**Основы JS**

**Конспект «Условия и создание элементов»**

**Как подключить на страницу несколько скриптов**

Чтобы подключить на страницу ещё один файл с кодом на JavaScript, снова используем тег script:

<body>

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

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

</body>

Браузер обрабатывает инструкции последовательно: сначала в первом файле, потом во втором — как будто они все находятся в одном месте. Программы часто разбивают на несколько файлов, обычно одному файлу соответствует одна задача: например, управление темами или подписка на рассылку.

**Числа**

Переменной можно присвоить значение — число. Числа не нужно оборачивать в кавычки:

let number = 0;

Чтобы увеличить или уменьшить число в JavaScript, можно использовать разные записи:

// Полная запись

number = number + 2; // Значение переменной: 2

number = number - 2; // Значение переменной: 0

// Краткая запись

number += 2; // Значение переменной: 2

number -= 2; // Значение переменной: 0

// Увеличение числа на 1

number++; // Значение переменной: 1

// Уменьшение числа на 1

number--; // Значение переменной: 0

**Метод classList.contains**

Метод classList.contains проверяет, есть ли у элемента класс:

элемент.classList.contains('класс');

Метод вернёт true (истина), если класс у элемента есть, и false (ложь), если класса нет. Значения true и false называют булевыми. Таких значений всего два.

**Условная конструкция**

Условная конструкция позволяет выполнять действия в зависимости от *условия*. Условие — это инструкция, которая возвращает true или false. Выглядит условная конструкция так:

if (условие) {

// Инструкции, которые выполнятся, если условие истинно

} else {

// Инструкции, которые выполнятся, если условие ложно

}

if

Условие записывают в *круглых* скобках после слова if (переводится с английского как «если»). После этого внутри *фигурных* скобок пишут инструкции, которые выполнятся, если условие *истинно*. Условие считается истинным, если инструкция внутри круглых скобок возвращает true.

else

Конструкция else (переводится как «иначе») говорит JavaScript, что делать, если условие ложно. Внутри фигурных скобок после else пишут инструкции, которые должны выполниться, если условие вернёт false.

Использование условных конструкций в скрипте ещё называют ветвлением, а код внутри фигурных скобок — веткой.

**Метод append**

элемент-родитель.append(добавляемый-элемент);

Метод append добавляет указанный в скобках элемент в конец элемента-родителя. При этом содержимое элемента-родителя не затирается. Добавлять с помощью этого метода можно и элементы, и простые строки.

**Метод createElement**

document.createElement('имя тега');

Чтобы создать новый элемент на странице, к которой подключён скрипт, нужно использовать слово document. Внутри скобок в кавычках указывают элемент, который нужно создать. Например:

// Создаём новый элемент <div> и записываем его в переменную

let newElement = document.createElement('div');

Новый элемент будет доступен из скрипта, но в разметке не появится, пока мы не скажем JavaScript, где разместить новый элемент. Для этого можно использовать метод append:

// Находим элемент-родитель

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

// Добавляем новый элемент на страницу

parent.append(newElement);

Элементы, созданные с помощью метода createElement, можно изменять так же, как и любые другие.

**Очищаем поле ввода**

Чтобы пользователь по ошибке не отправил форму несколько раз, после отправки поле ввода лучше очистить. Для этого в его свойство value записывают пустую строку. Вот так:

let input = document.querySelector('input');

input.value = '';

# Конспект «Коллекции и свойства элементов»

## Метод querySelectorAll

Метод querySelectorAll находит все элементы на странице, которые соответствуют указанному селектору, и возвращает **коллекцию** — набор этих элементов.

// Найдёт все абзацы на странице

let elements = document.querySelectorAll('p');

## Коллекция

Коллекцию можно сохранить в переменной. Самый простой способ узнать, какие элементы содержит коллекция, — вывести её в консоль:

// Выведет коллекцию в консоль

console.log(elements);

В консоли коллекция выглядит как список, в котором элементы перечислены через запятую. Весь список обёрнут в квадратные скобки, а у элементов указан только их тег и, например, класс. Чтобы элементы отобразились так же, как в разметке, коллекцию нужно развернуть, кликнув на стрелку-треугольник слева.

[p.card\_\_text, p, p]

<p class="card\_\_text">Готовим мороженое!</p>

<p>Санкт-Петербург</p>

<p>mail@htmlacademy.ru</p>

К элементу коллекции можно обращаться по индексу. **Индекс** — это порядковый номер элемента в коллекции. Отсчёт начинается с нуля, поэтому у первого элемента индекс 0, а у второго — 1. Индексы пишут в квадратных скобках после имени коллекции:

console.log(elements[0]); // Выведет первый элемент коллекции

console.log(elements[1]); // Выведет второй элемент коллекции

## Data-атрибуты

В HTML можно создавать свои собственные атрибуты. Имена таких атрибутов начинаются с префикса data-, после которого идёт любое выбранное разработчиком слово.

<div data-cat-name="Кекс">

Чтобы получить значение data-атрибута в JavaScript, используют свойство dataset, после которого указывают имя атрибута без префикса data-:

элемент.dataset.имяАтрибутаБезПрефикса

Если имя атрибута состояло из нескольких слов и в нём были дефисы, то в JavaScript его записывают в «верблюжьем» стиле (по-английски *camelCase*): дефисы убирают, а каждое слово, кроме первого, пишут с большой буквы.

let element = document.querySelector('div');

console.log(element.dataset.catName); // Выведет: Кекс

## Цикл for of

Цикл — это конструкция, которая позволяет выполнить код несколько раз. Цикл for of выполнит код из фигурных скобок столько раз, сколько элементов содержится в коллекции, указанной в круглых скобках. Каждое такое повторение называется *итерацией*.

for (переменная of коллекция) {

// Код, который нужно выполнить несколько раз

}

При создании цикла в круглых скобках также нужно указать переменную. Обычно для этого объявляют новую переменную и используют её только внутри цикла. На каждой итерации JavaScript будет автоматически записывать в эту переменную очередной элемент коллекции.

let elements = document.querySelectorAll('p'); // Находим все абзацы

for (let element of elements) { // Создаём цикл и переменную

console.log(element); // Выводим элементы в консоль

}

Цикл for of завершится, когда в коллекции закончатся элементы. После этого JavaScript перейдёт к инструкциям, которые идут после цикла.

## Обработчик событий oninput

Обработчик событий oninput (в переводе с английского это означает «при вводе») позволяет выполнять инструкции из фигурных скобок каждый раз, когда меняется значение в поле ввода. Изменением считается и добавление, и удаление символов.

// Найдём поле ввода

let textarea = document.querySelector('textarea');

// Добавим обработчик событий

textarea.oninput = function () {

// Выведем данные из поля ввода

console.log(textarea.value);

};

## Свойство length

Узнать длину строки можно с помощью свойства length (по-английски «длина»). Значение этого свойства равно числу символов в строке. Символами считаются не только буквы и цифры, но также пробелы и переносы строки.

let text = 'Я люблю JavaScript';

console.log(text.length); // Выведет: 18

let textarea = document.querySelector('textarea');

console.log(textarea.value); // Выведет: Кекс

console.log(textarea.value.length); // Выведет: 4

## Оператор сравнения >

Оператор сравнения > («больше») сравнивает два числа и возвращает булево значение: true, если левое число больше правого, и false во всех остальных случаях:

console.log(3 > 2); // Вернёт: true

console.log(1 > 2); // Вернёт: false

console.log(2 > 2); // Вернёт: false

## Свойство disabled

Блокировать и разблокировать кнопку в JavaScript можно, присваивая булевы значения свойству disabled (по-английски значит «отключён») этой кнопки. Если присвоено значение true, то кнопка заблокирована, а если false — разблокирована.

let button = document.querySelector('button');

// Блокирует кнопку

button.disabled = true;

// Разблокирует кнопку

button.disabled = false;

# Конспект «Прокрутка и операторы»

## Окно браузера и прокрутка

Окно (или вкладка) браузера в JavaScript обозначается словом window.

### Обработчик событий onscroll

Чтобы отследить прокрутку, используем обработчик событий onscroll. Он срабатывает каждый раз, когда страницу прокручивают, даже если её сдвинули всего на один пиксель.

window.onscroll = function () {

console.log('Страница прокручена');

}

### Свойство pageYOffset

Свойство pageYOffset окна браузера содержит количество пикселей, на которое пользователь прокрутил страницу по вертикали:

// Если мы на самом верху страницы

console.log(window.pageYOffset); // Выведет: 0

// Прокрутим страницу на 200px

console.log(window.pageYOffset); // Выведет: 200

Величина горизонтальной прокрутки хранится в свойстве pageXOffset.

### Метод scrollTo

Чтобы прокрутить страницу, используем метод scrollTo:

window.scrollTo(координата-Х, координата-Y);

Координата Х указывает, куда нужно прокрутить страницу по горизонтали, а координата Y — куда нужно прокрутить страницу по вертикали. Когда браузер выполнит инструкцию, указанная точка окажется в левом верхнем углу окна. Координаты задаются в пикселях, но указывать единицы измерения не нужно:

// Прокрутит страницу на 100px вправо и на 50px вниз

window.scrollTo(100, 50);

Если прокрутить страницу до указанных координат не получается, браузер прокрутит её так далеко, как сможет, но увеличивать страницу не будет. Если страница помещается в окно целиком и полосы прокрутки нет, то браузер проигнорирует эту инструкцию.

## Обработчик событий onchange

Обработчик событий onchange срабатывает, когда пользователь выбирает новое значение из выпадающего списка.

// Находим выпадающий список

let select = document.querySelector('select');

// Добавляем обработчик событий

select.onchange = function () {

// Выводим в консоль новое значение

console.log(select.value);

};

Обработчик событий onchange можно использовать с разными элементами. Например, он срабатывает, когда пользователь переключает [чекбокс](https://htmlacademy.ru/courses/46/run/10) или [радиокнопки](https://htmlacademy.ru/courses/46/run/12).

## Оператор строгого равенства

Чтобы проверить, равны ли два значения, используем оператор строгого равенства. Он обозначается тремя знаками равно:

'a' === 'a'; // Результат: true

'a' === 'b'; // Результат: false

Оператор строгого равенства сравнивает два значения и возвращает true, если они равны, и false, если они не равны. Значения, которые проверяет оператор, называют *операндами*.

## Оператор строгого неравенства

Оператор строгого неравенства работает противоположно оператору строгого равенства. Он сравнивает два значения и возвращает false, если значения равны, и true, если они не равны.

Оператор строгого неравенства обозначается восклицательным знаком и двумя знаками равно:

'a' !== 'a'; // Результат: false

'a' !== 'b'; // Результат: true

В JavaScript также есть оператор *нестрогого* равенства == и оператор нестрогого неравенства !=. О них мы поговорим в одной из [следующих частей](https://htmlacademy.ru/courses/209/run/5).

## Оператор логическое И

Чтобы объединить две части условия, используем логическое И. Оно обозначается двойным амперсандом — &&.

if (article.dataset.category !== filter.value && filter.value !== 'all') {

article.classList.add('hidden');

}

Логическое И возвращает true, только если обе части условия возвращают true. Если хотя бы одна из частей вернёт false, то условие целиком также будет считаться ложным:

true && true; // Результат: true

true && false; // Результат: false

false && true; // Результат: false

false && false; // Результат: false

## Оператор логическое ИЛИ

Логическое ИЛИ обозначается двумя вертикальными чертами и возвращает true, если хотя бы один из операндов возвращает true:

true || true; // Результат: true

true || false; // Результат: true

false || true; // Результат: true

false || false; // Результат: false