<https://developer.mozilla.org/en-US/docs/Web/Events#event_listing>

Список названий событий

ПОПАП

HTML

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<title>CandyShop</title>

<link href="candy-shop/setting.css" rel="stylesheet">

<link href="style.css" rel="stylesheet">

</head>

<body>

<section class="card">

<div class="card\_\_content">

<h1 class="visually-hidden">Магазин мороженого CandyShop</h1>

<img class="card\_\_logo" src="candy-shop/candyshop.svg" width="205" height="63" alt="CandyShop">

<p class="card\_\_text">Готовим мороженое из самых свежих сливок!</p>

<img class="card\_\_img" src="candy-shop/icecream.png" width="196" height="240" alt="Мороженое">

</div>

<div class="card\_\_controls">

<a href="contacts.html" class="card\_\_button button-open">Показать контакты</a>

</div>

</section>

<section class="modal modal--show">

<div class="modal\_\_content">

<h2 class="visually-hidden">Контакты</h2>

<h3>Адрес:</h3>

<p>191186, Санкт-Петербург, ул. Большая Конюшенная, д. 19/8</p>

<h3>Телефон:</h3>

<p>+7 (812) 275-75-75</p>

<h3>Почта:</h3>

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

<button class="modal\_\_button button-close" type="button">Закрыть</button>

</div>

</section>

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

</body>

</html>

CSS

h3 {

margin: 0.5em 0;

}

p {

margin: 0.5em 0 2em;

}

.card {

width: 420px;

margin: 50px auto 0;

}

.card\_\_content {

position: relative;

background-color: #fad400;

display: flex;

flex-direction: column;

align-items: flex-start;

padding-top: 15px;

padding-left: 20px;

box-shadow: 0 5px 8px 0 #e8e8e8;

}

.card\_\_img {

align-self: flex-end;

margin-top: auto;

}

.card\_\_logo {

position: absolute;

}

.card\_\_text {

font-size: 16px;

line-height: 22px;

font-weight: bold;

position: absolute;

left: 20px;

top: 100px;

width: 190px;

}

.card\_\_controls {

display: flex;

justify-content: center;

align-items: center;

border: 2px solid #e8e8e8;

border-top: 0;

border-bottom-right-radius: 10px;

border-bottom-left-radius: 10px;

padding-top: 20px;

padding-bottom: 20px;

}

.card\_\_button {

border: 0;

padding: 12px 0;

width: 230px;

text-transform: uppercase;

text-align: center;

border-radius: 50px;

color: #ffffff;

font-weight: bold;

font-size: 16px;

text-decoration: none;

background-color: #fad400;

}

.card\_\_button:hover,

.card\_\_button:focus {

background-color: #82da03;

}

.card\_\_button:active {

color: rgba(255, 255, 255, 0.3);

background-color: #6cb502;

}

.modal {

display: none;

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

}

.modal--show {

display: block;

}

.modal::after {

content: "";

position: fixed;

top: 0;

left: 0;

height: 100%;

width: 100%;

background-color: rgba(0, 0, 0, 0.3);

z-index: 1;

}

.modal\_\_content {

position: relative;

width: 300px;

margin: 100px auto;

padding: 30px 70px 15px;

background-color: #ffffff;

box-shadow: -1px 5px 12px 0 rgba(89, 90, 90, 0.3);

z-index: 2;

}

.modal\_\_button {

position: absolute;

top: 20px;

right: 20px;

width: 20px;

height: 20px;

font-size: 0;

background: none;

border: 0;

}

.modal\_\_button::after,

.modal\_\_button::before {

content: "";

position: absolute;

right: 0;

top: 20px;

width: 26px;

height: 6px;

margin: -13px -3px;

background-color: #7656de;

}

.modal\_\_button::before {

transform: rotate(45deg);

}

.modal\_\_button::after {

transform: rotate(-45deg);

}

.modal\_\_button:hover::after,

.modal\_\_button:hover::before,

.modal\_\_button:focus::after,

.modal\_\_button:focus::before {

background-color: #9b7ff7;

}

.modal\_\_button:active::after,

.modal\_\_button:active::before {

background-color: #4023a0;

}

.visually-hidden {

position: absolute;

width: 1px;

height: 1px;

margin: -1px;

border: 0;

padding: 0;

white-space: nowrap;

clip-path: inset(100%);

clip: rect(0 0 0 0);

overflow: hidden;

}

JS

var popup = document.querySelector('.modal');

var openPopupButton = document.querySelector('.button-open');

var closePopupButton = popup.querySelector('.button-close');

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

evt.preventDefault();

popup.classList.add('modal--show');

});

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

popup.classList.remove('modal--show');

});

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

if (evt.keyCode === 27) {

popup.classList.remove('modal--show')

}

});

код любой клавиши

<https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode>

# Конспект «События в JavaScript». Раздел 1

**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](https://developer.mozilla.org/en-US/docs/Web/Events).
* 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](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode).

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](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key" \t "_blank) и [code](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/code" \t "_blank). Они возвращают названия клавиш, а не их номера. Эти свойства пока поддерживаются не во всех браузерах, но когда поддержка станет лучше, стоит начать использовать их вместо keyCode в соответствии с современным стандартом JavaScript.

var collectContainer = function () {

var food = 'макароны';

var eatDinner = function () {

console.log('Поел ' + food);

}

return eatDinner;

};

var schoolkid = collectContainer();

// В переменной функция eatDinner,

// которую вернула функция collectContainer

// Вызываем функцию eatDinner из переменной schoolkid

schoolkid();

// Выведет: Поел макароны

Галерея

var photos = [

'gallery/laptop-large.jpg',

'gallery/microphone-large.jpg',

'gallery/keyboard-large.jpg',

'gallery/signboard-large.jpg',

'gallery/tree-large.jpg'

];

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 «События в JavaScript». Раздел 2

### 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]);

}