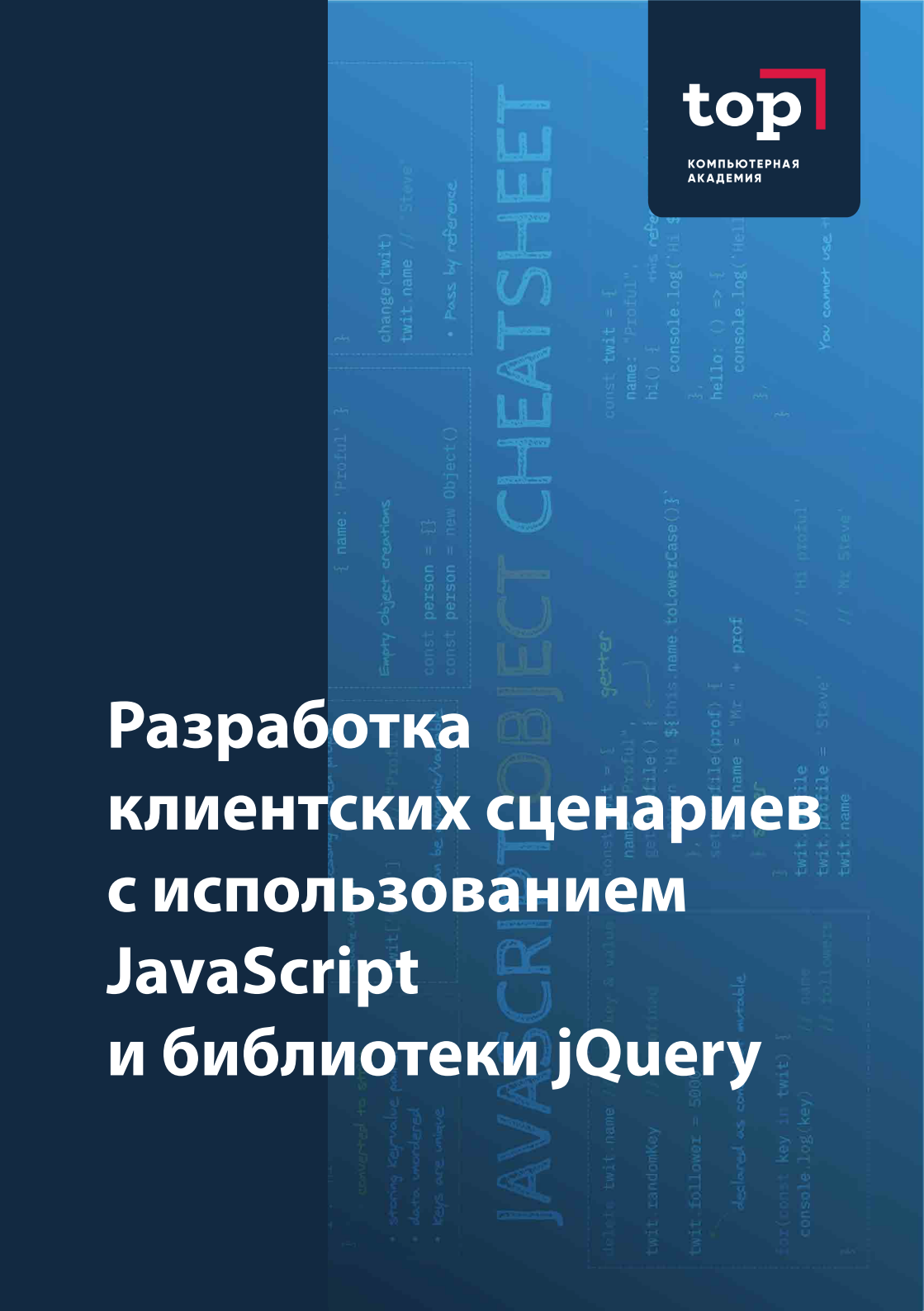
top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Разработка клиентских сценариев с использованием JavaScript и библиотеки jQuery



Unit 3

Обработка событий

Содержание

Что такое событие и обработчик события? 4
Обработка событий в сценариях 10
Объект event и его свойства 16
Управление стилями элементов web-страницы 24
События интерфейса пользователя 36
События жизненного цикла....56
Обработчики событий по умолчанию (стандартные обработчики), запрет вызова стандартного обработчика 75
Домашнее задание....83

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

Что такое событие и обработчик события?

Представьте себе, что Вы общаетесь с собеседником и в это время звонит телефон. Звонок важный и на него необходимо ответить. Поскольку вести два разговора одновременно невозможно, Вы приостанавливаете текущую беседу и говорите по телефону.

Тема телефонного разговора, наверняка, будет отличаться от предмета вашей беседы, то есть Вам надо «переключиться» на другую тему, вспомнить детали, события, имена и т.п. После окончания телефонного разговора Вы возвращаетесь к предыдущей беседе с того места, в котором она была прервана звонком, «переключившись» обратно к теме беседы, ее деталям и особенностям.

Аналогичным образом устроен компьютер. У него есть центральный процессор, который может в один момент времени выполнять только одну команду. Однако, ему постоянно «звонят» — двигается мышь, нажимаются кнопки на клавиатуре, приходят данные по сети и т.д. Процессору приходится «отвечать» на эти «звонки» — прекращать работу с одной программой, переключаться на обработку другой, и снова возвращаться к предыдущей программе. Такие «звонки» получили название событий.

Понятие «событие» имеет ряд оттенков и особенностей для различных областей использования. Для того чтобы разобраться с этими особенностями, рассмотрим детальнее, какие процессы будут происходить в компьютере, когда Вы включаете блокнот и печатаете в нем некоторую заметку.

Блокнот является обыкновенной программой, выполняемой операционной системой. Процессор компьютера исполняет инструкцию за инструкцией, обеспечивая отображение окна блокнота, его элементов, а также набранного ранее текста.

В некоторое время на клавиатуре нажимается кнопка. Процессор компьютера прекращает выполнять коды программы «блокнот», сохраняет необходимые данные и переходит к обработке сигнала от клавиатуры. Для этого в памяти компьютера существует специальная программа (драйвер), определяющая, какая кнопка была нажата, и что при этом необходимо сделать. Некоторые кнопки отвечают за набор символов, тогда как другие могут управлять громкостью динамиков или яркостью монитора. Предположим, что была нажата символьная кнопка. В таком случае определяется, какой символ соответствует кнопке, и данная информация сохраняется в системном буфере обмена.

После того как сигнал от клавиатуры будет обработан, процессор вернется к выполнению программы «блокнот», восстановит сохраненные ранее данные, и напечатает символ, который был сохранен в буфере при работе драйвера клавиатуры.

А что значит напечатает? Программа поместит код символа в необходимое место видеопамяти. Затем процессор, аналогично описанному выше способу, переключится на задачу передачи видеоданных на монитор, который уже и отобразит новый символ.

Процессы переключения процессора на выполнение различных задач называется термином «прерывание»

(*англ.* — interruption). Прерывания — это события системного уровня, «звонки» от подключенных к компьютеру устройств.

Прерывания обеспечивают работу клавиатуры, мыши, дисков, монитора и других периферийных устройств компьютера. Для разных прерываний предусмотрены разные программы, отвечающие за их обработку. Они так и называются «обработчики прерываний». Эти программы загружаются при включении компьютера и старте операционной системы.

Обработка прерываний происходит быстро, создавая ощущение, что мы просто печатаем текст в блокноте. Тем не менее, пока мы печатаем происходит множество прерываний — идут часы, мигает курсор, работают диски, проверяется электронная почта и т.п.

Идея прерываний, заложенная в основу работы компьютера, нашла применение и в разработке прикладных программ, в том числе веб-страниц. Только в этой области вместо прерывания используется термин «событие» (*англ.* — event).

Событие — это некоторое происшествие, запускающее специальную программу — обработчик события. Вернее, правильнее сказать «подпрограмму», поскольку обработчик события не является отдельной программой (как обработчик прерывания), а входит в состав одной, основной программы. Обычно, обработчики событий — это отдельные функции (в программном понимании функции, как именованной подпрограммы).

Понятие события тесно связано с понятием сообщения (*англ.* — message). Их даже иногда смешивают,

определяя событие как сообщение. Тем не менее, между ними есть определенная разница. Когда Вы слышите звук звонящего телефона — Вы получаете «сообщение». Это сообщение сопровождается дополнительной информацией — на экране телефона отображается номер звонящего или его имя, пусть для примера, Адам. Событием становится Ваш вывод: «мне звонит Адам». Разница тонкая: сообщение — это звук звонка, а событие — указанный вывод. Однако, требуются определенные действия для того, чтобы, услышав звук звонящего телефона, прийти к выводу «мне звонит Адам».

Рассмотрим соотношение сообщений и событий более подробно. Механизм обмена сообщениями является неотъемлемой частью операционной системы. Различные программы могут посылать друг другу сообщения, сообщениями могут обмениваться части одной программы, сообщения могут поступать в программу прямо от операционной системы. Каждая программа имеет у себя очередь сообщений, которые обрабатываются одно за другим или накапливаются, если не успевают обрабатываться. Как если во время одного телефонного разговора Вам поступит еще один «параллельный» звонок.

Операционная система имеет свой набор стандартных сообщений, например, движение мыши или нажатие ее клавиш, переключение между окнами, изменение их размеров или положения, нажатие кнопок клавиатуры и т.д. Прикладные программы могут создавать другие (дополнительные) сообщения. Например, браузеры дополняют системные сообщения собственным набором, необходимым именно для веб-страниц.

Обработка сообщения, принятие решение о его типе, анализ дополнительных данных уже называется событием. Когда доходит очередь до обработки определенного сообщения, тогда и возникает событие, — программа определяет, какую функцию (подпрограмму) необходимо запустить для обработки данного события. Если для события обработчика нет, то такое сообщение просто игнорируется и удаляется из очереди. Можно сказать, что сообщение — это понятие системное, межпрограммное, тогда как событие — понятие внутреннее, действующее только для данной программы.

Возможно, более существенную разницу между сообщением и событием можно привести на следующем примере. Вы ответили на телефонный звонок, а звонящий Адам попросил пригласить к телефону Вашего собеседника, с которым Вы говорили до звонка. В данном случае звонок является сообщением, основным событием — принятое решение «мне звонит Адам», а дополнительным событием становится Ваше действие, приглашающее собеседника к телефону.

С точки зрения веб-программиста, детальный анализ процесса формирования сообщений можно пропустить. Эти сведения больше нужны для разработчиков системных программ. К тому же, как уже отмечалось выше, браузер видоизменяет системный набор сообщений, поэтому перейдем к рассмотрению событий, которые будут возникать в веб-страницах.

В JavaScript принято обращаться к событиям по их именам (в операционной системе — по кодам). Имена содержат в себе подсказку о происхождении данного

события. Например, событие с именем «click» возникает, когда по элементу совершают щелчок мыши.

Полный перечень возможных событий достаточно большой. Ознакомиться с ним можно, например, на странице. В качестве обобщения события можно условно разделить на несколько групп:

- События интерфейса пользователя;
- События жизненного цикла;
- Индивидуальные события.

События интерфейса пользователя возникают за счет активности пользователя веб-страницы. Это движения курсора мыши, нажатие кнопок клавиатуры и кнопок, нарисованных на странице, прокрутка страницы или содержимого элемента и т.д.

События жизненного цикла посылаются элементам при их создании, загрузке, в том числе ошибках загрузки, при получении системных сообщений, переходу в online или offline режим, запуске анимации и т.п.

Индивидуальные события характерны только для определенных объектов и для объектов другого типа не используются. Примером может быть событие «pause», возникающее при остановке воспроизведения (паузе) медиаконтента. Очевидно, что это событие касается только медиа-контейнеров. Событие «input» возникает, когда элемент получает ввод от пользователя, то есть в этот элемент добавляется текст, который печатается на клавиатуре. Такое событие актуально только для элементов, в которых можно что-то вводить (печатать). Подобные события есть и у других групп элементов.

Обработка событий в сценариях

Для всех стандартных событий у элементов страницы предусмотрены обработчики. Традиционно их имена формируются из префикса «on» и имени события. Например, обработчик события щелчка мыши «click» будет иметь название «onclick».

Существует несколько способов определить тело для обработчика событий. Первый — это указать его как HTML-атрибут при объявлении элемента. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_1.html*)

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Events</title>
    <style>
        div {
            background: navy;
            height: 50px;
            width:  100px;
        }
    </style>
</head>
<body>
    <div onclick="alert('DIV have been clicked')"></div>
</body>
</html>
```

Основным содержимым страницы является блок (div), для которого в заголовочной части указаны стилевые атрибуты: ширина, высота и фоновый цвет. В теле документа, при объявлении блока указан атрибут «onclick», являющийся обработчиком события щелчка мыши. В значении данного атрибута указываются инструкции JavaScript, которые будут выполнены при наступлении события, то есть при щелчке мыши.

Обратите внимание на необходимость чередования кавычек разного типа для всего значения атрибута «onclick» и для текста сообщения диалогового окна «alert». Подобные ситуации возникают довольно часто при внедрении JavaScript инструкций:

- Во-первых, сам код обработчика должен быть заключен в кавычки: <div onclick="…".
- Во-вторых, текст сообщения для команды «alert» также требует кавычек: alert('…').

Если применять кавычки одного типа, то возникнет ошибка, так как внутренние кавычки будут закрывать внешние (для наглядности добавлены дополнительные отступы):

```
<div onclick="alert("DIV have been clicked")" ->
<div onclick="alert("   DIV have been clicked   ")"
```

Для более сложных инструкций современный стандарт языка допускает применение трех типов кавычек. Напомним их:

- Одиночные прямые кавычки: '…';
- Двойные кавычки: "…";
- Одиночные обратные кавычки: `…`.

Для текстовых выражений все три типа кавычек полностью эквивалентны. Отличия заключаются в том, что в обратных кавычках можно подставлять значения переменных, например, предположим, что в скрипте объявлена переменная «x=10», тогда следующие выражения приведут к соответствующим результатам:

```
"x=${x}"   ->   x=${x}
'x=${x}'   ->   x=${x}
'x=${x}'   ->   x=10
```

Сохраните созданный файл и откройте его в браузере. Наведите курсор мыши на синий блок и щелкните по нему левой кнопкой мыши. В результате должно появиться сообщение, указанное в обработчике (рис. 1).

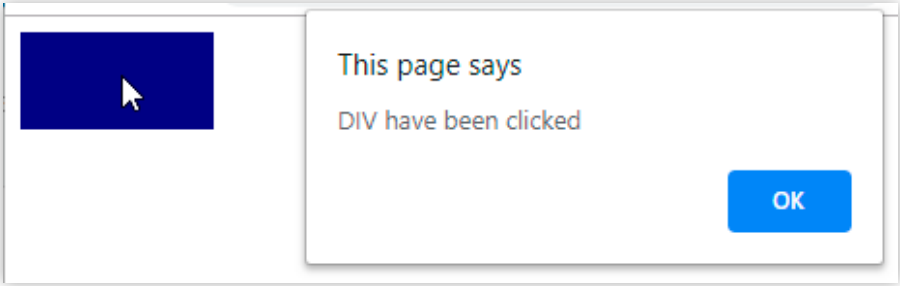


Рисунок 1

Выполните щелчок, отведя курсор мыши за пределы блока. Убедитесь, что в этом случае сообщение не появляется. Это свидетельствует об адресации событий. Кроме того, что событие возникает само по себе как факт «системой зафиксирован щелчок мыши», это событие имеет цель (*англ.* — target) — элемент, для которого

это событие предназначено. Другие объекты это событие не получают, в чем мы убедились, щелкая мышью за пределами блока.

Второй способ определить обработчик события — это указать одноименный метод («onclick» в нашем примере) в скриптовой части кода. Внесите в созданный файл следующие изменения (*код с изменениями доступен в папке Sources, файл JS_3_2.html*):

```
<body>
    <div id="clickableDiv"></div>
    <script>
        clickableDiv.onclick=function(){
            alert('DIV have been clicked')
        }
    </script>
</body>
```

Блоку (div) присваивается идентификатор id="clickableDiv". По этому идентификатору данный блок становится доступным в скриптовой части, где для него указывается обработчик события «clickableDiv.onclick». В качестве значения задается функция, выполняющая те же действия, что и в предыдущем примере.

Сохраните изменения, откройте файл в браузере или обновите открытую страницу. Убедитесь в том, что события обрабатываются точно также, как и ранее.

Третий способ задать обработчик события для элемента — это применить специальный метод «addEventListener». Внесите следующие изменения в скриптовую часть документа (*код с изменениями доступен в папке Sources, файл JS_3_3.html*):

```
<script>
    clickableDiv.addEventListener("click", function(){
       alert('DIV have been clicked')
    })
</script>
```

Блок все также управляется при помощи своего идентификатора «clickableDiv», только вместо метода «onclick» вызывается метод «addEventListener». В качестве аргументов для него передаются два значения — имя события и функция-обработчик. Обратите внимание, в данном случае передается имя события «click» (без префикса «on», используемого для имени обработчика). Тело функции обработчика сохранено из предыдущего примера.

Сохраните изменения, откройте файл в браузере или обновите открытую страницу. Убедитесь в том, что работоспособность кода не изменилась.

Любым из описанных способов можно создать обработчик для произвольного события. Принципиальных отличий в различных способах нет. Первый способ, определяющий тело обработчика непосредственно в HTML теге, может подойти для небольших программных инструкций, не загромождающих разметку элементов. Для более объемных кодов удобнее отделять их в самостоятельные функции и подключать к элементам вторым или третьим способом. Единственное, что при этом следует отметить, так это то, что смешивать несколько способов не допускается. Если определить обработчик в HTML теге, а затем в скриптовой части снова указать функцию для того

же события, то новое значение будет использовано вместо старого. Другими словами, новое тело обработчика сотрет старое и займет его место. Если обработчик устанавливается несколько раз, то актуальным будет тот, который установлен последним.

Объект event и его свойства

При обработке события часто бывает необходима дополнительная информация, связанная с возникновением самого события. Например, событие нажатия клавиши на клавиатуре возникает при нажатии любой клавиши. Очевидно, что программы должны по-разному реагировать на нажатия разных клавиш, а значит в обработчик события эта информация должна быть передана.

При обработке системного сообщения и создании программного события браузер формирует специальный объект «event», в котором собираются все данные о событии. В теле обработчика этот объект может использоваться для получения этой дополнительной информации.

Информация в объекте «event» зависит от типа события, которое обрабатывается. Логично, что для событий мыши не нужны данные о кнопке клавиатуры, а для событий клавиатуры не нужны координаты указателя мыши.

В то же время, объект «event» имеет свойства, определяемые для всех типов событий. Одним из таких свойств является целевой объект события (target). Выше мы уже отмечали, что любое событие имеет своего адресата. Рассмотрим пример, иллюстрирующий работу с объектом «event» и его свойством «target». Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_4.html*). Стилевые определения могут быть скопированы из предыдущих примеров.

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Events</title>
    <style>
        div {
            background: navy;
            height: 50px;
            width:  100px;
        }
    </style>
</head>
<body onclick = "out.innerText =
                 'Click detected over: ' +
                  event.target.nodeName">
    <div></div>
    <p id="out"></p>
</body>
</html>
```

Аналогично с рассмотренными ранее кодами, в теле документа располагается блок, цвет и размеры которого задаются стилями в заголовочной части. Дополнительно после блока размещается абзац (параграф) с идентификатором «out». Он будет служить нам контейнером для вывода сообщений.

В отличие от предыдущих примеров, обработчик события «onclick» определен в теге «body». Это позволяет перехватывать все события, относящиеся к данному документу. В теле обработчика формируется сообщения из текста «Click detected over:», к которому добавляется имя целевого объекта события «event.target.nodeName».

Сохраните документ и откройте его в браузере. Выполните щелчки мыши по блоку и за его пределами. Обратите внимание на изменения сообщения о целевом объекте. Выполните щелчок над самим текстом сообщения. Убедитесь, что целевой объект меняется и в этом случае.

Рисунок 2

Рисунок 3

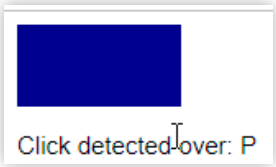


Рисунок 4

Как видно из примера, объект «event» не требует специального описания. Он создается браузером и передается в обработчик, уже заполненный необходимыми данными. Для того чтобы узнать, какие данные доступны для конкретного события, можно вывести объект «event» в консоль разработчика. Дополните код обработчика события инструкцией «console.log(event)», обработчик должен принять следующий вид:

```
<body onclick = "out.innerText =
                 'Click detected over: ' +
                 event.target.nodeName;
                 console.log(event)">
```

Рисунок 5

Сохраните изменения, обновите страницу браузера. Откройте в браузере панель разработчика (кнопка F12), переключитесь на вкладку «Console» (если она сразу не активна). Переведите курсор мыши на блок и выполните щелчок. В консоли раскройте детали сообщения. Результат должен быть подобен приведенному на рисунке 5 (возможны некоторые отличия при использовании разных браузеров, пример соответствует браузеру «Chrome»)

Не будем детально описывать значения всех полей, остановимся на тех, которые наиболее часто находят применение в веб-разработке. Конечно же таковыми являются координаты курсора мыши в момент совершения щелчка и сведения о целевом объекте.

В объекте «event» можно обнаружить несколько различных пар значений для координат. Среди них есть стандартизированные:

- screenX, screenY — координаты курсора относительно экрана монитора
- pageX, pageY — координаты относительно начала веб-страницы
- clientX, clientY — координаты относительно клиентской части окна браузера.

Дополнительные пары координат являются экспериментальными или не стандартизированными. При их использовании необходимо убедиться в поддержке данным типом браузера. Можно даже сказать, что их использование не рекомендуется, из-за отличий для разных браузеров.

- layerX, layerY — координаты относительно целевого элемента, имеющего позиционирование (стилевое

свойство «position» которого имеет значение, отличное от «static»)

- offsetX, offsetY — координаты относительно любого целевого элемента.
- x, y — псевдонимы для clientX, clientY

Как видно из анализа числовых данных, браузер «Chrome» в качестве layerX и layerY использует значения из пары (x, y), что не соответствует целевому элементу. Если запустить данный пример в браузере «Firefox», то можно убедиться, что в нем для координат offsetX и offsetY указываются нули независимо от положения курсора. До тех пор, пока не будет установлен стандарт для указанных выше полей, их использование крайне нежелательно.

Отличия между способами отсчета стандартных координат иллюстрирует следующий рисунок:

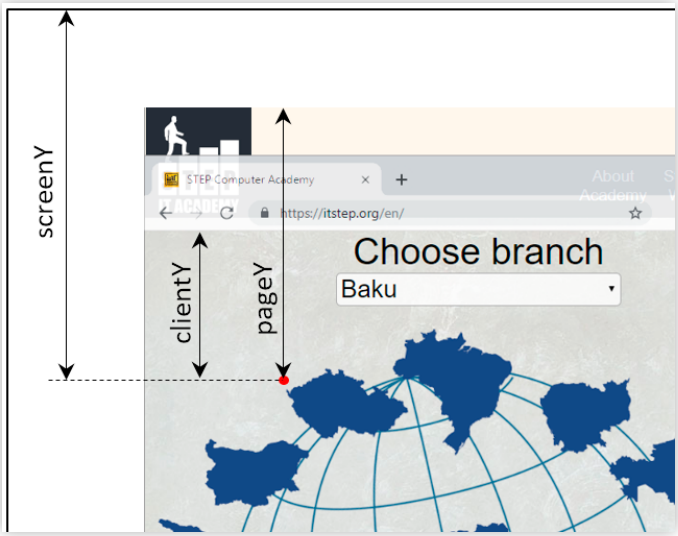


Рисунок 6

Черными линиями на рисунке 6 показаны границы экрана (монитора). На экране изображено окно браузера, в котором открыт сайт. Этот сайт прокручен, поэтому в окне отображается лишь его часть. Скрытая часть также приведена на рисунке, хотя в реальной ситуации её, естественно, не видно. Предположим, что событие щелчка мыши произошло в точке, отмеченной красной точкой.

Координата screenY точки будет отсчитана от границ самого экрана. Это системная координата, не привязанная ни к какому из окон. Координата clientY берет отсчет от окна браузера, точнее, от его клиентской части, которая не учитывает адресную строку, заголовок вкладки, кнопки управления размерами окна браузера. Значение pageY определяется относительно начала веб-страницы. В данном случае это значение будет больше, чем clientY, т.к. начало страницы находится выше края окна за счет прокрутки. Если страница не прокручена, то значения координат clientY и pageY совпадают.

Полностью аналогично определяются координаты точки события screenX, clientX и pageX, только их отсчет происходит по горизонтальной оси.

Следует отметить еще одну особенность использования объекта-события. Объект «event» автоматически связывается с параметром функции-обработчика, то есть служит ее аргументом. В том случае, когда обработчик события определяется вне HTML-тега, для получения данных о событии можно указать параметр функции обработчика. Не обязательно использовать для него имя «event», чаще всего для экономии места его обозначают просто как «e».

```
element.onclick = function(e){…}
element.addEventListener("click", function(e){…})
```

В теле функции параметр «e» является тем самым объектом «event», который сопровождает событие. Координаты точки события можно узнать применяя запись, на подобие «e.pageX», а целевой объект — «e.target».

Как уже отмечалось выше, данные в объекте «event» зависят от типа события, которое передается на обработку. В данном разделе мы рассмотрели только одно событие, сопровождающее щелчок мыши. О свойствах «event», которые характеризуют другие события, мы будем говорить дальше при детальном рассмотрении соответствующих событий. В любом случае, проверить состав объекта «event» можно путем вывода его в консоль разработчика.

Задание для самостоятельной работы. Реализуйте функциональность рассмотренного выше примера по определению целевого объекта события при помощи второго и третьего способов установки обработчика события (element.onclick и element.addEventListener).

Управление стилями элементов web-страницы

В предыдущих упражнениях в ответ на поступление события мы использовали текстовые сообщения, выводя их прямо на страницу или в диалоговые окна. Гораздо больший спектр возможностей реагирования на события предоставляет управление стилями — цветом, размером, положением элементов и многим другим. Это позволит более гибко обеспечить взаимодействие нашей веб-страницы с посетителем.

При помощи управления стилями можно создать ощущение, что сайт каким-либо образом «откликается» на действия пользователя. Если он забыл заполнить нужное поле, то сайт не просто выдаст сообщение, напоминающее о необходимости ввода данных, а еще и выделит пропущенный элемент, например, красной рамкой. Также управление стилями может обеспечить эффекты анимации, делая страницу динамичной и привлекательной.

Для иллюстрации принципа управления стилями модифицируем рассмотренный в предыдущих разделах пример следующим образом: при щелчке мышью синий блок должен перемещаться в точку, где находится курсор мыши.

Приведем текст документа и далее рассмотрим принцип его работы. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_6.html*)

```
<!doctype html />
<html>

<head>
    <meta charset="utf-8" />
    <title>Events</title>
    <style>
        body {
        position: relative;
        height: 2000px;
    }
        div {
            background: navy;
            height: 50px;
            width:  100px;
            position:absolute;
        }
    </style>
</head>

<body onclick="moveStranger(event)">
    <div id="stranger"></div>
    <script>
        function moveStranger(e){
            stranger.style.left=e.pageX+'px';
            stranger.style.top =e.pageY+'px'
        }
    </script>
</body>
</html>
```

Для того чтобы поместить блок в заданную позицию можно воспользоваться двумя технологиями:

а) указать отступы (margin) для блока или

б) задать блоку абсолютное позиционирование и указать координаты (left, top).

Если блок на странице один, то методы полностью эквивалентны. Однако, когда на странице есть другие элементы, изменение отступов блока (вариант а) приведет к «толканию» соседних элементов. Поэтому, хотя блок у нас на странице будет один, используем для практической реализации вариант б).

В стилях блока, по сравнению с предыдущими примерами, добавлено определение «position:absolute». Для того чтобы использовать абсолютные координаты элемента, необходимо указать от чего эти координаты будут отсчитываться. Контейнером для блока является тело документа «body», поэтому указываем в его стилях «position: relative». Дополнительно укажем большую высоту для тела «height: 2000px», это даст нам возможность прокручивать страницу вверх-вниз для проверки правильности выбора координат из объекта-события (мы помним из предыдущего раздела, что их есть как минимум три пары).

В теле документа все так же присутствует один блок, теперь с идентификатором «stranger». По этому имени блок будет доступен в скриптах. При определении тела (body) устанавливается обработчик события. Для сокращения записей в HTML-теге использовано имя дополнительной функции «moveStranger», в которую передается объект «event» (он нужен для определения координат). В скриптовой части приводится тело функции «moveStranger», принимающей параметр «e».

При запуске функции происходит изменение стилевых определений блока «stranger». Как видно из кодов, доступ к этим стилям обеспечивает объект «stranger.style». Для стилевого атрибута «stranger.style.left» формируется

значение из координаты события «e.pageX» и единицы измерения ('px'). Помним, что единицы измерения обязательны для указания длин в стилях, тогда как в объекте «event» присутствуют только числовые значения. Аналогично устанавливается стиль «stranger.style.top» из координаты «e.pageY».

Сохраните файл и откройте его в браузере. Щелкните мышью в произвольной точке страницы, убедитесь, что блок перемещается в точку щелчка. Прокрутите страницу и повторите щелчок. Блок все также должен следовать за курсором мыши. Это свидетельствует о правильном выборе координат из объекта «event». В качестве проверки можете использовать другую пару координат из объекта «event» (например, clientX, clientY) и убедиться, что при прокрутке окна работа нарушается. Верните коды к исходному состоянию.

Присмотритесь внимательно к взаимному размещению блока и курсора мыши (рис. 7).

Рисунок 7

между острием курсора и углом блока присутствует определенный отступ. Это внешний отступ (margin) тела документа (body). Обычно, он устанавливается в значение «8px» и создает небольшую рамку, отделяющую страницу от границ окна браузера. В нашем случае этот отступ

приводит к неправильному отсчету координат, т.к. начало страницы и границы окна изначально не совпадают.

Добавьте стилевое определение «margin:0» для элемента «body» (для нулевого значения единицы измерения разрешается не указывать). Сохраните файл и обновите страницу в браузере. Убедитесь, что блок стал следовать за курсором мыши более точно, совпадая левым верхним углом с острием указателя.

Как нам уже известно, любой элемент, созданный HTML разметкой, имеет свое представление в виде программного объекта, доступного по названию своего идентификатора (атрибута «id»). Пусть, для примера, в нашем документе присутствует элемент с атрибутом «id="element"».

Все стилевые атрибуты этого элемента собраны в коллекции «element.style». Как и любая другая коллекция в JavaScript, она предусматривает два способа доступа к своим компонентам — при помощи объектного синтаксиса

```
element.style.left
```

или при помощи синтаксиса доступа к массивам

```
element.style["left"]
```

Оба эти способа эквиваленты, использовать можно любой из них по Вашему предпочтению. Однако, отличия все же наблюдаются для составных имен стилевых атрибутов.

В CSS принято, что знак «-» является допустимым в именах атрибутов, например, «background-color». При

этом в JavaScript данный знак обозначает арифметическую операцию (вычитание) и не может быть частью имени объекта или его свойства. В то же время, на ключи массива данное ограничение не распространяется. Это значит, что запись

```
element.style["background-color"]
```

является допустимой и правильной, тогда как

```
element.style.background-color
```

представляет собой операцию вычитания, а не доступ к полю «background-color» объекта.

Для обеспечения работы с составными именами стилевых атрибутов в объектном синтаксисе используется стиль «lowerCamelCase», применяемый и для других имен в JavaScript. Знаки «-» из названий стилевых атрибутов убираются и каждое новое слово (кроме первого) начинают с заглавной буквы. При использовании синтаксиса доступа к коллекции, как к массиву, названия атрибутов точно такие же, как и в CSS определениях. В следующей таблице приведено несколько примеров использования коллекции «element.style» при помощи различных синтаксисов

Синтаксис массивов	Синтаксис объектов
element.style["background-color"]	element.style.backgroundColor
element.style["margin-top"]	element.style.marginTop
element.style["list-style-type"]	element.style.listStyleType
element.style["border-top-right-radius"]	element.style. borderTopRightRadius

Любые изменения, внесенные в коллекцию «element.style», сразу же поменяют стиль элемента, независимо от того, какой синтаксис был использован. Дополнительных действий по фиксации новых стилей совершать не нужно — изменения сразу же вступают в силу после записи в коллекцию стилей. То есть инструкция

```
element.style["background-color"] = "tomato"
```

сама-по-себе поменяет фоновый цвет элемента, никаких обновлений или перерисовок окна запускать не требуется. В коллекции «element.style» предусмотрены все возможные стилевые атрибуты, которые доступны и в CSS.

Продемонстрируем способ программного управления фоновым цветом блока на следующем примере. Мы хотим, чтобы щелчок мыши на блоке приводил к изменению его цвета. Уточним, что цвет должен формироваться случайным образом без какой-либо предварительной закономерности.

Приведем текст документа и далее рассмотрим принцип его работы. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_7.html*)

```
<!doctype html />
<html>

<head>
    <meta charset="utf-8" />
    <title>Random color</title>
    <style>
        div {
```

```
            background: navy;
            height: 255px;
            width:  255px;
        }
    </style>
</head>

<body >
    <div id="colorBlock"></div>
    <script>
      colorBlock.addEventListener("click", function(){
        colorBlock.style.backgroundColor =
             "rgb(" +
             Math.round(255*Math.random()) +
             "," +
             Math.round(255*Math.random()) +
             "," +
             Math.round(255*Math.random()) +
             ")" ;
        })
    </script>

</body>
</html>
```

Основу страницы все так же представляет один блок, изначально синего цвета. Для доступа к нему при помощи JavaScript, ему указан атрибут «id="colorBlock"».

В скриптовой части документа для данного блока устанавливается обработчик события щелчка мыши при помощи метода «colorBlock.addEventListener». Отсутствие специального алгоритма для формирования цвета позволяет нам использовать в качестве обработчика события функцию без параметра «function()».

В теле функции-обработчика мы формируем случайный цвет при помощи CSS функции «rgb». Для этого мы вызываем генератор случайных чисел «Math.random()». Он генерирует дробные случайные числа в диапазоне от 0 до 1.

Как нам известно из курса HTML/CSS, в качестве цветовых компонентов для «rgb» нужны целые числа от 0 до 255, поэтому полученные от генератора случайные числа мы умножаем на 255 и округляем функцией «Math.round». Инструкция получения целого случайного числа, подходящего для цветового компонента, приобретает вид

```
Math.round(255*Math.random())
```

Далее, три таких инструкции объединяются в одну строку. Ее конечный вид должен соответствовать формату «rgb(127,201,57)», то есть между числами добавляются запятые, а в начале и конце — круглые скобки. Вместо чисел вставлены описанные выше выражения.

Результирующая строка помещается в поле стилевой коллекции блока, ответственного за фоновый цвет «colorBlock.style.backgroundColor».

Сохраните файл и откройте его при помощи браузера. Наведите курсор мыши на блок и совершите щелчок левой кнопкой мыши. Убедитесь в том, что цвет блока меняется при каждом щелчке случайным образом.

В качестве следующего примера модифицируем созданную программу. Зададим алгоритм формирования цвета: пусть красный компонент всегда будет равен среднему значению (127), зеленый компонент будет зависеть

от координаты «X» курсора мыши — если курсор находится возле левого края блока, то его значение близко к нулю, если возле правого, то максимальное. Синий компонент аналогичным образом будет зависеть от координаты «Y» курсора мыши.

Для большей оперативности потребуем, чтобы изменения цвета происходили при каждом перемещении курсора без необходимости нажатия кнопок мыши.

Проведем анализ новых условий. Во-первых, вместо случайных чисел необходимо использовать сведения о положении курсора. Соответственно, обработчик события должен принимать параметр, отвечающий за дополнительные сведения о событии (объект «event»).

Во-вторых, обработчик необходимо привязать к другому событию, возникающему при любом перемещении мыши. Это событие имеет имя «mousemove».

В-третьих, воспользуемся некоторыми упрощениями для исключения сложных расчетов. Нам известно, что максимальное значение цветового компонента — это число «255». Ограничим размеры блока этими значениями. Обратим внимание, что блок уже имеет нужный нам размер. Однако, как мы разбирали в одном из предыдущих примеров, необходимо еще убедиться, что сама страница (и блок вместе с ней) не будет смещена относительно окна браузера, иначе координаты курсора в блоке будут отличаться от желаемого диапазона. Для того чтобы предотвратить смещения, укажем стилевой атрибут «margin: 0» для тела документа «body».

Внесите описанные изменения в документ. Результат должен быть подобен приведенному далее коду (*код*

с изменениями также доступен в папке Sources, файл JS_3_8.html)

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Changing color</title>
    <style>
        body {
            margin: 0;
        }
        div {
            background: navy;
            height: 255px;
            width:  255px;
        }
    </style>
</head>
<body >
    <div id="colorBlock"></div>
    <script>
        colorBlock.addEventListener("mousemove",
                                    function(e){
            colorBlock.style.backgroundColor =
                 "rgb(127," +
                 e.pageX +
                 "," +
                 e.pageY +
                 ")" ;
        })
    </script>
</body>
</html>
```

Как видно из кода, подбор размеров блока, и коррекция смещений позволяет в качестве цветовых компонент

указать координаты курсора «e.pageX» и «e.pageY» без дополнительного их пересчета. Для смещенного блока или при других его размерах необходимо будет обрабатывать эти значения.

Сохраните изменения и обновите страницу браузера. Наведите курсор мыши на блок, убедитесь, что его цвет изменяется в зависимости от положения курсора (рис. 8). Поскольку алгоритм изменения цвета стал детерминированным, результаты должны быть одинаковыми для различных браузеров и при разных запусках, чего не наблюдалось при использовании случайных чисел.

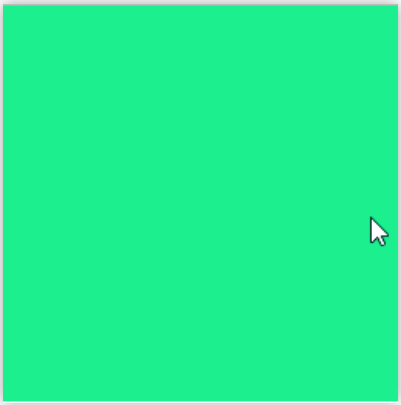


Рисунок 8

Задание для самостоятельной работы. Создайте программу, которая будет увеличивать или уменьшать размеры блока — левая верхняя точка блока всегда будет находиться в начале страницы (в левом верхнем углу), а другая точка будет задаваться курсом мыши. При щелчке мышью блок должен заполнить область от начала страницы до текущего положения курсора.

События интерфейса пользователя

События интерфейса пользователя (User Interface, UI events) представляют собой группу событий, возникающих вследствие активности человека, просматривающего нашу веб-страницу (пользователя). Можно охарактеризовать эту группу событий «от обратного» — если пользователь отойдет от компьютера, то эти события перестанут поступать в программу.

В следующей таблице приводится описание основных событий интерфейса пользователя. Полный перечень и детальное описание можно также посмотреть в стандартах, например, по ссылке. События имеют имена, отражающие смысл или условие их возникновения, что облегчает их использование и запоминание.

Событие	Условие возникновения
События мыши	
auxclick	Щелчок дополнительной кнопки мыши (если есть)
click	Щелчок левой кнопкой
dblclick	Двойной щелчок
mousedown	Нажата кнопка мыши
mouseenter	Курсор мыши зашел в пределы элемента (получает верхний элемент)
mouseleave	Курсор мыши вышел за пределы элемента (получает верхний элемент)
mousemove	Курсор мыши переместился
mouseout	Курсор мыши вышел за пределы элемента (получают все нижние элементы)

Событие	Условие возникновения
mouseover	Курсор мыши зашел в пределы элемента (получают все нижние элементы)
mouseup	Отпущена кнопка мыши
wheel	Повернуто колесо мыши
События клавиатуры	
keydown	Нажата кнопка клавиатуры
keyup	Отпущена кнопка клавиатуры
keypress	Нажата и затем отпущена символьная кнопка
События устройств ввода текста	
compositionstart	Начало формирования текста
compositionupdate	Обновлено содержание текста
compositionend	Завершено формирование текста
События ввода данных	
beforeinput	Посылается перед обновлением содержания элемента
blur	Элемент потерял фокус ввода
focus	Элемент получил фокус ввода
focusin	Посылается перед получением фокуса
focusout	Посылается перед потерей фокуса
input	Посылается после обновления содержания элемента

Рассмотрим более подробно некоторые тонкости, связанные с событиями данной группы. Среди событий мыши есть две пары очень похожих событий: «mouseenter» и «mouseover» возникают, когда курсор мыши входит в область, принадлежащую данному элементу, «mouseleave» и «mouseout» — когда курсор выходит из этой области.

Различие между этими событиями заключается в том, что события «mouseenter» и «mouseleave» получает только самый «верхний» элемент, тогда как другая пара собы-

тий «mouseover» и «mouseout» передается всем элементам, находящимся под курсором мыши. Данный эффект называется всплытием событий (*англ.* bubbling или propagation). При использовании стандарта из приведенной выше ссылки обращайте внимание на атрибут «Bubbles»: если для него указано «No», то значит данное событие не всплывает — не передается следующему элементу. Если значение атрибута «Yes», то такое событие предусматривает всплытие.

Процесс всплытия можно продемонстрировать следующим рисунком. Предположим, что на нашей веб-странице есть два блока, вложенных один в другой. Внутренний блок находится «выше», то есть он первый принимает сообщения от мыши. Стрелками приблизительно указаны точки возникновения событий (на самом деле события возникают на границе блока). Стрелка, иллюстрирующая невсплывающие события «mouseenter» и «mouseleave», заканчивается на верхнем блоке. Другая стрелка, отвечающая за всплывающую пару «mouseover» и «mouseout», продолжается после верхнего блока и транслирует события нижнему блоку (рис. 9).

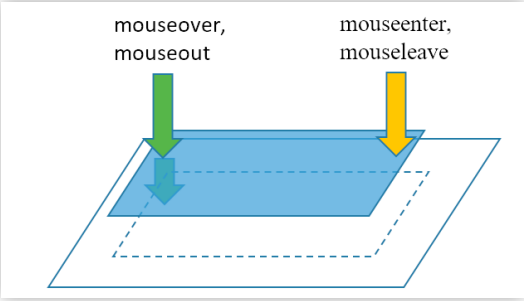


Рисунок 9

Для практического анализа описанных процессов сделаем страницу с вложенными блоками и обработчиками для событий разного типа. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_9.html*)

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Bubbling basics</title>
    <style>
        #d1 {
            border:1px solid navy;
            height: 200px;
            width:  200px;
        }
        #d2 {
            background: skyblue;
            height: 100px;
            margin: 50px;
            width:  100px;
        }
    </style>
</head>
<body>
    <h1>Mouse enter / Mouse over</h1>
    <div id="d1">
         <div id="d2"></div>
    </div>
    <p>Mouse enter <span id="s1"></span></div>
    <p>Mouse over <span id="s2"></span></div>
    <script>
        d1.onmouseenter = function(){
        s1.innerText = "d1"
        console.log("d1.onmouseenter")
    }
```

```
    d1.onmouseover = function(){
        s2.innerText = "d1"
        console.log("d1.onmouseover")
    }
    d2.onmouseenter = function(){
        s1.innerText = "d2"
        console.log("d2.onmouseenter")
    }
    d2.onmouseover = function(){
        s2.innerText = "d2"
        console.log("d2.onmouseover")
    }
    </script>
</body>
</html>
```

Основу страницы составляют два блока с идентификаторами «d1» и «d2». Блок «d2» вложен в блок «d1». При помощи стилей внешнему блоку устанавливается рамка, внутреннему — фоновый цвет. Также задаются размеры блоков. После блоков следуют текстовые параграфы, предназначенные для вывода информации о событиях.

В скриптовой части для блоков устанавливаются обработчики событий «onmouseover» и «onmouseenter». В них данные о событиях записываются в соответствующие элементы текстовых абзацев, а также дублируются в консоль.

Сохраните файл и откройте его в браузере. Поместите курсор мыши справа от блоков и начните медленно двигать его в сторону блоков. Обратите внимание на изменения, происходящие после пересечения курсором границ внешнего и внутреннего блоков.

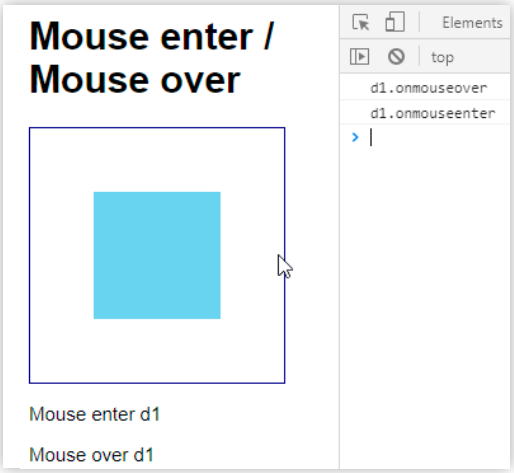


Рисунок 10

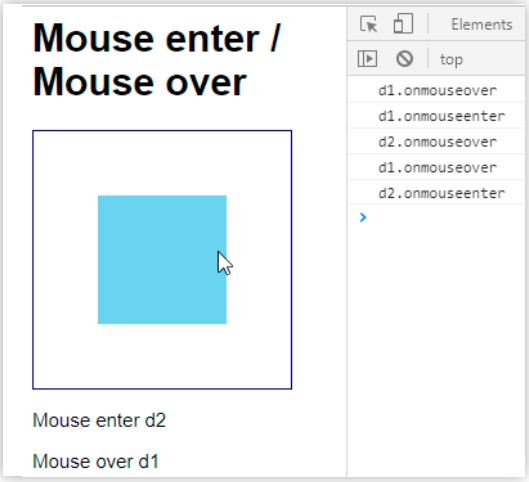


Рисунок 11

При пересечении границ внешнего блока оба события имеют одинаковый эффект. Обе надписи «Mouse enter» и «Mouse over» устанавливаются в значение «d1», в консоли также видны эти два сообщения.

Когда курсор мыши пересекает границу внутреннего блока, возникает три события, что следует из сообщений в консоли. Событие «onmouseover» приходит дважды — сначала от блока «d2», затем от «d1». В то же время событие «onmouseenter» приходит только один раз от верхнего блока «d2». Дублирование события «onmouseover» приводит к тому, что текст для сообщения «Mouse over» также устанавливается дважды. Последнее из них стирает предыдущее и в результате на экране остается результат «Mouse over d1», хотя курсор зашел на блок «d2». Об этих особенностях следует помнить при использовании событий разного типа.

О всплытии событий и управлении этим процессом детальнее будет рассказано в следующем уроке, после знакомства со структурой документа и компоновкой его элементов. Перейдем к рассмотрению событий клавиатуры.

Как правило, клавиатура используется для ввода определенных данных в специальные поля веб-страницы. Однако, для манипуляций с этим процессом предусмотрена отдельная группа событий, возникающих именно в процессе ввода данных. Такие события клавиатуры, как нажатие кнопки (keydown) или ее отпускание (keyup) обычно используют для задач, не связанных напрямую с вводом данных, например, для игровых или анимационных эффектов.

Выберем себе в качестве программы для реализации именно такую игровую задачу — «выстрел из пушки». Предположим, что у нас есть некий объект-пушка и, на некотором расстоянии от нее, мишень. При нажатии

клавиши «пробел» из пушки должен вылетать снаряд, пролетать до мишени и исчезнуть после попадания в нее.

Остановимся более подробно на задаче обеспечения полета снаряда. Классически, анимационные эффекты во многих языках программирования реализуются при помощи периодического выполнения некоторой функции или периодической посылкой некоторого сообщения (таймера) и являются частью модели событий, которые мы рассматриваем в данном разделе. В языке JavaScript, согласно спецификации ECMAScript, таких событий не предусмотрено.

Описанные возможности предоставляет нам окружение языка JavaScript — браузер, в виде функций «setTimeout» и «setInterval». Акцент на разделение обязанностей языка и браузера сделан не случайно, ведь, казалось бы, какая разница, на каком уровне реализованы функции, если их можно использовать в своих целях. Дело в том, что сам язык JavaScript не берет на себя ответственности за внутренний контроль времени из-за отсутствия таких механизмов. Браузер, в свою очередь, не имеет абсолютно полного влияния на процесс выполнения скриптов, то есть не может прервать одну инструкцию и перейти к другой по прошествии заданного временного интервала. В итоге, отсчет времени становится неточным и накапливает эти неточности шаг за шагом. Указанные механизмы нельзя использовать для задач, где требуется точный контроль времени.

Для нашей задачи большой точности не требуется, поэтому можно свободно применить одну из отмеченных браузерных функций. Рассмотрим их более детально:

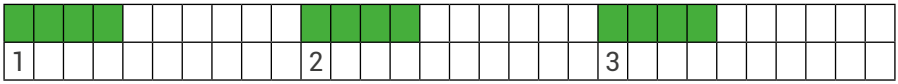
Функция	Описание
handle=setTimeout(func,dt)	Планирует запуск функции «func» через «dt» миллисекунд
clearTimeout(handle)	Отменяет запланированный запуск «handle»
handle=setInterval(func,dt)	Устанавливает периодический запуск «func» каждые «dt» миллисекунд
clearInterval(handle)	Отменяет периодический запуск «handle»

Функции делятся на те, которые планируют отложенный запуск функции, и те, которые эти планы отменяют. Планирующие функции «setTimeout» и «setInterval» возвращают идентификатор (дескриптор) своего «плана» — «handle». Возможно создать несколько различных планов отложенного запуска, отличающихся этим значением. При помощи дескриптора «handle» отменяющие функции определяют один конкретный план, который следует прекратить.

Функция «setTimeout» предполагает однократный отложенный запуск указанной функции, тогда как «setInterval» планирует бесконечную цепочку перезапусков, пока эта цепочка не будет прервана отменяющей функцией «clearInterval» или не закроется вкладка браузера с данной страницей. Обе функции требуют указать интервал времени до следующего запуска «dt» в миллисекундах. Напомним, что 1000 мс — это 1 секунда. То есть 500 мс — это 0,5 секунды, что соответствует 2 запускам за 1 секунду. 100 мс будет соответствовать 10 запускам за 1 секунду и т.д.

Хотя функция «setInterval» кажется более привлекательной для создания таймера — периодического запуска определенной функции, она имеет ряд недостатков.

Во-первых, она устанавливает время между началами запуска функции. Например, если функция сама-по-себе выполняется 0,4 секунды, и мы указали интервал запуска 1 секунду, то пауза между остановкой предыдущей и запуском новой функции будет 0,6 секунд



Во-вторых, функция «setInterval» накапливает ошибку. Если по каким-то причинам происходят задержки таймера, то в дальнейшем они не корректируются и могут привести к отклонениям ожидаемого и реального процессов. Часы, созданные при помощи функции «setInterval», будут отставать от действительного времени, причем, чем дольше будет работать программа, тем сильнее будет отставание.

В-третьих, браузер может сам приостановить или замедлить работу таймера, когда страница неактивна или компьютер переходит на питание от батарей. Также по-разному таймер может искажать свой ход при появлении диалоговых окон.

Как итог, функция «setInterval» крайне ненадежна. Вместо нее рекомендуется использовать периодический перезапуск функции однократного отложенного запуска «setTimeout». Можно добавить, что подобный прием является основным способом создания периодических процессов в некоторых языках, например, в Python.

Определившись с выбором функции управления перезапуском, перейдем к созданию интерфейса. Его вид приведен на следующем рисунке 12.

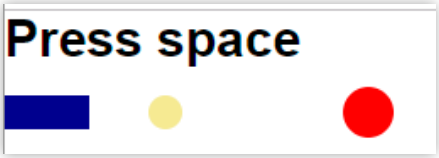


Рисунок 12

Основу страницы будут составлять три блока: прямоугольник, символизирующий ствол пушки, и два круга — для летящего снаряда и мишени.

Как мы уже исследовали в предыдущих примерах, управлять размещением элементов программным способом удобнее, применяя абсолютное позиционирование. Поэтому сразу укажем для всех элементов соответствующие стилевые атрибуты. Разметочная часть нашего документа примет следующий вид

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Keyboard events</title>
    <style>
        body {
            margin: 0;
            position: relative;
        }
        #gun, #bullet, #aim {
            position: absolute;
        }
        #bullet, #aim {
            border-radius: 50%;
        }
        #gun {
            background: navy;
```

```
            height: 20px;
            left: 0;
            top:  50px;
            width:  50px;
        }
        #bullet {
            background: khaki;
            height: 20px;
            left: 30px;
            top:  50px;
            width: 20px;
            z-index: -1;
        }
        #aim {
            background: red;
            height: 30px;
            left: 200px;
            top:  45px;
            width: 30px;
        }
    </style>
</head>

<body>
    <h1>Press space</h1>
    <div id="gun"></div>
    <div id="bullet"></div>
    <div id="aim"></div>
</body>
</html>
```

Блок, отвечающий за снаряд «bullet», изначально размещен за блоком-пушкой. Его координаты «left» и «top» подобраны таким образом, чтобы он располагался возле правого конца пушки. Атрибут «z-index: -1» перемещает блок «bullet» на задний план, из-за чего он будет скры-

ваться как за пушкой, так и, в последствие, за мишенью. Для того чтобы убедиться в правильности компоновки можете указать для блока «bullet» значение «z-index: 1», в таком случае он отобразится поверх других элементов и станет видимым для анализа размещения.

Перейдем к реализации скриптовой части. Во-первых, создадим переменные и функцию, отвечающие за движение снаряда.

```
<script>
    var bulletX = 30;
    var onFly = false;
    function moveBullet() {
        if(onFly){
            bulletX += 5;
            if(bulletX >= 200){
                bulletX = 30;
                onFly = false;
            }
            else {
                setTimeout(moveBullet, 50);
            }
            bullet.style.left = bulletX + "px";
        }
    }
</script>
```

Поскольку движение снаряда будет происходить только по горизонтали, нам нужна одна переменная, отвечающая за координату «x» снаряда — «bulletX». Ее начальное значение (30) соответствует стилевому атрибуту «left» блока «bullet». Также, для обеспечения перезапуска функции нам потребуется переменная «onFly», изначально установленная в «false».

Функция «moveBullet» отвечает за перемещение снаряда. Все тело функции окружено условным оператором «if(onFly)», то есть функция будет выполняться только в режиме полета снаряда. Координата «bulletX» увеличивается на несколько единиц. В нашем случае это «5», в дальнейшем можно поменять эту величину для более быстрого или медленного полета.

Далее выполняется проверка на то, что снаряд достиг мишени «if(bulletX >= 200)». Величина «200» также взята из стилевых определений атрибута «left», только у блока «aim». В том случае, если снаряд достиг мишени, его координата устанавливается в исходное состояние «bulletX = 30» (за пушкой) и режим полета прекращается изменением значения переменной «onFly».

Если же снаряд не достиг мишени, планируется отложенный запуск этой же функции «moveBullet» через 50 мс (то есть 20 раз в секунду). При перезапуске повторятся описанные действия.

В любом случае после изменения координаты «bulletX» ее значение переносится в стилевое определение блока-снаряда «bullet.style.left = bulletX + "px"». Не забываем, что для этого атрибута необходимо указывать единицы измерения.

Обратите внимание, что инструкция «setTimeout» не остановит работу текущего запуска функции, а лишь запланирует новый отложенный запуск. Поэтому блок «else» завершится и выполнится следующая за ним инструкция по изменению стиля, отвечающего за положение элемента.

Осталось добавить обработчик события нажатия клавиатуры, который будет запускать полет снаряда, то есть

функцию «moveBullet». Добавим в определение тела документа следующий атрибут

```
<body onkeypress="keyHandler(event)" >
```

А в скриптовую часть — код для функции-обработчика

```
function keyHandler(e){
    if(e.code == "Space" && !onFly) {
        onFly = true;
        moveBullet();
    }
}
```

Как уже было отмечено, функция проверяет тот факт, что нажатая клавиша соответствует пробелу. Для этого анализируется свойство «code» объекта-события. Согласно стандарта, это свойство имеет символьное описание нажатой кнопки, то есть для пробела имеет значение «Space». Дополнительно проверяется условие, что в данный момент снаряд не находится в состоянии полета.

При выполнении описанных условий устанавливается режим полета «onFly = true» и инициируется первый запуск функции «moveBullet». В дальнейшем, она сама себя будет перезапускать, пока снаряд не достигнет мишени.

Создайте новый HTML-файл. Наберите или скопируйте в него приведенное выше содержание (*полный код программы также доступен в папке Sources, файл JS_3_10.html*). Убедитесь, что внешний вид страницы соответствует приведенному выше рисунку, а при нажатии кнопки «пробел» происходит «выстрел» — пролет снаряда от блока-пушки до блока-мишени.

Расширим условие задачи: добавим в нашу программу возможность отмены выстрела при нажатии клавиши «Esc» (*Escape*).

Для начала, попробуем узнать детали события, происходящего при нажатии клавиши «Escape». Добавим в обработчик события «keyHandler» команду логгирования параметра «e» в консоль

```
function keyHandler(e){
console.log(e)
    if(e.code == "Space" && !onFly) {
        onFly = true;
        moveBullet();
    }
}
```

Сохраните изменения, обновите вкладку браузера, откройте консоль разработчика. После этого переведите фокус ввода из консоли на страницу — щелкните мышью в произвольном месте вкладки (иначе нажатие кнопок будет приниматься консолью, а не вкладкой).

Нажмите пробел, убедитесь, что происходит «выстрел», а в консоли появляется сообщение о нажатии клавиши «пробел» (рис. 13).

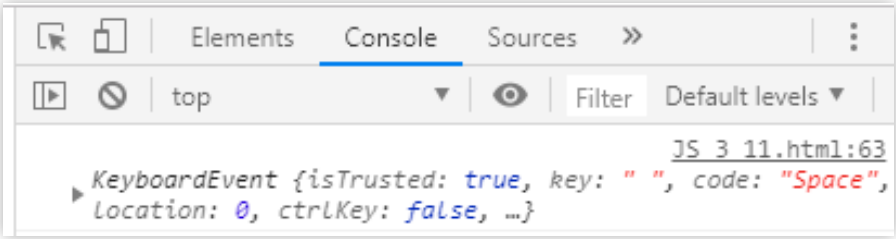


Рисунок 13

Нажмите клавишу «Escape». Обратите внимание, что сообщение в консоли не появляется. Это происходит вследствие особенности события «keypress». Оно не формируется для некоторых клавиш, которые имеют служебное назначение. Например, стрелки управления курсором или функциональные клавиши «F1», «F2» и т.д. Можете исследовать на базе созданной нами программы какие клавиши создают событие «keypress», а какие нет.

Для того чтобы иметь возможность обработать события от нажатия служебных клавиш необходимо использовать событие «keydown». Это системное событие, сопровождающее любые действия клавиатуры. Замените обработку события «keypress» на «keydown». Тег, формирующий тело документа, должен при этом принять следующий вид

```
<body onkeydown ="keyHandler(event)" >
```

Сохраните изменения и обновите вкладку браузера. Убедитесь, что фокус ввода находится на вкладке и нажмите клавишу «Escape». Убедитесь, что теперь в консоли появляется соответствующее сообщение (рис. 14).

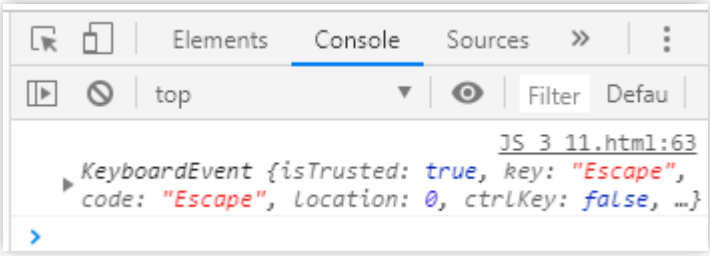


Рисунок 14

Также можете убедиться, что события приходят и от тех клавиш, от которых не приходили события «keypress». Обратите внимание, что среди параметров события также есть «code», который мы использовали в обработчике предыдущего события. Значит, тело обработчика должно функционировать и для нового события. Нажмите клавишу «пробел» и убедитесь в этом.

Главной информацией для нас является кодовое название клавиши «code: "Escape"». По этому значению мы можем задать действие для нажатия клавиши. Добавим в обработчик события следующий код

```
if(e.code == "Escape" && onFly) {
    onFly = false;
    bulletX = 30;
    bullet.style.left = bulletX + "px";
}
```

В условном операторе производится проверка на то, что событие пришло именно от клавиши «Escape», а также снаряд находится в состоянии полета «onFly». В таком случае переменной «onFly» присваивается значение «false». Это прекратит работу функции, обеспечивающей полет «moveBullet», поскольку в ней первым делом проверяется указанная переменная.

Затем, возвращаем снаряд в исходное состояние: указываем значение для координаты снаряда «bulletX = 30» и заносим это значение в стилевой атрибут «bullet.style.left», отвечающий за позицию блока-снаряда. В итоге, он должен «спрятаться» за блоком-пушкой, создав эффект исчезновения с экрана.

Сохраните изменения (*код с изменениями доступен в папке «Sources», файл «JS_3_12.html»*). Обновите вкладку браузера. Нажмите пробел, убедитесь, что происходит «выстрел». Во время полета снаряда нажмите клавишу «Escape» — снаряд должен прекратить полет и визуально исчезнуть. Проверьте, что можно совершить повторные выстрелы и сохраняется возможность их отмены.

В процессе работы над программой мы столкнулись с двумя событиями, отвечающими за нажатие кнопок клавиатуры «keypress» и «keydown». Рассмотрим более детально различия между ними. Событие «keydown» посылается каждой кнопкой клавиатуры, что дает более широкие возможности их обработки в скриптах. Однако, в качестве символа нажатой кнопки «key» обычно посылается только базовый идентификатор кнопки — латинский символ в верхнем регистре. То есть нажатие кнопки «s» или «shift-s» приведет к одному и тому же значению «key» — «S». Более того, это значение будет сохраняться, если пользователь поменяет раскладку клавиатуры или переключится на другой язык ввода.

Событие «keypress» учитывает состояние регистра (нажатая кнопка «Shift» или включенный режим «Caps Lock»), раскладку клавиатуры и язык ввода. То есть значение «key» отвечает не за кнопку, которая была нажата, а за символ, который ей соответствует. Это является более удобным при анализе текстового ввода. Также, событие «keypress» не посылается служебными кнопками, что также не нужно для работы с текстовой информацией.

Как итог — событие «keypress» более удобно для работы с текстовым вводом, тогда как событие «keydown» —

для других задач, в которых может быть задействована клавиатура.

В качестве замечания можно отметить тот факт, что с развитием браузеров и операционных систем значение для поля «key» в событии «keydown» также может учитывать раскладку и язык ввода. Для создания универсальных приложений лучше пользоваться стандартизированным полем «code», хранящим обозначение клавиши, одинаковое для всех браузеров.

События жизненного цикла

Когда Вы включаете браузер и открываете в нем новую веб-страницу происходит определенная цепочка процессов: браузер находит сервер, на котором находится страница, и запрашивает ее HTML код. Получив ответ, браузер, обрабатывает его и формирует структуру страницы. Обычно, в HTML коде находятся ссылки на дополнительные ресурсы — стилевые файлы, файлы скриптов, рисунков, фреймы и т.п. Обработав HTML код и собрав данные о всех ссылках, браузер формирует запросы на эти ресурсы, получая каждый из них отдельно, и подключает их к ранее загруженной странице. Когда все ресурсы будут получены, веб-страница будет считаться загруженной окончательно.

После загрузки страницы начинается процесс взаимодействия с пользователем. При этом на странице могут появляться, исчезать или изменяться ее составные элементы. Некоторые из них могут потребовать новых обращений к серверу, например, добавление новых изображений.

Закончив работу с веб-страницей пользователь закрывает вкладку, и браузер приступает к освобождению памяти, занятой этой страницей и ее ресурсами. Также удаляются или перемещаются в специальное хранилище (кеш) загруженные с сервера файлы стилей, скриптов и изображений.

Описанный процесс носит название жизненного цикла веб-страницы.

Как сама страница, так и отдельные ее элементы, требующие отдельной загрузки, получают события, связанные с началом или окончанием того или иного этапа жизненного цикла. События данной группы позволяют встраивать определенную функциональность в нужном месте цикла. Например, нецелесообразно вызывать функции, описанные в отдельном файле, до тех пор, пока файл еще полностью не загружен.

Отметим также тот факт, что в процессе загрузки любого из дополнительных ресурсов возможно возникновение ошибки. Есть вероятность, что ресурс был перемещен или удален, обновлен, из-за чего поменял имя, или просто возникли сбои с сетевым подключением. В этих случаях события жизненного цикла могут предоставить нам сведения о проблемах с подключением ресурсов. Если страница может работать и без дополнительных функций, то можно принять соответствующее решение. Например, если за счет неудачной загрузки скриптов не будет работать «карусель», меняющая один за другим слайды, то это можно считать некритической ошибкой и продолжать работу страницы. Если же ресурсы обеспечивают основную функциональность и без них работа невозможна, то события жизненного цикла позволят нам отреагировать на такие ситуации.

Жизненный цикл веб-страницы сопровождается следующими событиями

- DOMContentLoaded — браузер полностью загрузил HTML, файлы стилей и скриптов, построил структуру документа (DOM-структуру, детальнее о DOM будет рассказано в следующем уроке);

- load — браузер загрузил все дополнительные ресурсы — изображения, фреймы;
- beforeunload — браузер получил команду закрыть страницу (вкладку);
- unload — браузер закрыл страницу (вкладку).

Событие «DOMContentLoaded» является одним из наиболее популярных среди событий жизненного цикла. Оно посылается объекту «document» тогда, когда загружен код HTML, стилевые файлы и скрипты. На момент посылки этого сообщения могут быть еще не загружены изображения или фреймы. Тем не менее, структура страницы уже определена, то есть в обработчике события все элементы страницы доступны и к ним можно обращаться по их атрибутам («id», «class», «name» и т.п.).

Обработчик события «DOMContentLoaded» подключается *только* при помощи команды «addEventListener»:

```
document.addEventListener( "DOMContentLoaded" ,
    function() {
        alert("DOM loaded")
    }
)
```

Использование атрибутов с префиксом «on», как для большинства других событий, не даст должного эффекта. Ошибки не возникнет, т.к. мы имеем право создать любое дополнительное поле в произвольном объекте, однако, такой обработчик не будет вызван при загрузке документа

```
document.onDOMContentLoaded = function() {
    alert("DOM loaded")
}
```

Событие «load» посылается после загрузки всех дополнительных ресурсов — изображений, фреймов и т.п. Данное событие относится к объекту «window» и может быть обработано всеми допустимыми вариантами — как при помощи метода «addEventListener», так и указанием атрибутов с префиксом «on»

```
<body onload="alert('body loaded')">
window.onload = function() {alert('body loaded')}
window.addEventListener( "load",
       function() {alert('body loaded')})
```

Следует обратить внимание на то, что загрузка изображений еще не означает их отображение. Событие «load» посылается по факту окончания загрузки, но не по окончанию прорисовки. В большинстве случаев изображения хранятся в сжатых форматах, и для их вывода на страницу браузеру необходимо повести «декомпрессию» — восстановление графической информации из сжатого файла. На это уходит определенное время, которого может не хватить до вызова обработчика события.

Рассмотрим процессы прихода событий и состояния страницы в эти моменты на следующем примере. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_13.html*). Скопируйте также из папки «*Sources*» в свою рабочую папку файл-изображение «*step.jpg*»

```
<!doctype html />
<html>
<head>
    <style>
        img {
            height: auto;
            width:  200px;
        }
    </style>
</head>

<body onload="alert('body loaded')">
    <img src="step.jpg" />
    <script>
       document.addEventListener("DOMContentLoaded" ,
           function() {
               alert("DOM loaded")
           }
       )
    </script>
</body>

</html>
```

На данной странице размещается единственный элемент — изображение (рис. 17).

Также в коде установлены два обработчика событий жизненного цикла: «DOMContentLoaded» и «load». Первый — при помощи метода «addEventListener», второй — через атрибут «onload» тела документа.

Сохраните файл, откройте его в браузере. В процессе загрузки появится два сообщения, после нажатия на кнопку «OK» диалогового окна страница будет принимать следующий вид (рис. 15, 16)

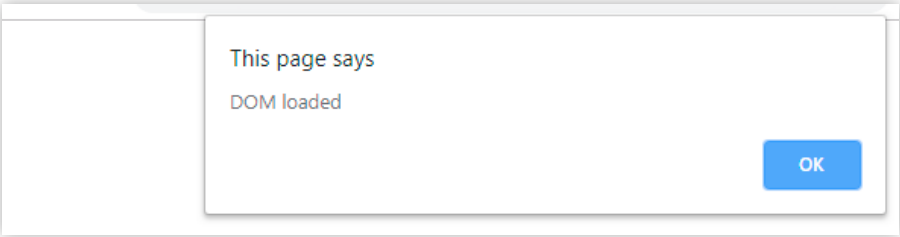


Рисунок 15

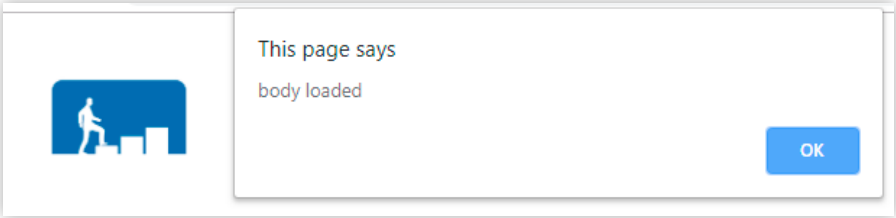


Рисунок 16

Рисунок 17

Событие «DOMContentLoaded» приходит сразу после загрузки HTML кода страницы и построения ее составных элементов. Само содержимое элементов может еще быть не загруженным. Поэтому, в нашем случае, в момент прихода события и вызова сообщения на вкладке браузера ничего не отображается (наш единственный элемент требует загрузки дополнительного ресурса — файла

«*step.jpg*»). Появление диалогового окна (alert) останавливает выполнение скриптов и загрузку страницы, благодаря чему мы успеваем увидеть последовательность процессов загрузки и посылки сообщений.

Нажмите кнопку «OK». Появится следующее сообщение «body loaded». Это соответствует моменту окончания загрузки всех ресурсов, в данном случае — изображения. Однако на вкладке мы видим лишь его часть — тот фрагмент, который браузер успел восстановить из полученного файла и отобразить на странице. Вызов диалогового окна снова остановил работу и зафиксировал момент частичной прорисовки изображения.

Нажмите «OK» во втором диалоговом окне, и изображение должно появиться полностью. Следует отметить, что в зависимости от быстродействия компьютера, планшета, смартфона или другого устройства, на котором открывается страница, промежуточный фрагмент изображения может быть разного размера. Возможно, при малом быстродействии, фрагмент не появится вообще, т.к. браузер не успеет обработать достаточно данных даже для частичного отображения. При высоком быстродействии или при малом размере изображения оно может успеть обработаться полностью. В таком случае на момент события «load» на странице будет видно все содержимое картинки.

Для того чтобы исследовать процессы загрузки страницы и ее ресурсов в инструментах разработчика браузера есть специальная вкладка «Network». Откройте консоль уже известным Вам способом и переключитесь на указанную вкладку сетевого монитора. Для сбора информации

необходимо перезагрузить страницу с открытой инструментальной вкладкой «Network», поэтому обновите страницу, нажав клавишу «F5» клавиатуры либо кнопку обновления в браузере. Инструмент начнет сбор данных.

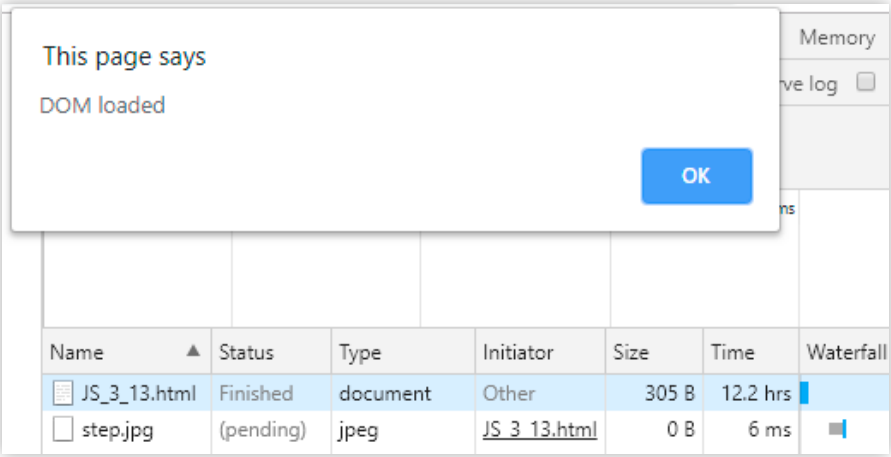


Рисунок 18

В момент прихода первого сообщения мы увидим следующие параметры: основной документ «*JS_3_13.html*» уже загружен, о чем свидетельствует статус «Finished», в то же время ресурс «*step.jpg*» находится в состоянии приема «(pending)». В колонке «Waterfall» отображается диаграмма последовательности процессов. Как видно, ресурс начал загружаться, но загрузка еще не закончена, о чем свидетельствует синий бегунок в конце диаграммы. Вполне возможно, что при других параметрах быстродействия, загрузка может еще не начаться либо, наоборот, будет загружена большая часть ресурса.

Нажмите кнопку «OK». Состояние сетевого монитора сменится на следующее (рис. 19).

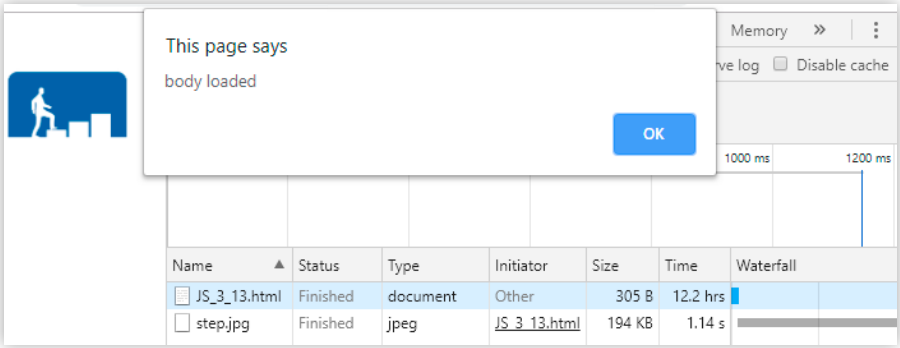


Рисунок 19

Как видно из статуса процессов, загрузка всех ресурсов завершена. Однако, диаграмма «Waterfall» еще не определила момент конца загрузки, т.к. не пришел отклик от браузера об обработки полученных данных. Из-за этого диаграмма загрузки «*step.jpg*» не имеет конца. Рисунок на странице, как мы уже видели ранее, отображается частично.

Закройте диалоговое окно, нажав «OK». Сетевой монитор получит сведения от браузера об окончании обработки ресурса и сможет определить время, потраченное на загрузку и обработку изображения.

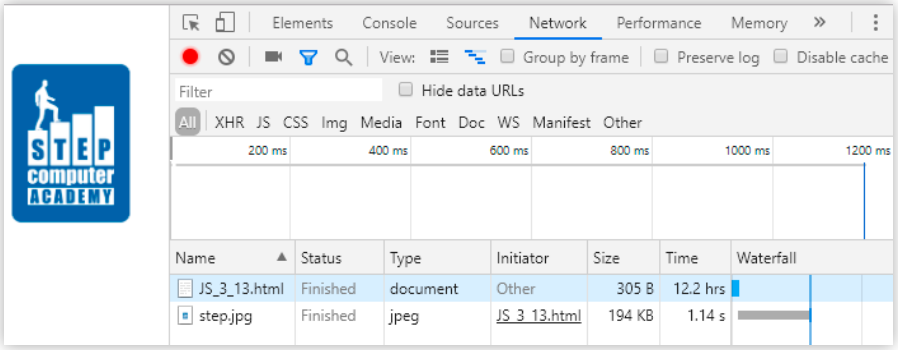


Рисунок 20

Диаграмма загрузки ресурса отобразится как законченный блок, линия в конце него означает, что завершены все загрузки, необходимые для страницы. Как следствие, рисунок на странице отображается полностью.

Поскольку дополнительные ресурсы страницы загружаются отдельно от HTML кода, для них также посылаются события об итоговом статусе. Такими событиями являются «load» и «error». Событие «load» посылается элементу, когда его загрузка успешно завершается. В случае неудачной загрузки элемент получает событие «error». Эти события дают возможность убедиться в том, что ресурс полностью загрузился (или возникла ошибка загрузки).

Нам уже известно, что изображения загружаются в самостоятельных потоках. Значит, для них будут посылаться события «load» или «error». Рассмотрим их обработку на следующем примере, определяющем успешность загрузки ресурсов-изображений. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_14.html*). Для правильной работы примера ему нужен файл «*step.jpg*». Убедитесь, что этот файл доступен из предыдущего примера или скопируйте его из папки «Sources» в свою рабочую папку.

```
<!doctype html />
<html>
<head>
    <style>
        img {
            height: auto;
            width:  100px;
        }
```

```
    </style>
</head>

<body>
    <img src="step.jpg"
       onload="addMessage('step.jpg load OK')"
       onerror="addMessage('step.jpg load error')"/>
    <img src="step2.jpg"
       onload="addMessage('step2.jpg load OK')"
       onerror="addMessage('step2.jpg load error')"/>
    <p id="txt"></p>

    <script>
        function addMessage(msg) {
            window.txt.innerHTML += msg + "<br/>"
        }
    </script>
</body>
</html>
```

В теле документа расположены два изображения «<img src="step.jpg"» и «<img src="step2.jpg"». Для одного из них файл-ресурс существует, для другого — нет. Поэтому первое изображение должно загрузиться с успешным статусом, тогда как второе — с ошибкой.

В каждом из изображений установлены по два обработчика событий «load» и «error». Обработчики вызывают дополнительную функцию «addMessage», которая описана в скриптовой части и «выводит» сообщение, добавляя его к внутреннему содержимому абзаца «<p id="txt">» через его атрибут «*txt.innerHTML*».

Сохраните файл и откройте его при помощи браузера. Изображение, для которого файл-ресурс присутствует

в рабочей папке, должно отобразиться нормально. Если с ним возникает ошибка, то убедитесь, что файл скопирован в ту же папку, в которой находится HTML файл с текущим примером.

Второе изображение, которое ссылается на несуществующий файл, загружается с ошибкой и, соответствующим образом отображается на странице. Итоговый вид страницы представлен на следующем рисунке:

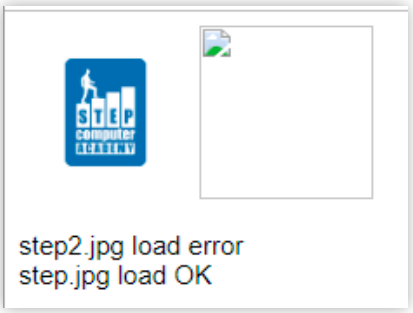


Рисунок 21

Обратите внимание, что сообщение об ошибке загрузки второго изображения появляется раньше, чем об успешной загрузке первого. Это связано с тем, что загрузка первого изображение занимает какое-то время, тогда как второе изображение сразу вызывает ошибку. Поэтому эта ошибка приходит раньше, чем сообщение об успехе, и надпись появляется первой, хотя изображение обрабатывается позже.

Задание для самостоятельной работы: добавьте на страницу еще несколько изображений с контролем загрузки. В конце блока сообщений добавьте итоги, например: «успешно загружено — 3», «ошибки загрузки — 1».

События «beforeunload» и «unload» посылаются странице на последних этапах ее жизненного цикла — при закрытии вкладки браузера, содержащую данную страницу, либо при переходе по ссылке на другой сайт (в этой же вкладке). Сам процесс закрытия страницы состоит из нескольких этапов, проходящих в обратной последовательности по сравнению с загрузкой. Сначала происходит освобождение памяти, занятой ресурсами страницы. В нашем примере — это изображение, восстановленное из сжатого «JPG» файла. Затем удаляются из памяти структурные элементы страницы. На последнем этапе закрывается окно вкладки или весь браузер, либо начинается процесс загрузки новой страницы — в зависимости от действий пользователя.

Событие «beforeunload» посылается окну (объекту «window») и сигнализирует о начале закрытия (выгрузки, unload) страницы. На данном этапе можно предупредить пользователя, что на странице остались несохраненные данные, незаконченные процессы отправки или получения данных. При этом пользователь может отменить закрытие страницы и вернуться к работе с ней. Использование данного события очень популярно для страниц, на которых используется пользовательский ввод — текстовые редакторы, онлайн компиляторы различных языков программирования и т.п.

Событие «unload» приходит в самом конце жизненного цикла. Отменить закрытие страницы на данном этапе уже невозможно, поэтому данное событие используется крайне редко. В обработчике данного события обычно

закрывают связанные с данной страницей дополнительные окна (если они создавались).

Обработка события «beforeunload» имеет несколько особенностей. Рассмотрим их на практическом примере. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_15.html*).

```
<!doctype html />
<html>
<head>
    <style>
        #text {
            height: 200px;
            width:  400px;
        }
    </style>
</head>

<body>
    <textarea id="text"></textarea><br/>
    <a href="http://itstep.org">IT Step</a>
    <script>
        window.onbeforeunload =
            function(e) {
                if(text.value.length > 0) {
                var msg = 'Text not saved';
                e.returnValue = msg;
                return msg;
            }
            return null;
        }
    </script>
</body>
</html>
```

В теле документа расположен текстовый блок (<textarea id="text">), для которого в заголовочной части определены стили, устанавливающие размеры блока. Ниже блока расположена ссылка на внешний ресурс (<a href="http://itstep.org">), нажатие на которую приведет к выгрузке текущей страницы и загрузке новой.

В скриптовой части документа устанавливается обработчик события «window.onbeforeunload». Принцип работы обработчика заключается в том, что он должен вернуть непустое сообщение, если необходимо уведомить пользователя о несохраненных данных. В приведенном примере в качестве сообщения использовано «msg = 'Text not saved'». Для совместимости с разными типами браузеров необходимо:

а) установить свойство «e.returnValue=msg» для объекта-события «e»;

б) вернуть сообщение в качестве результата работы функции-обработчика «return msg».

В случае, если сообщение пользователю выводить не нужно, обработчик возвращает значение «null». Критерием проверки для выдачи сообщения является наличие пользовательского ввода в текстовом блоке. Это проверяется путем определения длины текста, содержащегося в текстовом блоке: «if(text.value.length > 0)».

Обратите внимание, что мы не создаем никаких диалоговых окон на подобие «alert» или «confirm». Мы лишь обеспечиваем возврат ненулевого значения (сообщения) из обработчика события.

Сохраните файл и откройте его в браузере. Не вводя никакого текста, нажмите на ссылку под текстовым блоком. Должен произойти переход на другую страницу.

Вернитесь на предыдущую страницу (нажмите комбинацию Alt-«стрелка влево» либо кнопку «назад» в браузере). Введите в текстовый блок произвольные данные и снова нажмите на ссылку. В результате должно появиться сообщение. Вид этого сообщения сильно зависит от используемого браузера:

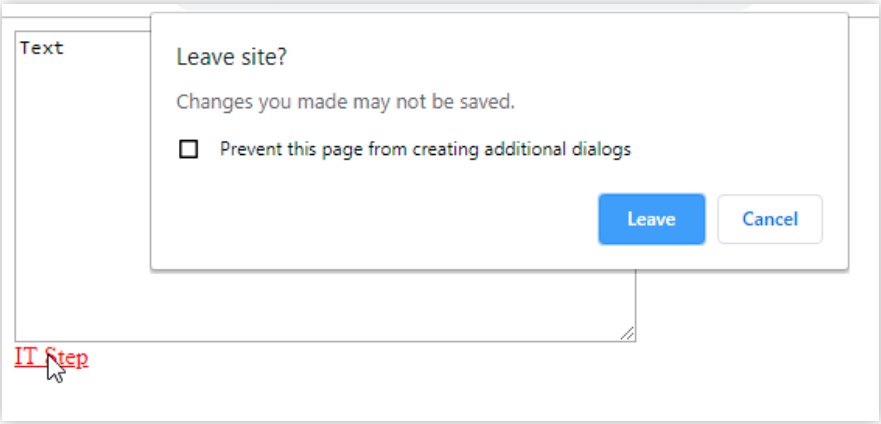


Рисунок 22. Google Chrome

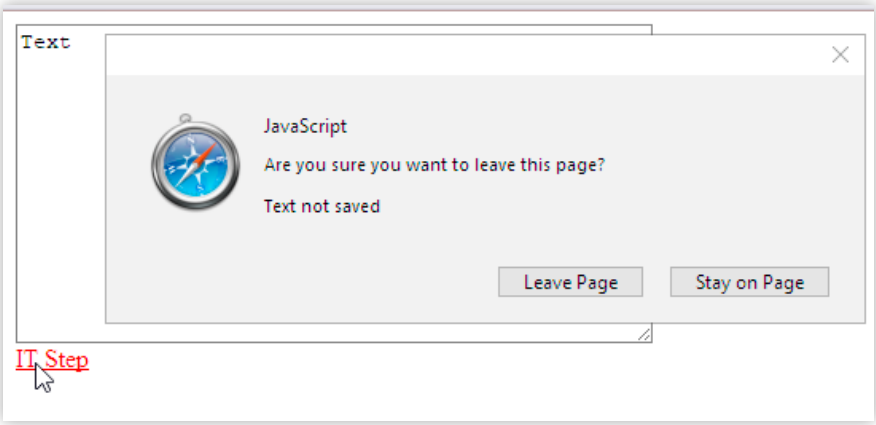


Рисунок 23. Apple Safari

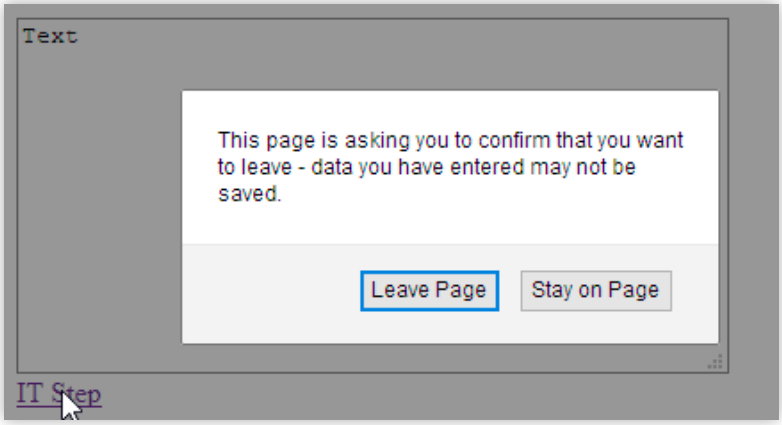


Рисунок 24. Mozilla Firefox

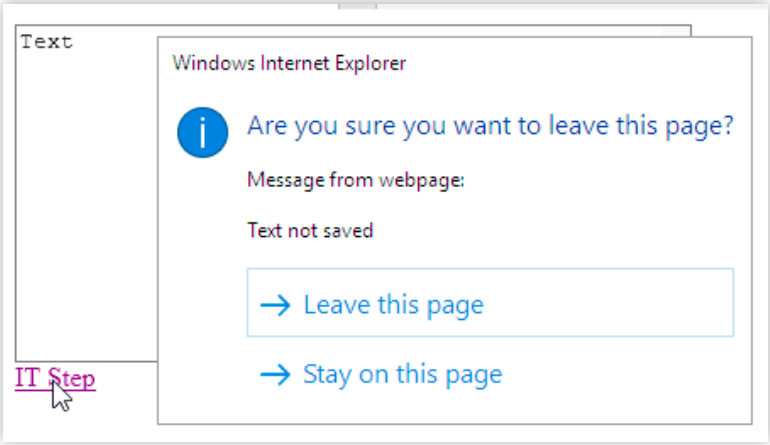


Рисунок 25. Internet Explorer

Кроме того, что сообщения отличаются внешним видом, характерным для каждого вида браузеров, обращает на себя внимание тот факт, что текст нашего сообщения ('Text not saved') отображается не во всех случаях. Браузеры «Google Chrome» и «Mozilla Firefox» полностью

игнорируют содержание данного сообщения. Браузеры «Apple Safari» и «Internet Explorer» добавляют его к своим собственным сообщениям. Если у Вас установлен другой браузер, то и сообщение может отличаться от приведенных выше.

Почему браузеры игнорируют сообщения, заложенные в скриптовой части? Это делается с целью безопасности. Исторически, браузеры предоставляли программисту полный контроль над процессом выгрузки (закрытия) страницы. Недобросовестные разработчики сайтов с целью удержания посетителей на своих страницах использовали разнообразные негативные приемы. Например, «переставленные кнопки» — при нажатии на кнопку «покинуть страницу» на самом деле генерировалась команда «остаться на странице». Также использовались сообщения пугающего содержания, на подобие «При нажатии на кнопку с Вашего счета будет списано $10».

Для упреждения подобных уловок, было принято решение о полном или частичном игнорировании данных, передаваемых в сообщение. Браузеры в любом случае не дают возможность управлять внешним видом диалогового окна — менять его текст или порядок кнопок. А дополнительный текст сообщения, если и показывается, то так, чтобы было полностью понятно, что это лишь дополнительный текст, не искажающий смысл данного диалогового окна. Как мы уже убедились, в ряде случаев этот текст вообще будет проигнорирован.

Также с целью безопасности в обработчике события «onbeforeunload» не допускается создание дополнительных диалоговых окон. Если к коду обработчика добавить

строку «alert(3)», то в консоли разработчика при приходе события появится предупреждение:

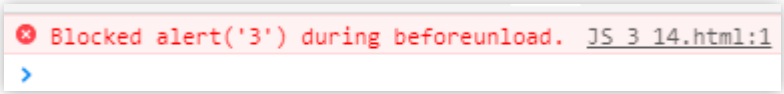


Рисунок 26

Можете убедиться в этом на практике, добавив указанную инструкцию в тело обработчика.

Задание для самостоятельной работы: добавьте на страницу еще два поля ввода текста. Дополните обработчик события «onbeforeunload» проверкой на наличие текста хотя бы в одном из полей ввода — если такой текст есть, то выдается предупреждение о несохраненных данных.

Обработчики событий по умолчанию (стандартные обработчики), запрет вызова стандартного обработчика

Для некоторых событий в браузере предусмотрены специальные стандартные обработчики, которые запускаются автоматически — без необходимости добавления их программным путем (обработчики событий по умолчанию). Обычно, это широко распространенные для многих приложений и операционных систем действия: вызов контекстного меню при нажатии правой клавиши мыши, масштабирование контента при прокрутке колеса мыши с нажатой кнопкой «Ctrl» на клавиатуре, совершение определенных действий при нажатии «горячих» комбинаций, например, сохранение страницы при нажатии «Ctrl-S», копирование и вставка выделенного текста «Ctrl-C» — «Ctrl-V», и многие другие.

В некоторых случаях обработчики событий по умолчанию могут мешать логике работы основного содержимого страницы. Такие ситуации возникают, если в интерфейсе страницы используются средства управления, совпадающие с «зарезервированными» возможностями. Например, когда необходимо задействовать для своих целей правую клавишу мыши, ее колесо или сочетания некоторых удобных для нажатия кнопок клавиатуры, которые традиционно вызывают действия по умолчанию.

Рассмотрим в качестве примера следующую задачу: создать блок, который будет менять цвет при нажатии на нем клавиши мыши: зеленый, если нажата левая клавиша; синий, если средняя (или колесо), красный — если правая.

При нажатии клавиши мыши объект получает событие «mousedown». Это событие формируется основными клавишами мыши — левой, средней (или нажатие на колесо) и правой. Если мышь оснащена дополнительными клавишами (кроме перечисленных), то их нажатие сопровождается событием «auxclick» (см таблицу описания событий в разделе 3). Будем обрабатывать в нашем примере только события от основных клавиш мыши.

Поскольку все три клавиши мыши посылают одно и то же событие «mousedown», они устанавливают различные значения для свойства «which» объекта-события, передаваемого в функцию-обработчик: 1 — для левой клавиши, 2 — для средней, 3 — для правой. Используем эти значения для определения того, от какой клавиши поступило событие, и применим к блоку необходимый цвет. Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_16.html*).

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Defaults</title>
    <style>
        #d1 {
            border: 2px solid gray;
            height: 200px;
```

```
            width:  200px;
        }
    </style>
</head>

<body>
    <div id="d1"></div>
    <script>
        window.d1.onmousedown = function(e) {
            var bgColor;
            switch( e.which ) {
                case 1:
                    bgColor = "lime" ;
                    break ;
                case 2:
                    bgColor = "aqua" ;
                    break ;
                case 3:
                    bgColor = "tomato" ;
                    break ;
            }
            window.d1.style.backgroundColor = bgColor ;
        }
    </script>
</body>
</html>
```

В теле документа создается блок , для которого в заголовочной части при помощи стилей задаются размеры 200×200 пикселей и рамка, обеспечивающая видимость его границ. В скриптовой части для блока устанавливается обработчик события «onmousedown». Основным действием функции-обработчика является установка фонового цвета блока «window.d1.style.backgroundColor = bgColor». Сам цвет определяется в теле опе-

ратора «switch» в зависимости от значения поля «e.which», отвечающего за номер нажатой клавиши мыши.

Сохраните файл и откройте его при помощи браузера. Нажмите на блок левой клавишей, средней (или нажмите на колесо), правой — убедитесь, что блок меняет свой цвет. Обратите внимание, что при нажатии правой клавиши кроме смены цвета на блоке появляется контекстное меню (рис. 27).

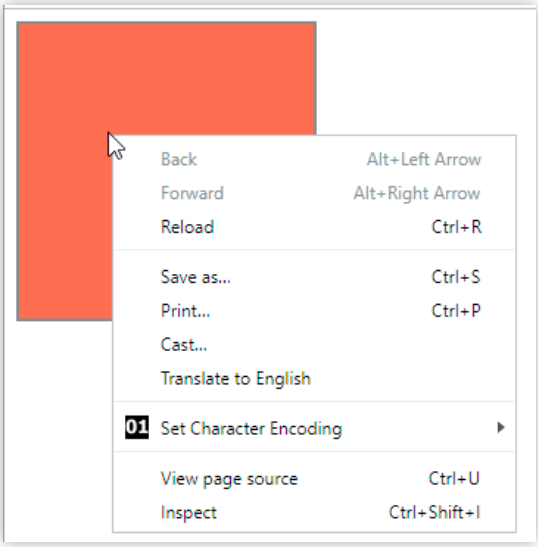


Рисунок 27

Вызов контекстного меню определяется браузером как действие по умолчанию, предназначенное для правой клавиши мыши. Мы в коде нигде не указывали данную функциональность. В нашем примере такое поведение является нежелательным, поэтому необходимо заменить стандартный обработчик события контекстного меню «oncontextmenu» для блока.

Добавьте в скриптовую часть документа следующее определение (*код с изменениями также доступен в папке Sources, файл JS_3_16a.html*)

```
window.d1.oncontextmenu = function() { return false }
```

Сохраните изменения и обновите страницу, открытую в браузере. Убедитесь, что теперь при щелчке на блоке правой клавишей мыши контекстное меню не появляется. Тем не менее, при щелчке за пределами блока меню все так же вызывается.

Задание для самостоятельной работы. Модифицируйте код таким образом, чтобы контекстное меню не вызывалось и за пределами блока.

Стандартные обработчики, вызываемые определенными комбинациями кнопок клавиатуры, переопределяются несколько иным способом. Рассмотрим это на следующем примере: создадим программу, выводящую на экран информацию о нажатой кнопке или их комбинации.

Создайте новый HTML-файл, наберите или скопируйте в него следующее содержимое (*код также доступен в папке Sources, файл JS_3_17.html*).

```
<!doctype html />
<html>
<head>
    <meta charset="utf-8" />
    <title>Defaults</title>
</head>

<body>
    <h1>Click on page and press a key</h1>
```

```
    <h2 id="out"></h2>
    <script>
        document.body.onkeydown = function(e) {
            var txt = "" ;
            if(e.ctrlKey)  txt += "Ctrl – " ;
            if(e.altKey)   txt += "Alt – " ;
            if(e.shiftKey) txt += "Shift – " ;
            txt += e.key ;
            out.innerText = txt ;
        }
    </script>
</body>
</html>
```

На странице расположен заголовок — подсказка «Click on page and press a key», напоминающая о назначении программы. Далее следует блок <h2 id="out">, который будет использоваться для вывода сообщений о нажатых кнопках.

В скриптовой части документа устанавливается обработчик события «onkeydown». Мы с ним знакомились ранее в этом уроке, поэтому детально на его описании останавливаться не будем. В обработчике проверяются статусы нажатия кнопок «Ctrl», «Alt», «Shift» и, если они истинны, к сообщению добавляются соответствующие надписи. Далее сообщение дополняется символом кнопки «e.key» и выводится в блок «out».

Сохраните файл и откройте его в браузере. Щелкните мышью по станице для того, чтобы быть уверенным в том, что фокус ввода клавиатуры принадлежит данной вкладке. Нажимайте кнопки клавиатуры, в том числе удерживая кнопки «Ctrl», «Alt», «Shift» в различных

комбинациях. Убедитесь в соответствии нажатых кнопок и сообщений на странице.

Нажмите комбинацию, для которой установлено действие по умолчанию. Например, «Ctrl-e». Как результат, кроме сообщения о нажатых кнопках браузер запустит средства поиска:

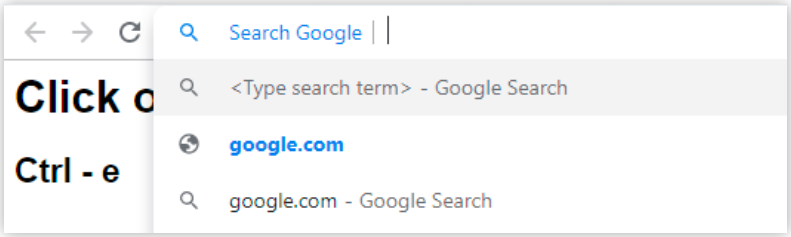


Рисунок 28

Попробуйте другие «горячие» комбинации: «Ctrl-s» будет открывать диалог сохранения страницы, «Ctrl-f» вызывать средства текстового поиска по вкладке, «Ctrl-u» — откроет окно с исходным кодом страницы, «Ctrl-r» — перезагрузит вкладку.

События от клавиатуры не предусматривают отдельных обработчиков действий по умолчанию, как это было с контекстным меню. Все они также проходят через событие «keydown». Однако, добавлением инструкции «return false» в обработчик события предотвратить действие по умолчанию не удается. Можете убедиться в этом, дополнив функцию-обработчик указанной инструкцией.

Для того чтобы браузер не выполнял действия по умолчанию необходимо вызвать метод «preventDefault» у объекта-события «event» или, в нашем случае, у параметра функции-обработчика «e». Добавьте в любом месте

описания функции инструкцию (*код с изменениями также доступен в папке Sources, файл JS_3_17a.html*)

```
e.preventDefault() ;
```

Сохраните файл и обновите страницу браузера. Убедитесь в том, что на комбинацию «Ctrl-e» браузер перестал реагировать и средства поиска не запускаются. Также теряют активность другие комбинации, описанные выше.

Следует отметить, что инструкция «preventDefault» действует только на те «горячие» комбинации, которые касаются вкладки браузера. Данная инструкция не отменяет действия, предусмотренные для браузера в целом, фоновых программ или для операционной системы. Например, комбинация «Ctrl-n» относится к браузеру (создает новое окно) и отдельной вкладкой не может быть отменена. Также коды вкладки не повлияют на системные комбинации, на подобие «Ctrl-Esc» или «Ctrl-Alt-Del». Вмешательство в работу таких комбинаций требует применение средств системного программирования и выходит за возможности JavaScript, ограниченного отдельным браузером.

Домашнее задание

1. Разместите на странице 4 блока, как показано на рисунке. Напишите инструкции, меняющие цвет нижнего блока в зависимости от того, на каком из верхних блоков находится указатель мыши. Если указатель не находится ни над одним из верхних блоков, нижний блок должен приобрести серый цвет.

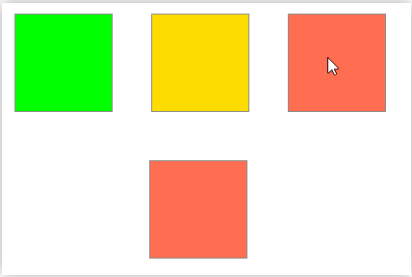


Рисунок 29

2. Разместите на странице один блок. Напишите инструкции, которые обеспечат перемещение блока при нажатии левой клавиши мыши в точку, в которой находится указатель мыши.
3. Разместите на странице 3 блока. Напишите инструкции, которые обеспечат перемещение в точку, в которой находится указатель мыши: первого блока — при нажатии левой клавиши мыши, второго блока — при нажатии средней клавиши (или колеса), третьего блока — при нажатии правой клавиши мыши. Контекстное меню при нажатии правой клавиши мыши появляться не должно.

Домашнее задание