**Конспект «Основы JavaScript»**

**JavaScript: что это такое и как подключить его на страницу**

Язык программирования JavaScript придумали специально для того, чтобы создавать интерактивные сайты.

Код на языке JavaScript называют скриптом. Его сохраняют в отдельный файл с расширением js, а чтобы запустить, подключают этот файл на страницу. В HTML для добавления JavaScript есть специальный тег:

<script src="адрес\_файла"></script>

Подключают скрипт обычно в самом конце страницы, перед закрывающим тегом </body>.

Программа на JavaScript — это последовательность инструкций, то есть указаний браузеру выполнить какие-то действия. Инструкции выполняются последовательно, сверху вниз.

Чтобы сказать JavaScript, что инструкция закончена, нужно поставить точку с запятой или перейти на новую строку. Новая строка правильно работает *в большинстве случаев*, а точка с запятой — **всегда**. Поэтому лучше ставить точку с запятой в конце каждой инструкции.

JavaScript не меняет исходный файл с разметкой, но, выполняя инструкции, меняет страницу прямо в браузере пользователя.

**Комментарии**

Комментарий — это текст, поясняющий код. Он не выводится в браузер и никак не влияет на работу программы. Инструкции внутри комментария не выполняются, поэтому комментарии часто используют, если нужно временно отключить часть кода.

В JavaScript есть два вида комментариев:

// Однострочные комментарии.

/\*

И многострочные.

Они могут отключить сразу несколько строк кода.

\*/

**Метод querySelector**

Чтобы найти на странице элемент, нужно использовать метод querySelector, он ищет по селектору:

document.querySelector('селектор');

Эта инструкция состоит из двух частей. Первая часть — элемент, внутри которого будет искать JavaScript. Словом document обозначается веб-страница, к которой скрипт подключён. Неважно, как называется файл на самом деле, в JavaScript это всегда «документ». Он является элементом-родителем для любого другого элемента на странице.

Вторая часть инструкции — это то, что нужно сделать. Её называют методом.

**Консоль**

Консоль — инструмент разработчика, который помогает тестировать код. Если во время выполнения скрипта возникнет ошибка, в консоли появится сообщение о ней. А ещё в консоль можно выводить текстовые подсказки. Чтобы вывести сообщение в консоль, нужно использовать console.log:

console.log('Привет от JavaScript!');

// Выведет: Привет от JavaScript!

console.log(document.querySelector('.page'));

// Выведет в консоль найденный элемент

**Переменная**

Переменная — способ сохранить данные, дав им понятное название.

Переменную можно создать, или **объявить**, с помощью ключевого слова let. За ним следует имя переменной. После объявления в переменную нужно записать, или **присвоить**, какое-то значение:

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

Имя переменной может быть почти любым, но не должно начинаться с цифры, а из спецсимволов разрешены только '\_' и '$'. Ещё для именования переменных нельзя использовать [зарезервированные слова](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Lexical_grammar#%D0%9A%D0%BB%D1%8E%D1%87%D0%B5%D0%B2%D1%8B%D0%B5_%D1%81%D0%BB%D0%BE%D0%B2%D0%B0). Имена переменных чувствительны к регистру: header, Header и HEADER — это разные переменные. Имя переменной должно описывать то, что в ней хранится.

Когда в коде встречается переменная, браузер вместо её имени подставляет присвоенное ей значение. Когда мы используем переменную, снова писать let не нужно:

console.log(header);

Ключевое слово let появилось в JavaScript в 2015 году, до этого для объявления переменных использовалось слово var.

**Методы для изменения классов**

Чтобы убрать у элемента класс, нужно использовать метод classList.remove. Он убирает с элемента тот класс, который указан в скобках:

элемент.classList.remove('класс');

Чтобы добавить элементу класс, нужно использовать метод classList.add:

элемент.classList.add('класс');

Метод-переключатель classList.toggle убирает у элемента указанный класс, если он есть, и добавляет, если этого класса нет:

элемент.classList.toggle('класс');

**Свойство textContent**

У каждого элемента имеется множество свойств: его размеры, цвет и так далее. Свойство textContent хранит в себе текстовое содержимое элемента. Свойствам можно присваивать новые значения:

let paragraph = document.querySelector('p');

paragraph.textContent = 'Здесь был Кекс. Мяу!';

**Свойство value**

У полей ввода есть особое свойство — value. Оно хранит данные, введённые в поле. Мы можем вывести их прямо на страницу:

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

paragraph.textContent = input.value;

**Конкатенация**

Операция, когда мы «склеиваем» несколько значений, называется **конкатенацией** и в JavaScript выполняется с помощью знака плюс.

let name = 'Кекс';

paragraph.textContent = 'Вас зовут ' + name + '. Хорошего дня!';

console.log(paragraph.textContent);

// Выведет: Вас зовут Кекс. Хорошего дня!

**Обработчики событий onclick и onsubmit**

JavaScript следит за всем, что происходит на странице. Клик по кнопке или отправка формы — это *событие*. Мы можем сказать JavaScript, что сделать, когда некое событие произойдёт. Для этого используют обработчики событий. Инструкции, которые должны будут выполниться, когда событие произойдёт, располагают между фигурных скобок.

Свойство onclick означает «по клику»:

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

button.onclick = function() {

console.log('Кнопка нажата!');

};

При каждом клике по кнопке в консоли будет появляться новое сообщение Кнопка нажата!.

За обработку отправки формы отвечает свойство onsubmit:

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

form.onsubmit = function() {

console.log('Форма отправлена!');

};

После отправки формы в консоли появится сообщение Форма отправлена!.

# 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://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/courses/46/run/10) или [радиокнопки](https://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/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://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/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

# Конспект «Динамические стили элементов»

## Свойство style

С помощью свойства style можно управлять стилями элемента. После style через точку указывают то CSS-свойство, которое нужно изменить. Чтобы изменить стиль элемента, указанному свойству нужно присвоить новое значение

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

// Зададим абзацу зелёный цвет текста

element.style.color = 'green';

Стили, заданные с помощью свойства style, работают так же, как если бы их указали в разметке в [атрибуте](https://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/courses/307/run/15) style самого элемента. Они имеют больший [приоритет](https://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/courses/66/run/8), чем CSS-правила из файла со стилями.

В JavaScript нельзя использовать дефисы в названиях свойств, вместо этого для разделения слов используется [«верблюжий» стиль](https://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/courses/349/run/5). Например:

| **CSS** | **JаvaScript** |
| --- | --- |
| font-size | fontSize |
| background-color | backgroundColor |
| border-left-width | borderLeftWidth |

Если мы получаем данные из поля ввода, то чтобы задать единицы измерения, используем конкатенацию:

longread.style.fontSize = sizeSetting.value + 'px';

// Допустим, пользователь ввёл число 16

longread.style.fontSize = 16 + 'px'; // Результат: '16px'

Чтобы узнать, какие стили уже применяются к элементу, используйте метод window.getComputedStyle.

## Обработчики onchange и oninput

Чтобы отслеживать изменения в поле ввода, можно использовать обработчики onchange и oninput. Разница между ними заключается в следующем:

* onchange срабатывает, если значение поля ввода изменилось и пользователь закончил ввод. Например, если пользователь передвинул ползунок и отпустил его. Или ввёл что-то в текстовое поле и убрал из него курсор.
* oninput срабатывает на каждое изменение значения, независимо от того, завершил пользователь ввод или нет. Например, он сработает на каждое изменение положения ползунка, даже если пользователь продолжает его двигать. И на каждый новый символ в текстовом поле, даже если пользователь продолжает вводить текст.

Когда мы меняем размер элементов, браузеру приходится перерисовывать страницу. Это трудоёмкая операция, и лучше выполнять её как можно реже. Используйте обработчик oninput с осторожностью.

## Свойство type

Для ввода пароля используется специальное поле с типом password. Особенность этого поля в том, что введённый текст в нём маскируется. Обычно браузеры используют для этого звёздочки или кружочки. Подробнее о типах полей вы можете узнать из [этой части](https://up.htmlacademy.ru/profession/react-lite/1/lite-prepare/1/courses/46).

Чтобы показать пароль, нужно превратить поле ввода пароля в текстовое поле ввода. Для этого надо изменить его тип на text. За тип в JavaScript отвечает свойство type. Чтобы изменить тип поля ввода, нужно записать в свойство type новое значение:

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

// Сделаем input текстовым полем ввода

input.type = 'text';

## Чекбокс и свойство checked

Чекбокс — это поле ввода, у которого может быть одно из двух состояний: включён или выключен. Обычно в браузерах на состояние чекбокса указывает галочка: если галочка стоит — чекбокс включён, если галочки нет — выключен. Когда галочку ставят или убирают, состояние чекбокса меняется. Чтобы отследить это событие из скрипта, используем обработчик событий onchange.

Чтобы проверить состояние чекбокса, используем свойство checked. Это свойство имеет булево значение: true, если чекбокс включён, и false, если нет.

// Проверяем, включён ли чекбокс

if (showPassword.checked) {

// Инструкции выполнятся, если чекбокс включён

} else {

// Инструкции выполнятся, если чекбокс выключен

}

## Операторы сравнения и умножение

В JavaScript есть несколько операторов сравнения. Эти операторы работают так же, как в математике:

1 > 2; // Оператор «больше»

1 < 2; // Оператор «меньше»

1 >= 2; // Оператор «больше или равно»

1 <= 2; // Оператор «меньше или равно»

Операторы «больше или равно» и «меньше или равно» обозначаются двумя символами: угловой скобкой и знаком равно.

Умножение в JavaScript обозначается звёздочкой:

console.log(3 \* 10); // Выведет: 30

## Конструкция else if

Конструкция else if позволяет добавить в условную конструкцию альтернативную ветку с условием. После else if в круглых скобках указывают условие, а в фигурных — инструкции, которые должны выполниться, если это условие вернёт true.

if (условие-1) {

// Инструкции выполнятся, если истинно условие-1

} else if (условие-2) {

// Инструкции выполнятся, если условие-1 ложно, а условие-2 истинно

} else {

// Инструкции выполнятся, если оба условия ложны

}

JavaScript выполняет инструкции сверху вниз. Сначала он проверит условие-1. Если оно истинно, выполнятся инструкции из первой ветки. Если условие-1 ложно, JavaScript проверит условие-2. Если оно истинно, выполнятся инструкции из второй ветки. Если оба условия ложны, то выполнятся инструкции из ветки else.

Веток else if в условной конструкции может быть сколько угодно. Но чем их больше, тем запутаннее получается код.