# Лабораторная работа №4

Тема: Изучение JavaScript и DOM

**Задание**

На основе лабораторной работы №2 или №3 добавить на страницу элементы интерактивности.

**Требования**

* Реализовать добавление поста или статьи на страницу;
* Добавить кнопки лайк и дизлайк к посту, показывать количество лайков;
* Добавить кнопку для переключения темы или других стилей;
* Создать обработчики событий мыши, клавиатуры, формы и окна (хотя бы по 1 на каждую группу).

# Справочная информация

## Введение в JavaScript

JavaScript — это полноценный динамический язык программирования, который применяется к HTML документу и может обеспечить динамическую интерактивность на веб-сайтах.

Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Им не нужна специальная подготовка или компиляция для запуска.

Несмотря на то, что JavaScript разрабатывался как клиентский язык для веб-приложений, он использоваться и на стороне сервера. Например, библиотека node.js использует JavaScript в качестве серверного языка для обращения к базе данных, получения доступа к файловой системе сервера, выполнения сетевых запросов и произведения многих других действий на сервере.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером. Например, в браузере JavaScript может:

* Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
* Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
* Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы
* Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
* Запоминать данные на стороне клиента («local storage»).

Язык JavaScript также известен под названием ECMAScript. ECMAScript - это официальная спецификация JavaScript, разработанная и поддерживаемая организацией по стандартизации Ecma, так что запись вроде ES6 означает всего лишь 6 версию языка JavaScript.

## Подключение к HTML

Для того, чтобы работать с языком JavaScript необходимо создать в папке проекта файл с расширением .js.

Далее перейдите в ваш html файл и введите следующий элемент на новой строке прямо перед закрывающим тегом </body>:

<script src="myscript.js"></script>

Этот код выполняет ту же работу, что и элемент <link> для CSS — добавляет JavaScript на страницу, позволяя ему взаимодействовать с HTML.

Причиной, по которой мы поставили элемент <script> в нижней части HTML файла, является то, что HTML-элементы загружаются браузером в том порядке, в котором они расположены в файле.

## Особенности языка JavaScript

Данный курс предполагает, что вы знакомы с основами программирования, поэтому кратко разберем, как выглядит синтаксис JS.

### Структура кода

Инструкции разделяются точкой с запятой:

alert('Привет'); alert('Мир');

Точка с запятой не требуется после блоков кода {…} и синтаксических конструкций с ними, таких как, например, циклы:

function f() { … }

for(;;) { … }

Единственными исключениями являются операторы присваивания, такие как let obj = {};

Комментарии задаются следующим образом:

//однострочный комментарий

/\*многострочный комментарий\*/

В JavaScript можно использовать следующие методы для отправки сообщений в консоль:

console.log() для общих сообщений;

console.dir() для объектов браузера;

console.warn() для предупреждений;

console.error() для ошибок.

Кроме того, консоли предоставляют возможность установки контрольных точек и слежения за кодом в целях отладки.

Консоль можно открыть в браузере с помощью контекстного меню или комбинацией ctrl+shift+i.

### Переменные

Переменные можно объявить при помощи let, const (константа) и var (устаревший способ).

Ключевое слово let позволяет объявлять переменные с ограниченной областью видимости — только для блока {...}, в котором происходит объявление. Это называется блочной областью видимости. Вместо ключевого слова var, которое обеспечивает область видимости внутри функции, стандарт ES6 и выше рекомендует использовать let.

var a = 2;

{

let a = 3;

console.log(a); // 3

}

console.log(a); // 2

Другой формой объявления переменной с блочной областью видимости является ключевое слово const. Оно предназначено для объявления переменных (констант), значения которых доступны только для чтения. Это означает не то, что значение константы неизменно, а то, что идентификатор переменной не может быть переприсвоен.

Вот простой пример:

{

const ARR = [5, 6];

ARR.push(7);

console.log(ARR); // [5,6,7]

ARR = 10; // TypeError

ARR [0] = 3; // значение можно менять

console.log(ARR); // [3,6,7]

}

Имя переменной может включать буквы и цифры, однако цифра не может быть первым символом. Символы $ и \_ используются наряду с буквами. JavaScript чувствителен к регистру: myVariable отличается от переменной myvariable. Также в качестве имени переменной нельзя использовать зарезервированные слова языка, такие как class, return и т.д.

Переменные типизируются динамически. В них могут храниться любые значения:

let x = 5;

x = "Вася";

### Типы данных

number – целые и вещественные числа

string – строки, заключаются в кавычки.

boolean - логические значения истинности или ложности: true/false

null – тип с единственным значением null, т.е. «пустое значение» или «значение не существует»

undefined – тип автоматически присваивается переменным, которые были только объявлены или аргументам, для которых не были установлены значения

object - объекты

symbol – примитивный тип данных, использующийся для создания уникальных идентификаторов. Может быть использовано как ключ для свойства объекта.

Оператор typeof возвращает тип значения переменной, с двумя исключениями:

typeof null == "object" // ошибка в языке

typeof function(){} == "function" // именно для функций

### Объекты

Объекты используются для хранения коллекций различных значений и более сложных сущностей. Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств.

Свойство – это пара «ключ: значение», где ключ – это строка (также называемая «именем свойства»), а значение может быть чем угодно.

Пустой объект можно создать, используя один из двух вариантов синтаксиса:

let user = new Object(); // синтаксис "конструктор объекта"

let user = {}; // синтаксис "литерал объекта"

Обычно используют вариант с фигурными скобками {...}. Такое объявление называют литералом объекта или литеральной нотацией.

При использовании литерального синтаксиса {...} мы сразу можем поместить в объект несколько свойств в виде пар «ключ: значение»:

let user = { // объект

name: "John", // под ключом "name" хранится значение "John"

age: 30 // под ключом "age" хранится значение 30

};

Свойства объекта также иногда называют полями объекта. У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие ":", и затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

Для обращения к свойствам используется запись «через точку»: user.name.

Оператор "in" для проверки существования свойства в объекте

Синтаксис оператора: "key" in object

Пример:

let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age существует

alert( "birthday" in user ); // false, user. birthday не существует

Обратите внимание, что слева от оператора in должно быть имя свойства. Обычно это строка в кавычках.

Если мы опускаем кавычки, это значит, что мы указываем переменную, в которой находится имя свойства. Например:

let user = { age: 30 };

let key = "age";

alert( key in user ); // true, имя свойства было взято из переменной key

Цикл «for…in»

Для перебора всех свойств объекта используется цикл for..in. Этот цикл отличается от изученного ранее цикла for(;;).

Синтаксис:

for (key in object) {

// тело цикла выполняется для каждого свойства объекта

}

К примеру, чтобы выведести все свойства объекта user:

let user = {

name: "John",

age: 30,

isAdmin: true

};

for (let key in user) {

// ключи

alert( key ); // name, age, isAdmin

// значения ключей

alert( user[key] ); // John, 30, true

}

Все конструкции «for» позволяют нам объявлять переменную внутри цикла, как, например, let key здесь.

Присваивание и копирование объектов

Объекты в javascript присваиваются и копируются по ссылке. Другими словами, переменная хранит не «значение объекта», а «ссылку» (адрес в памяти) на этот объект. Поэтому копирование такой переменной или передача её в качестве аргумента функции приводит к копированию этой ссылки, а не самого объекта. Все операции с использованием скопированных ссылок (например, добавление или удаление свойств) выполняются с одним и тем же объектом. Таким образом, при изменении скопированного по ссылке объекта изменится также исходный.

Поверхностное копирование объектов

Поверхностно скопированным объектом можно называть объект, в котором скопированы данные верхнего уровня, но вложенные объекты будут также использоваться между оригиналом и копией.

Создать поверхностную копию можно с помощью метода Object.assign или оператора (…):

let objCopy = Object.assign({}, obj); ⬄ let objCopy = {…obj};

Например:

let student = {

fullname: ‘Иванов Иван’,

age:17

marks: {

maths: 5,

literature: 4

},

}

При копировании данного объекта поля fullname и age станут независимыми, а оценки останутся ссылочным объектом поскольку это вложенный объект.

Чтобы сделать более глубокую копию, можно проделать данную процедуру еще раз:

let objCopy = {…obj},

objCopy.marks = {…obj. marks }

Также можно воспользоваться методами cloneDeep библиотеки Lodash или более быстрой fastest-clone.

В JavaScript есть много других типов объектов:

Array для хранения упорядоченных коллекций данных,

Date для хранения информации о дате и времени,

Error для хранения информации об ошибке и т.д.

Но они не являются отдельными типами, а относятся к типу Object.

### Массивы

Существует два варианта синтаксиса для создания пустого массива:

let arr = new Array();

let arr = [];

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

let students = ["Иванов И.И", "Петров П.П.", "Сидоров С.С."];

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках: students[0]

Общее число элементов массива содержится в его свойстве length.

В массиве могут храниться элементы любого типа.

Например:

let arr = ['Кошка', 5, true, {name: 'Маня'}];

Внутреннее устройство массива

Массив – это особый подвид объектов. Квадратные скобки, используемые для того, чтобы получить доступ к свойству arr[0] – это по сути обычный синтаксис доступа по ключу, как obj[key], где в роли obj у нас arr, а в качестве ключа – числовой индекс.

Массивы расширяют объекты, так как предусматривают специальные методы для работы с упорядоченными коллекциями данных, а также свойство length. Но в основе всё равно лежит объект.

Циклы в массивах

Для массивов возможен вариант цикла, for..of:

for (let student of students) { //тело функции }

Также для перебора массива используется классический for (;;).

Методы массива

Для добавления/удаления элементов:

push (...items) – добавляет элементы в конец,

pop() – извлекает элемент с конца,

shift() – извлекает элемент с начала,

unshift(...items) – добавляет элементы в начало.

splice(pos, deleteCount, ...items) – начиная с индекса pos, удаляет deleteCount элементов и вставляет items.

slice(start, end) – создаёт новый массив, копируя в него элементы с позиции start до end (не включая end).

concat(...items) – возвращает новый массив: копирует все члены текущего массива и добавляет к нему items. Если какой-то из items является массивом, тогда берутся его элементы.

Для поиска среди элементов:

indexOf/lastIndexOf(item, pos) – ищет item, начиная с позиции pos, и возвращает его индекс или -1, если ничего не найдено.

includes(value) – возвращает true, если в массиве имеется элемент value, в противном случае false.

find/filter(func) – фильтрует элементы через функцию и отдаёт первое/все значения, при прохождении которых через функцию возвращается true.

findIndex похож на find, но возвращает индекс вместо значения.

Для перебора элементов:

forEach(func) – вызывает func для каждого элемента. Ничего не возвращает.

Для преобразования массива:

map(func) – создаёт новый массив из результатов вызова func для каждого элемента.

sort(func) – сортирует массив «на месте», а потом возвращает его.

reverse() – «на месте» меняет порядок следования элементов на противоположный и возвращает изменённый массив.

split/join – преобразует строку в массив и обратно.

reduce(func, initial) – вычисляет одно значение на основе всего массива, вызывая func для каждого элемента и передавая промежуточный результат между вызовами.

Методы sort, reverse и splice изменяют исходный массив.

### Взаимодействие с пользователем

В качестве рабочей среды мы используем браузер, так что простейшими функциями взаимодействия с пользователем являются:

prompt (question, [default])

Задаёт вопрос question и возвращает то, что ввёл посетитель, либо null, если посетитель нажал на кнопку «Отмена».

confirm(question)

Задаёт вопрос question и предлагает выбрать «ОК» или «Отмена». Выбор возвращается в формате true/false.

alert(message)

Выводит сообщение message.

### Операторы

JavaScript поддерживает следующие операторы:

Арифметические

Простые + (сложение), - (вычитание), /(деление), \*(умножение), а также деление по модулю % и возведение в степень \*\*.

Бинарный плюс + объединяет строки. А если одним из операндов является строка, то второй тоже будет конвертирован в строку:

alert( '1' + 2 ); // '12', строка

alert( 1 + '2' ); // '12', строка

Операторы присваивания

Простые a = b и составные a \*= 2.

Условия

Условия — это структуры, которые позволяют вам проверять, истинно или ложно выражение, а затем выполнить другой код в зависимости от результата. Самая распространенная форма условия if ... else. Например:

let catName = 'Murzik';

if (catName === 'Murzik') {

alert('ks-ks-ks');

} else {

alert('It’s not my cat');

}

Выражение внутри if ( ... ) — это проверка, которая использует тождественный оператор (как описано выше), чтобы сравнить переменную catName со строкой 'Murzik' и увидеть равны ли они. Если это сравнение возвращает true, выполнится первый блок кода. Если нет, этот код пропустится и выполнится второй блок кода, после инструкции else.

Кроме того, условие можно задать в виде:

cond ? resultA : resultB. Если условие cond истинно, возвращается resultA, иначе – resultB.

let catName = 'Murzik';

(catName === 'Murzik') ? alert('ks-ks-ks') : alert('It’s not my cat');

Логические операторы

Логические И &&, ИЛИ || используют так называемое «ленивое вычисление» и возвращают значение, на котором оно остановилось (не обязательно true или false). Логическое НЕТ ! конвертирует операнд в логический тип и возвращает инвертированное значение.

Сравнение

Проверка на равенство == значений разных типов конвертирует их в число (за исключением null и undefined, которые могут равняться только друг другу), так что примеры ниже равны:

alert( 0 == false ); // true

alert( 0 == '' ); // true

Другие операторы сравнения тоже конвертируют значения разных типов в числовой тип.

Оператор строгого равенства === не выполняет конвертирования: разные типы для него всегда означают разные значения.

Значения null и undefined особенные: они равны == только друг другу, но не равны ничему ещё.

Операторы сравнения больше/меньше сравнивают строки посимвольно, остальные типы конвертируются в число.

Конструкция «switch»

Конструкция «switch» может заменить несколько проверок if. При сравнении она использует оператор строгого равенства ===.

Например:

let age = prompt('Сколько вам лет?', 18);

switch (age) {

case 18: alert("Так не сработает"); // результатом prompt является строка, а не число

case "18": alert("А так сработает!"); break;

default: alert("Любое значение, неравное значению выше");

}

Spread / Rest оператор

... оператор называют как spread или rest, в зависимости от того, как и где он используется.

При использовании в любом итерируемом объекте, данный оператор "разбивает" его на индивидуальные элементы:

function foo(x, y, z) {

console.log(x, y, z);

}

let arr = [1, 2, 3];

foo(...arr); // 1 2 3

Другим распространённым использованием оператора ... является объединение набора значений в один массив. В данном случае оператор работает как "rest".

function foo(...args) {

console.log(args);

}

foo(1, 2, 3, 4, 5); // [1, 2, 3, 4, 5]

### Циклы

Синтаксис для трех видов циклов:

while (condition) { ... }

do { ... } while (condition);

for(let i = 0; i < 10; i++) { ... }

Переменная, объявленная в цикле for(let...), видна только внутри цикла. Но мы также можем опустить let и переиспользовать существующую переменную.

Директивы break/continue позволяют выйти из цикла/текущей итерации. Используйте метки для выхода из вложенных циклов.

### Функции

Функции - ключевая концепция в JavaScript. Любая функция - это объект, и следовательно ею можно манипулировать как объектом, в частности:

* передавать как аргумент и возвращать в качестве результата при вызове других функций (функций высшего порядка);
* создавать анонимно и присваивать в качестве значений переменных или свойств объектов.

Это позволяет относить JavaScript к числу языков, реализующих функциональную парадигму программирования.

У функций могут быть локальные переменные: т.е. объявленные в теле функции. Такие переменные видимы только внутри функции. У параметров могут быть значения по умолчанию: function sum(a = 1, b = 2) {...}.

Функции всегда что-нибудь возвращают. Если нет оператора return, результатом будет undefined. Функции можно задать разными способами, рассмотрим самые распространённые.

**Function Declaration**: функция в основном потоке кода

function sum(a, b) {

let result = a + b;

return result;

}

**Function Expression**: функция как часть выражения

let sum = function sum (a, b) {

let result = a + b;

return result;

};

В конце стоит точка с запятой, поскольку функция по синтаксису является инструкцией. В function declaration точки с запятой ставить не нужно.

Также функция может быть вида "function definition expression" и быть анонимной (не иметь имени).

let sum = function (a, b) {

let result = a + b;

return result;

};

Стрелочные функции

Стрелочные функции — функции вида "function expression", имеют укороченный ситаксис и лексически связывает значение this. Стрелочные функции всегда анонимны. Рассмотрим как могут выглядеть стрелочные функции:

Выражение в правой части

let sum = (a, b) => a + b;

Многострочный код в фигурных скобках { ... }, здесь необходимо использовать return:

let sum = (a, b) => {

// ...

return a + b;

}

Выражение без аргументов

let sayHi = () => alert("Привет");

Выражение с одним аргументом

let double = n => n \* 2;

Функции обратного вызова

Функция обратного вызова (от англ. «call back» – обратный вызов) - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия.

Ключевая идея в том, что мы передаём функцию и ожидаем, что она вызовется обратно когда-нибудь позже, если это будет необходимо.

Вложенные функции и замыкания

Функции можно вкладывать друг в друга. Вложенная функция — приватная и помещена в другую функцию. Так образуется замыкание. Замыкание — это выражение, которое может иметь свободные переменные вместе со средой, которая связывает эти переменые.

Таким образом вложенная функция может унаследовать аргументы и переменные функции, в которую она вложена и использовать их, в то время как внешняя функция не может использовать аргументы и переменные вложенной функции.

Параметры по умолчанию

JS позволяет установить параметры по умолчанию при объявлении функции. Вот простой пример:

let getFinalPrice = (price, tax = 0.7) => price + price \* tax;

getFinalPrice(500); // 850, так как значение tax не задано

getFinalPrice(500, 0.2); // 600, значение tax по-умолчанию заменяется на 0.2

## DOM

Объектная Модель Документа (DOM) – это программный интерфейс для HTML и XML документов.

Любой документ с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями «родительский-дочерний». Например, узел body может включать в себя div, который включает в себя узлы p и h1 и т.д. Все, что есть в HTML, даже комментарии, является частью DOM.

DOM позволяет программам и скриптам получить доступ к содержимому HTML и XML документов, изменять содержимое, структуру и оформление таких документов.

Объект document – основная «входная точка». С его помощью мы можем что-то создавать или менять на странице. Например:

// заменим цвет фона на красный,

document.body.style.background = "red";

Мы использовали в примере только document.body.style, но на самом деле возможности по управлению страницей намного шире.

Небольшой пример выше – это JavaScript. То есть пример написан на JavaScript, но при этом используется DOM для доступа к документу и его элементам. DOM не является языком программирования, но без него JavaScript не имел бы никакой модели или представления о веб-странице, HTML-документе, XML-документе и их элементах. Каждый элемент в документе: весь документ в целом, заголовок, таблицы внутри документа, заголовки таблицы, текст внутри ячеек таблицы — это части объектной документной модели для этого документа, поэтому все они могут быть доступны и могут изменяться с помощью DOM и скриптового языка наподобие JavaScript.

Основные методы

Так как с помощью методов DOM можно изменить что угодно на HTML странице, их количество очень велико. Рассмотрим некоторые из них.

Получение элементов

document.getElementById(id) - Возвращает ссылку на элемент по его идентификатору

document.getElementsByTagName(tag) - Возвращает коллекцию элементов с указанным именем тега. Поиск осуществляется по всему документу, включая корневой узел.

document.getElementsByClassName(class) - Возвращает коллекцию элементов с указанным именем класса.

document.querySelector(selector) - возвращает первый элемент, который соответствует одному или более селекторам. Данных метод может заменить любой их трех методов getElement, но работает немного медленее.

document.querySelectorAll(selector) - возвращает все элементы, соответствующие селектору. Удобен для поиска вложенных селекторов, например «div p» найдет все параграфы вложенные в элемент div.

Методы работы с элементами

element.addEventListener(event, listener) – назначает функцию к определенному событию для элемента. event — целевое событие, listener — это JavaScript функция.

element.innerHTML - устанавливает или получает содержимое HTML документа

element.setAttribute (name, value) - добавляет новый атрибут или изменяет значение существующего атрибута у выбранного элемента. name - имя атрибута, value - значение атрибута.

element.getAttribute (attributeName) - возвращает значение указанного атрибута элемента

element.tagName - Возвращает HTML-тег элемента

Методы работы с узлами

node.parentNode - возвращает родителя определенного элемента DOM дерева.

node.childNodes - возвращает коллекцию дочерних элементов данного элемента.

node.lastChild - возвращает последнего потомка в узле

node.firstChild - возвращает первого потомка в узле

Методы для различных вариантов вставки

node.append(...nodes or strings) – добавляет узлы или строки в конец node,

node.prepend(...nodes or strings) – вставляет узлы или строки в начало node,

node.before(...nodes or strings) –- вставляет узлы или строки до node,

node.after(...nodes or strings) –- вставляет узлы или строки после node,

node.replaceWith(...nodes or strings) –- заменяет node заданными узлами или строками.

Методы для создания узлов

document.createElement(tag) – создаёт элемент с заданным тегом

document.createTextNode(value) – создаёт текстовый узел (редко используется),

element.cloneNode(deep) – клонирует элемент, если deep==true, то со всеми дочерними элементами.

Методы для работы с классами

ClassList является геттером. Возвращаемый им объект имеет несколько методов:

element.classList.add( String [,String] ) – добавляет элементу указанные классы

element.classList.remove( String [,String] ) – удаляет у элемента указанные классы

element.classList.item ( Number ) – результат аналогичен вызову сlassList[Number]

element.classList.toggle ( String [, Boolean]) – если второй параметр не указан или равен true, то если класс у элемента отсутствует - добавляет, иначе - убирает. Когда вторым параметром передано false - удаляет указанный класс, а если true - добавляет.

contains ( String ) – проверяет, есть ли данный класс у элемента (вернет true или false)

## Браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

Вот список самых часто используемых DOM-событий:

### События

События мыши

click – одинарный щелчок левой кнопкой мыши (нажата и отпущена кнопка мыши)

dblclick – двойной щелчок

contextmenu – щелчок правой кнопкой мыши на элементе

selectstart – начало выделения контента.

mousewheel – нажата кнопка мыши в пределах текущего элемента

mousemove – перемещение курсора мыши в пределах текущего элемента

mouseover – наведение курсора мыши на текущий элемент

mouseout – выведение курсора мыши за пределы текущего элемента

mousedown – нажата кнопка мыши в пределах текущего элемента

mouseup – отпущена кнопка мыши в пределах текущего элемента

События элементов форм

focus – получение элементом фокуса (щелчок мышью на элементе или очередное нажатие клавиши табуляции)

blur – потеря текущим элементом фокуса, т.е. переход к другому элементу. Возникает при щелчке мышью вне элемента либо нажатии клавиши табуляции

change – изменение значений элементов формы. Возникает после потерей элементом фокуса, т.е. после события blur

reset – сброс данных формы ( щелчок по кнопке <input type="reset"> )

select – выделение текста в текущем элементе

submit – отправка данных формы ( щелчок по кнопке <input type="submit"> )

abort – прерывание загрузки изображения

Клавиатурные события

keydown – нажата клавиша на клавиатуре

keyup – отпущена клавиша на клавиатуре

keypress – нажата и отпущена клавиша на клавиатуре

События окна браузера

load – закончена загрузка документа

unload – попытка закрытия окна браузера и выгрузки документа

error – возникновение ошибки выполнения сценария

move – перемещение окна

resize – изменение размеров окна

scroll – прокрутка окна или области

### Обработчики событий

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло. Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик. Сейчас мы их рассмотрим, начиная с самого простого.

Использование атрибута HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.

Например, чтобы назначить обработчик события click на элементе input, можно использовать атрибут onclick, вот так:

<input type="button" onclick="alert('Клик!')" value="Кнопка">

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

Обратите внимание, для содержимого атрибута onclick используются одинарные кавычки, так как сам атрибут находится в двойных. Если мы забудем об этом и поставим двойные кавычки внутри атрибута, вот так: onclick="alert("'Клик!")", код не будет работать.

Атрибут HTML-тега – не самое удобное место для написания большого количества кода, поэтому лучше создать отдельную JavaScript-функцию и вызвать её там:

<input type="button" id="button" value="Кнопка">

<script>

button.onclick = function() {

alert('Клик!');

};

</script>

Эти два примера кода работают одинаково.

Использование свойства DOM-объекта

Можно назначать обработчик, используя свойство DOM-элемента on<событие>.

К примеру, elem.onclick:

<input id="elem" type="button" value=" Кнопка">

<script>

elem.onclick = function() {

alert(''Клик');

};

</script>

Если обработчик задан через атрибут, то браузер читает HTML-разметку, создаёт новую функцию из содержимого атрибута и записывает в свойство.

Обработчик всегда хранится в свойстве DOM-объекта, а атрибут – лишь один из способов его инициализации.

В примере ниже назначение через JavaScript перезапишет обработчик из атрибута:

<input type="button" id="elem" onclick="alert('Было')" value="Нажми меня">

<script>

elem.onclick = function() { // перезапишет существующий обработчик

alert('Станет'); // выведется только это

};

</script>

Обработчиком можно назначить и уже существующую функцию:

function sayThanks() {

alert('Спасибо!');

}

elem.onclick = sayThanks;

Доступ к элементу через this

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на который назначен обработчик.

В коде ниже button выводит своё содержимое, используя this.innerHTML:

<button onclick="alert(this.innerHTML)">Нажми меня</button>

Частые ошибки

Если вы только начинаете работать с событиями, обратите внимание на следующие моменты.

Функция должна быть присвоена как sayThanks, а не sayThanks().

// правильно

button.onclick = sayThanks;

// неправильно

button.onclick = sayThanks();

Если добавить скобки, то sayThanks() – это уже вызов функции, результат которого (равный undefined, так как функция ничего не возвращает) будет присвоен onclick. А вот в разметке, в отличие от свойства, скобки нужны:

<input type="button" id="button" onclick="sayThanks()">

Это различие просто объяснить. При создании обработчика браузером из атрибута, он автоматически создаёт функцию с телом из значения атрибута: sayThanks().

Так что разметка генерирует такое свойство:

button.onclick = function() {

sayThanks(); // содержимое атрибута

};

addEventListener

Фундаментальный недостаток описанных выше способов назначения обработчика –- невозможность повесить несколько обработчиков на одно событие. Например, одна часть кода при клике на кнопку должна делать её подсвеченной, а другая – выдавать сообщение.

Мы хотим назначить два обработчика для этого. Но новое DOM-свойство перезапишет предыдущее:

input.onclick = function() { alert(1); }

// ...

input.onclick = function() { alert(2); } // заменит предыдущий обработчик

Разработчики стандартов достаточно давно это поняли и предложили альтернативный способ назначения обработчиков при помощи специальных методов addEventListener и removeEventListener. Они свободны от указанного недостатка.

Синтаксис добавления обработчика:

element.addEventListener(event, handler[, options]);

event

Имя события, например "click".

handler

Ссылка на функцию-обработчик.

options

Дополнительный объект со свойствами:

once: если true, тогда обработчик будет автоматически удалён после выполнения.

capture: фаза, на которой должен сработать обработчик, подробнее об этом будет рассказано в главе Всплытие и погружение. Так исторически сложилось, что options может быть false/true, это тоже самое, что {capture: false/true}.

passive: если true, то указывает, что обработчик никогда не вызовет preventDefault(), подробнее об этом будет рассказано в главе Действия браузера по умолчанию.

Для удаления обработчика следует использовать removeEventListener:

element.removeEventListener(event, handler[, options]);

Для удаления нужно передать именно ту функцию-обработчик которая была назначена.

function handler() {

alert( 'Спасибо!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);

Объект события

Чтобы обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее.

Когда происходит событие, браузер создаёт объект события, записывает в него детали и передаёт его в качестве аргумента функции-обработчику.

Пример ниже демонстрирует получение координат мыши из объекта события:

<input type="button" value="Нажми меня" id="elem">

<script>

elem.onclick = function(event) {

// вывести тип события, элемент и координаты клика

alert(event.type + " на " + event.currentTarget);

alert("Координаты: " + event.clientX + ":" + event.clientY);

};

</script>

Некоторые свойства объекта event:

event.type - Тип события, в данном случае "click".

event.currentTarget - Элемент, на котором сработал обработчик. Значение – обычно такое же, как и у this, но если обработчик является функцией-стрелкой или при помощи bind привязан другой объект в качестве this, то мы можем получить элемент из event.currentTarget.

event.clientX / event.clientY - Координаты курсора в момент клика относительно окна, для событий мыши.

Объект-обработчик: handleEvent

Мы можем назначить обработчиком не только функцию, но и объект при помощи addEventListener. В этом случае, когда происходит событие, вызывается метод объекта handleEvent. К примеру:

<button id="elem">Нажми меня</button>

<script>

elem.addEventListener('click', {

handleEvent(event) {

alert(event.type + " на " + event.currentTarget);

}

});

</script>

Как видим, если addEventListener получает объект в качестве обработчика, он вызывает object.handleEvent(event), когда происходит событие.