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.
- 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. 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.

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]);
}
```