События. Введение.

Событие - это некое действие.
С точки зрения браузера, событие - это любое происшествие, произошедшее в окне браузера.
Какие бывают события:
- события мыши (нажал/отпустил кнопку мыши, навел указатель мыши на элемент, кликнул левой кнопкой, прокрутил колёсико и т.п.);
- события клавиатуры (нажал/отпустил кнопку, нажал функциональную кнопку);
- события страницы (страница полностью загрузилась, DOM построен и готов к использованию)

Список часто используемых событий здесь
Полный список событий здесь и здесь

Способ обработать произошедшее событие - назначить этому событию **обработчик**.

Обработчики - это то, что позволяет JavaScript взаимодействовать с событиями.
Обработчик - это функция, которая запускается после того, как произошло событие.

Событие всегда привязано к конкретному элементу (к документу, к кнопке, к картинке, к диву и т.д.)
Почти любое событие может быть обработано. Для этого нужно связать функцию-обработчик с конкретным событием.

События. Способы назначения.

Есть несколько способов назначить событие:
1. атрибуты тегов *on + eventName (onclick, onmousedown, onmouseup)*
<element onclick=”yourFunction()”>Test</element>
В данном случае yourFunction должен быть доступен в глобальном пространстве имён. Функцию нужно передать со скобками! ***Чаще всего не используется!***

2. следующий способ связать событие и обработчик - использовать свойства самих узлов DOM в JavaScript. Синтаксис события: *node.on + eventName (node.onclick, node.onkeyup)*

<**button** type="button">Click Me!</**button**>
var **btn** = document.querySelector(‘button’);
btn.**onclick** = function () { alert(‘I was clicked!’); }
btn.**onmouseover** = function () { console.log(‘Mouse over!’); }
btn.**onmouseout** = function () { console.log(‘Mouse out!’); }

Также чаще всего не используется.

События. Способы назначения.

3. наиболее гибкий и современный способ добавить событие - это использовать метод element.**addEventListener**(eventName, handler).
Этот метод позволяет назначать много обработчиков на одно и то же событие.

<**button** type="button">Click Me!</**button**>
var **btn** = document.querySelector('button');
btn.**addEventListener**('click', function () { alert('Was clicked!'); });
btn.**addEventListener**('click', function () { alert('Again was clicked!'); });

Удалить событие, назначенное с помощью addEventListener можно с помощью метода **removeEventListener**(eventName, functionName)

<**button** type="button">Click Me!</**button**>
var **btn** = document.querySelector('button'),
handler = function () { alert('Was clicked!'); };
btn.**addEventListener**('click', handler);
btn.**removeEventListener**('click', handler);
Удалится конкретная функция handler (другие обработчики останутся)

События. Объект события.

Каждый обработчик вызывается с аргументом, который содержит дополнительную информацию о событии (тип события, координаты, информация о нажатой кнопке и т.п.)

```
<button type="button">Click Me!</button>
var btn = document.querySelector('button');
btn.onclick = function (event) { console.log(event.type); }
btn.addEventListener('click', function (event) { alert(event.type); });
```

Более подробно здесь

События. Объект события.

Некоторые события происходят в браузере по умолчанию.
Например, клик по ссылке заставит браузер перейти по ссылке в href. Нажатие на **input** type=submit отправит данные формы на сервер.

Мы можем отменить действие по умолчанию, вызвав в обработчике метод

event.**preventDefault**().

```
<a href=”home”>Click me!</a>
document.querySelector(‘a’).addEventListener(‘click’, function (e) {
   console.log(e.target);
   e.preventDefault();
});

// при клике на ссылку в консоль выведется элемент, но переход по ссылке не произойдёт
```

События. Всплытие.

Если определенное событие назначено некоторому элементу, а событие произошло на дочернем элементе, то такое событие “всплывает” и на родительский элемент. То есть обработчик будет выполнен в любом случае.

В этом случае ***currentTarget*** объекта ***event*** будет содержать элемент, к которому прикреплен обработчик, а ***target*** будет содержать элемент, на котором непосредственно произошло событие.
Событие будет всплывать до самого верха (до window).

Если мы хотим остановить всплытие на определенном элементе, мы можем воспользоваться функцией event.**stopPropagation**().

```
<p>Test this <span>Span</span> and this <strong>Text</strong></p>
var paragraph = document.querySelector('p'),
span = paragraph.querySelector('span');
paragraph.addEventListener('click', function () { console.log('Clicked!'); });
span.addEventListener('click', function (e) { e.stopPropagation() });
// В данном случае клик на p и strong сработает, а при клике на span ничего не произойдёт.
```

События. Погружение.

Кроме всплытия, существует ещё одна фаза существования события в документе - фаза **погружения**.
Это самая первая фаза, с которой начинается жизнь события.
Она противоположна фазе всплытия.
Чтобы “ловить” событие на этой фазе, нужно в **addEventListener** передать третий аргумент, равный **true**:

element.**addEventListener**("click", handler, **true**);

Пример здесь

Подробней о событиях здесь и здесь (тут упоминается то чего мы не проходили).

События. Делегирование.

Обработка событий по шаблону “**делегирование**” означает обработку событий на родительском элементе с проверкой, произошло ли событие на конкретном дочернем узле.

```
<ul>
        <li>1 <span>Text<span/> </li>
        <li><a>2</a></li>
        <li>3</li>
</ul>

var ul = document.querySelector(‘ul’);
ul.addEventListener(‘click’, function (e) {
    if (e.target.tagName === ‘LI’) { console.log(‘Li was clicked!’); }
});

// теперь клик по любому из li сработает, хотя обработчик один
```

События. Делегирование.

Если мы хотим, чтобы клик сработал на li и всех вложенных тегах, нужно уточнить условие:

```
var ul = document.querySelector(‘ul’);
ul.addEventListener(‘click’, function (e) {
    var isLi = e.target.tagName === ‘LI’;
    if (isLi || e.target.closest(‘li’) ) {
      console.log(‘Li was clicked!’);
    }
});
// теперь клик по любому из li или вложенному элементу сработает
```

События. Делегирование.

Также можно повесить обработчик на отдельный класс или атрибут элемента. Обработчик вешается на весь документ или также как в предыдущем примере на родительский элемент, внутри обработчика проверяется, на каком элементе “**выстрелило**” событие. Если на элементе с нужным нам атрибутом - выполняем код.

```
<p>Text with <a href=”#”>Link</a> and <a href=”#” class=”link”>Other Link</a></p>
<article>Some article with <a href=”#” class=”link”>link</a></article>

document.addEventListener(‘click’, function (e) {
    if (e.target.className !== ‘link’) return;
    console.log(‘Element ’ + e.target.tagName + ‘ was clicked!‘);
});
```

События. Мобильные события.

Основная логика взаимодействия с тачскрином сведена к трем событиям:

touchstart: прикосновение к DOM element (аналог mousedown).
touchmove: движение пальца по DOM element (аналог mousemove).
touchend: палец убран с DOM element (аналог mouseup).

Объект события event включает в себя три списка точек прикосновения (списки пальцев):
touches: список всех точек прикосновения на экране.
targetTouches: список точек на текущем элементе.
changedTouches: список пальцев, участвующих в текущем событии.

Статьи ссылка 1 , ссылка 2 , ссылка 3

События. Код к задачам.

```
<button id="btn-msg" data-text=”Custom message”>Show message</button>
<button id="btn-generate">Generate item</button>
<p><strong id="tag">Tag:</strong></p>
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

События. Задачи.

1.По нажатию на кнопку "**btn-msg**" должен появиться алерт с тем текстом который находится в атрибуте **data-text** у кнопки.

2. При наведении указателя мыши на "btn-msg", кнопка становится красной; когда указатель мыши покидает кнопку, она становится прежнего цвета. Цвет менять можно через добавление класса.

3. При нажатии на любой узел документа показать в элементе с **id=tag** имя тега нажатого элемента.

4. При нажатии на кнопку **btn-generate** добавлять в список **ul** элемент списка **Li** с текстом **Item + порядковый номер Li по списку,** т.е Item 3, Item 4 и т.д

5. Из проекта todo реализовать редактирование задачи.

События. Задачи.

Код к задаче здесь

6. Реализовать примитивный дропдаун. Изначально все **dropdown-menu** скрыты через класс **.d-none**. При клике на **dropdown-item** должен отображаться блок **dropdown-menu** который вложен именно в тот **dropdown-item** на котором произошел клик. При повторном клике на этот же **dropdown-item** блок **dropdown-menu** должен закрыться. При клике на любой другой **dropdown-item** уже открытый **dropdown-menu** должен закрываться а на тот который кликнули открываться.

События. Задачи.

7. Из презентации “**Занятие 7 - Манипуляция DOM. Работа с атрибутами.**” дополнить функционал для таблицы из задачи 6. Создать кнопку которая будет при клике сортировать пользователей по возрастанию и убыванию поля balance при этом в кнопке должна показываться стрелка в какую сторону сейчас отсортированы пользователи. Иконки можете взять с font awesome, в качестве фреймворка использовался bootstrap.