменение значения атрибута *class*.

**Свойство style**

Свойство **style** представляет сложный объект для управления стилем и напрямую сопоставляется с атрибутом *style* *html*-элемента. Этот объект содержит набор свойств CSS:

element.style.свойство\_CSS

var root = document.documentElement;

root.style.color = "blue"; // устанавливаем стиль

Однако ряд свойств *css* в названиях имеют дефис (например, *font-family*). В JavaScript для этих свойств дефис не употребляется. Только первая буква, которая идет после дефиса, переводится в верхний регистр:

root.style.fontFamily = "Verdana";

**Свойство className**

С помощью свойства **className** можно установить атрибут *class* элемента *html*.

<style>

.**newStyle**{ font-family:Verdana; }

     .article{ font-size:20px; }

</style>

<div class="article">

Текст

</div>

<script>

var articleDiv = document.querySelector("div.article");

articleDiv.**className** = "**newStyle**"; // установка нового класса

alert(articleDiv.**className**); // получаем название класса

</script>

**Свойство classList**

Для управления множеством классов удобнее использовать свойство **classList**. Это свойство представляет объект, реализующий следующие методы:

* *add* (className): добавляет класс *className;*
* *remove* (className): удаляет класс *className;*
* *toggle* (className): если класса нет, то он добавляется, если есть, то удаляется.

Знакомство с JS - <https://webref.ru/dev/learn-javascript>

Самоучитель JS - <https://learn.javascript.ru/>

Самоучитель JS - <https://metanit.com/web/javascript/>

Справочник JS - <https://javascript.ru/manual>

Справочник JS - <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference>

**Общие задания**

Выполните следующие задания с использованием JS:

1. Создайте несколько элементов *div* (минимум 5). Создайте массив цветов (желательно оттенки одного цвета) в таком же количестве. Каждый элемент *div* должен иметь цвет из массива последовательно (то есть 1-ый элемент – 1-ый цвет, 2-ой элемент – второй цвет и т.д.). Сделайте так, чтобы каждую секунду цвет заднего фона каждого элемента менялся на следующий по порядку в массиве. С последнего должен меняться на первый цвет. Пример:

1 секунда: 

2 секунда: 

3 секунда: 

1. Создайте элемент *div*. Сделайте его круглым. В случайном порядке выбирайте координаты на окне браузера. Этот элемент должен двигаться в эту точку в течение нескольких секунд. Постарайтесь сделать движение плавным (можно использовать *requestAnimationFrame()* и *transfotm: translate()* для перемещения). После того, как он достигает выбранной координаты, должна выбираться новая случайная координата и так далее.
2. Напишите функцию, которая из массива данных формирует *html*-список. На странице пользователь должен вводить данные (*prompt*) в цикле до тех пор, пока не введет пустую строку. После этого в функцию должен передаваться считанный массив данных. Функция должна сформировать html-список как объект *Node*, при этом элементы списка также должны быть объектами *Node* (используйте функции *createElement*() и *appendChild*() для этого). Функция возвращает объект html-список в качестве результат. Присоедините его к странице в *body*.
3. Создайте страницу, на которой генерируются случайные посты. Создайте массив ссылок на картинки (нужно для случайной генерации картинок). Создайте массив предложений (нужно для случайной генерации текстов). Пользователь должен вводить количество постов (*K*) и количество предложений (*Kpr*) в каждом посте (*prompt*).

Для создания поста в первую очередь создайте элемент *div.* Далее создайте элемент картинку (*img*). Добавьте эту картинку в *div* как дочерний элемент. Создайте абзац (*p*) и также добавьте в *div* как дочерний элемент. Используйте для этого *createElement*() и *appendChild*().

Создайте массив постов и заполните его с помощью *cloneNode*() копиями поста в количестве *K*. Для каждой копии:

Задайте цвет шрифта и цвет заднего фона случайным образом. Задайте случайную ссылку на картинку элемента *img* с этого *div*. Случайным образом заполните элемент *p* предложениями в количестве *Kpr.*

Добавьте все посты из массива на страницу.