Конспект «JS в браузере». 1 часть

Поиск элементов на странице:

// Поиск элемента по тегу

var list = document.querySelector('ul');

// Поиск последнего элемента из списка

var lastProduct = document.querySelector('li:last-child');

// Поиск элемента по классу

var price = document.querySelector('.price');

// Поиск третьего элемента из списка товаров

var thirdProduct = document.querySelector('.product:nth-child(3)');

// Поиск всех элементов, подходящих по селектору

var listItems = document.querySelectorAll('.product');

querySelectorAll возвращает список (коллекцию) элементов. Этот список похож на массив, но им не является. Он называется псевдомассив и его можно перебирать с помощью цикла for.

Добавление класса элементу страницы:

// Когда ищем элемент по классу, используем точку

var product = document.querySelector('.product');

// Но когда добавляем класс, точки нет!

product.classList.add('product--sale');

Результат работы classList.add() такой же, как при ручном добавлении класса в разметку:

<!-- Исходное состояние разметки -->

<li class="product">

…

</li>

<!-- Состояние после вызова classList.add -->

<li class="product product--sale">

…

</li>

Конспект «JS в браузере». 2 часть

Свойство DOM-элемент.children возвращает коллекцию дочерних, то есть вложенных, DOM-элементов.

Создание элемента и добавление его в DOM-дерево:

var list = document.querySelector('.cards');

// Создаём новый элемент

var card = document.createElement('li');

card.classList.add('card');

// После вызова этого метода новый элемент отрисуется на странице

list.appendChild(card);

Вот что произойдёт с разметкой после вызова appendChild:

<!-- Исходное состояние разметки -->

<ul class="cards">

<li class="card">Существующий элемент</li>

</ul>

<!-- Состояние после вызова appendChild -->

<ul class="cards">

<li class="card">Существующий элемент</li>

<li class="card">Добавленный элемент</li>

</ul>

Работа с текстовым содержимым элемента:

// HTML

<p>Я — <em>текстовый элемент</em>.</p>

// JS

var p = document.querySelector('p');

console.log(p.textContent);

// Выведет: Я — текстовый элемент.

p.textContent = 'Теперь у меня новое содержимое.';

console.log(p.textContent);

// Выведет: Теперь у меня новое содержимое.

// В HTML содержание тега изменится

<p>Теперь у меня новое содержимое.</p>

Работа с изображениями:

// Создание изображения

var picture = document.createElement('img');

// Добавляем адрес картинки

picture.src = 'images/picture.jpg';

// Добавляет альтернативный текст

picture.alt = 'Непотопляемая селфи-палка';

// Добавляет изображение в конец родительского элемента

element.appendChild(picture);

Конспект «События в JavaScript». 1 часть

События — действия пользователя на странице (клик по кнопке, нажатие клавиши).

Добавление обработчиков событий

button.addEventListener('click', function () {

// Инструкции

});

В примере:

button — элемент, на котором мы хотим «слушать» событие.

addEventListener() — функция добавления обработчика события на элемент.

'click' — общепринятое название события, первый параметр функции addEventListener. Названия всех событий можно посмотреть здесь.

Второй параметр addEventListener — функция-обработчик, в ней записаны инструкции, которые выполнятся, только когда произойдёт событие.

Обратите внимание, мы передаём функцию, а не её вызов. Если мы вызовем функцию, код из этой функции выполнится сразу и больше не сработает.

А нам нужно, чтобы код выполнился асинхронно — в момент, когда произойдёт событие.

// Так добавлять обработчик неправильно

button.addEventListener('click', function () {

console.log('Клик по кнопке');

}());

// Сообщение сразу же выведется в консоль

// А такой код верный

button.addEventListener('click', function () {

console.log('Клик по кнопке');

});

// Сообщение выведется, когда произойдёт событие клика

В примере выше мы передаём в обработчик функцию, у которой нет своего имени, она не записана в переменную. Мы создали её там же, где передаём.

Такие функции, которые создаются в момент передачи и не имеют имени, называются анонимными функциями.

Объект event

Объект event — параметр функции-обработчика. Он всегда передаётся браузером в эту функцию в момент наступления события. Этот объект содержит

много полезных свойств и методов.

Чтобы использовать event, достаточно указать этот объект параметром функции-обработчика и написать инструкции. Остальное сделает JavaScript.

Среди некоторых разработчиков принято называть параметр сокращённо — evt, во избежание ошибок.

Действия по умолчанию

Некоторые элементы страницы имеют действия по умолчанию или дефолтные действия. Например, клик по кнопке отправления формы вызывает отправку

данных этой формы на сервер, а при клике по ссылке браузер переходит по этой ссылке.

Объект event содержит метод, который отменяет действие элемента по умолчанию: preventDefault().

link.addEventListener('click', function(evt) {

// Отменяем действие по умолчанию

evt.preventDefault();

// Добавляем инструкции для события клика

console.log('Произошёл клик');

});

Клавиатурные события

У события «нажатие на клавишу» есть специальное название — 'keydown'. Такое событие срабатывает при нажатии на любую клавишу. Обратите внимание,

слушать это событие можно только на элементах, которые имеют состояние фокуса: поля ввода, кнопки, элементы с атрибутом tabindex, документ.

При нажатии фокус должен находиться на соответствующем элементе.

Если мы хотим поймать нажатие какой-то конкретной клавиши, можно обратиться к свойству keyCode объекта event. Это свойство содержит код нажатой клавиши.

Например, у Enter код 13, а у ESC — 27. Эти номера универсальны и одинаковы в любой раскладке. Найти код любой клавиши можно здесь.

document.addEventListener('keydown', function(evt) {

// Проверяем, что код клавиши равен 27

if (evt.keyCode === 27) {

// Код отсюда выполнится только при нажатии ESC

}

});

Кроме keyCode есть и другие свойства для определения нажатой клавиши. Например, key и code. Они возвращают названия клавиш, а не их номера. Э

ти свойства пока поддерживаются не во всех браузерах, но когда поддержка станет лучше, стоит начать использовать их вместо keyCode в соответствии

с современным стандартом JavaScript.

Конспект «События в JavaScript». 2 часть

Области видимости

У каждой функции есть область видимости — все значения, доступные для этой функции.

Область видимости ограничена функцией, поэтому снаружи нельзя получить локальные переменные и параметры функции.

Локальные переменные — переменные, у которых область видимости ограничена функцией, где они объявлены. Такая область видимости называется локальной.

Глобальные переменные — переменные, которые объявлены на уровне всей программы, их видно из любого блока кода. Область видимости, в которой они объявлены,

называется глобальной.

Если внутри функции обратиться не к локальной переменной, JavaScript будет искать переменную снаружи, переходя наверх от уровня к уровню, пока не найдёт переменную.

Если переменной не будет ни внутри функции ни снаружи, будет ошибка.

Так как функция может использовать переменные, объявленные снаружи, их можно переопределять.

var food = 'макароны';

var eatDinner = function () {

console.log('Поел ' + food);

};

eatDinner();

// Выведет: Поел макароны

// Переопределяем переменную food

food = 'сельдерей';

eatDinner();

// Выведет: Поел сельдерей

Переопределять снаружи переменные, которые использует функция — не лучшая практика. Это может приводить к неожиданным последствиям и ошибкам в коде.

Использовать это нужно осторожно.

Области видимости создаются только функциями. Поэтому, если переменная была создана в другой конструкции, например, в цикле, она будет доступна для чтения из функции.

Замыкания

Замыкание — функция, которая помнит о своём окружении. Это функция + все значения вне локальной области видимости, которые она использует.

Благодаря замыканиям мы можем зафиксировать какое-то значение в функции, а использовать саму функцию позже.

var collectContainer = function (food) {

return function () {

console.log('Поел ' + food);

};

};

var schoolkid = collectContainer('макароны');

schoolkid();

// Выведет: Поел макароны

Замыкания и асинхронность

Некоторые функции выполняются асинхронно, поэтому в момент выполнения кода значение переменной может уже измениться. Чтобы избавиться от этой проблемы,

нужно создать отдельную область видимости. Так все переменные будут под контролем и замыкания не позволят потерять необходимые значения.

var thumbnails = document.querySelectorAll('.gallery\_\_photo-preview');

var fullPhoto = document.querySelector('.full-photo');

var addThumbnailClickHandler = function (thumbnail, photo) {

thumbnail.addEventListener('click', function () {

fullPhoto.src = photo;

});

};

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

addThumbnailClickHandler(thumbnails[i], photos[i]);

}

Конспект «Манипуляции с DOM»

Коллекции

Есть несколько способов, чтобы найти сразу несколько элементов на странице. Элементы записываются в структуру, которая называется коллекцией.

Коллекции похожи на массив, но ими не являются. При этом к элементам коллекции можно обращаться по индексу и перебирать в цикле for, как и обычные массивы.

Ниже несколько способов получения коллекций:

element.querySelectorAll() — возвращает все элементы, которые подходят под указанное правило. Эта запись остаётся статичной и изменения в DOM на неё никак не влияют.

Можно сказать, что querySelectorAll работает, как любая переменная, в которую мы записали какое-нибудь значение. Пока мы не переопределим переменную, в ней так и будет

находиться то значение, которое мы в неё записали, независимо от того, что происходит в коде. Поэтому такая коллекция называется статичной.

parentElement.children — вызывается на родительском элементе и собирает все дочерние элементы в динамическую коллекцию. Такие коллекции реагируют на изменения в DOM.

Если один из элементов коллекции будет удалён из DOM, то он пропадёт и из коллекции.

Работа с элементами

Удаление элемента

Удалять элементы со страницы можно разными способами, один из самых простых — вызов метода remove на элементе, который нужно удалить.

element.remove();

Метод из примера выше удалит element из DOM.

Клонирование элемента

С помощью клонирования мы можем копировать элементы сколько угодно раз и вставлять их в любые места на странице. Для этого существует метод cloneNode.

Синтаксис такой:

element.cloneNode(true);

// Вернёт склонированный элемент со всеми вложенностями

element.cloneNode(false);

// Вернёт склонированный элемент без вложенностей

element.cloneNode();

// 0\_o

При передаче true в качестве аргумента клонируется сам элемент вместе со всеми вложенностями. Причём клонируются атрибуты, классы и текстовое содержимое

всех вложенностей. Такое клонирование называется глубоким.

Если передать методу в качестве аргумента значение false, то будет скопирован сам элемент со своими классами и атрибутами, но без дочерних элементов.

Лучше всегда явно передавать аргумент в cloneNode, чтобы избежать ошибок в работе программ.

Как получить текст из поля ввода

Нужно обратиться к свойству поля ввода value. Оно хранит информацию, введённую в поле.

input.value;

Результат можно сохранить в переменную и использовать дальше в коде.

Шаблоны и тег template

Тег template хранит в себе шаблон для будущих элементов. Он там же, где и вся разметка сайта, только его содержимое не отображается на странице.

Чтобы получить template в JavaScript, можно найти его по идентификатору. Это уникальное название записывают в атрибут id. Такой атрибут можно указывать

для разных элементов, главное соблюдать правило — значение атрибута не должно повторяться на одной странице.

Шаблон в разметке:

<body>

…

<template id="text-template">

<p class="text"></p>

</template>

</body>

Поиск элемента в JavaScript:

document.querySelector('#text-template');

Решётка в параметре querySelector обозначает, что искать надо по id.

Внутри template находится document-fragment или просто фрагмент. Он является хранилищем содержимого тега template. Именно благодаря ему разметка из template

не отображается на странице.

Чтобы получить необходимые элементы в шаблоне, надо обратиться к document-fragment, он находится в свойстве content и дальше искать нужные элементы

привычными методами поиска.

<body>

…

<template id="text-template">

<p class="text"></p>

</template>

</body>

Если мы хотим найти элемент в шаблоне, надо искать так:

var template = document.querySelector('#text-template');

// Нашли template в документе

var content = template.content;

// Получили содержимое, фрагмент

var text = content.querySelector('.text');

// Нашли нужный шаблон

Эту запись можно сократить. Например, записать в отдельную переменную контент, а в другую искомый шаблон.

var textTemplate = document.querySelector('#text-template').content;

var text = textTemplate.querySelector('.text');

События

change — срабатывает, когда состояние поля ввода меняется. Например, невыбранный чекбокс становится выбранным или наоборот.

submit — реагирует на отправку формы. Формы отправляются по умолчанию так же, как при клике по ссылке происходит переход по указанному адресу.

Если вам не нужно отправлять форму в каких-то случаях, отмените действие по умолчанию с помощью preventDefault.