

## Лабораторная работа 6. Методы для работы с атрибутами и фильтрами

**attr(name)** - обеспечивает доступ к значению указанного атрибута первого элемента в наборе.

**Пример:**

```
var a=$("#i").attr("title");  
$("#div").text(a);
```

Данная инструкция найдет первый элемент в тегах i, найдет атрибут title этого элемента и добавит его значение в блок.

**attr(properties)** - установит атрибуты во всех отобранных элементах.

**Пример:**

```
$("#img").attr({src:"images/pict.gif", alt:"рисунок"});
```

Данная инструкция найдет все картинки и установит им соответствующие атрибуты.

**attr(key,value)** - установит значение (value) атрибута (key) для всех отобранных элементов.

**Пример:**

```
$("#button").attr("disabled", "disabled");
```

Данная инструкция установит для всех кнопок значение "disabled" атрибута "disabled".

**removeAttr(name)** - удалит указанный атрибут у всех элементов.

**Пример:**

```
$("#img").removeAttr("alt");
```

Данная инструкция удалит атрибут "alt" у всех картинок.

**hasClass(class)** - возвращает истину (true), если указанный класс присутствует хотя бы в одном из элементов.

**Пример:**

```
if ($("#p:last").hasClass("selected"))  
    $(this).css("background", "blue");
```

Данная инструкция сделает цвет фона последнего абзаца синим, если у него класс "selected".

**filter(expr)** - ограничивает элементы, к которым следует что-либо применить.

**Пример:**

```
$("#p").filter(".blue").css("background", "blue");
```

Данная инструкция сделает цвет фона синим, только у тех абзацев, которые имеют класс "blue".

**not(expr)** - обозначает элементы, к которым не следует что-либо применить.

**Пример:**

```
$("#p").not(".blue").css("background", "blue");
```

Данная инструкция сделает цвет фона синим у всех параграфов кроме тех, что имеют класс "blue".

**is(expr)** - возвращает истину (true), если хотя бы один из элементов соответствует выражению.

**Пример:**

```
if ($(this).is(":last-child"))  
    $("#p").text("последний");
```

Данная инструкция добавит в параграф текст "последний", только если проверяемый элемент последний.

`slice(start, end)` - отбирает поднабор из набора элементов.

**Пример:**

```
$("p").slice(1,4).css("background","blue");
```

Данная инструкция сделает цвет фона синим у всех параграфов с 1 по 4.

**Задание 1.** «Расставить» на шахматной доске, созданной ранее, шашки по нажатию второй кнопки. Для этого создать функцию, которая будет использовать метод `filter`, для выбора блоков-клеток черного цвета и функцию `slice`, которая расставит в клетки с 0-й по 12-ю черные шашки, а с 20 по 32 – белые шашки при помощи метода `append`.

Например, так:

```
function имя() {
$("блок").filter("клетка").slice(0,12).append('');
...
}
```

Обратите внимание, здесь мы используем цепочку вызовов: сначала отбираем все темные клетки (`filter()`), а затем уже из них отбираем необходимые (`slice()`) и в них вставляем картинки с шашками (`append()`).

## Лабораторная работа 7. Методы по обработке событий

**ready (fn)** - назначает функцию, которая будет выполняться, когда документ готов к работе.

**Пример:**

```
$(document).ready(init);
```

Данная инструкция говорит браузеру, что сразу после загрузки документа должна сработать функция `init`.

**bind(type, fn)** - связывает обработчик события с самим событием.

**Пример:**

```
$(div).bind('click', init);
```

Данная инструкция говорит браузеру, что при щелчке по блоку должна сработать функция `init` (здесь `click` - событие, а `init` - функция, обработчик события).

**Задание 1.** Создадим список уроков по jQuery и сделаем так, чтобы при клике по какому-нибудь из них, в поле ниже появлялось описание выбранного урока.

Создадим список и поле для вывода описаний на html-странице по образцу(рисунок 16). Для начала достаточно добавить в список 3 урока.

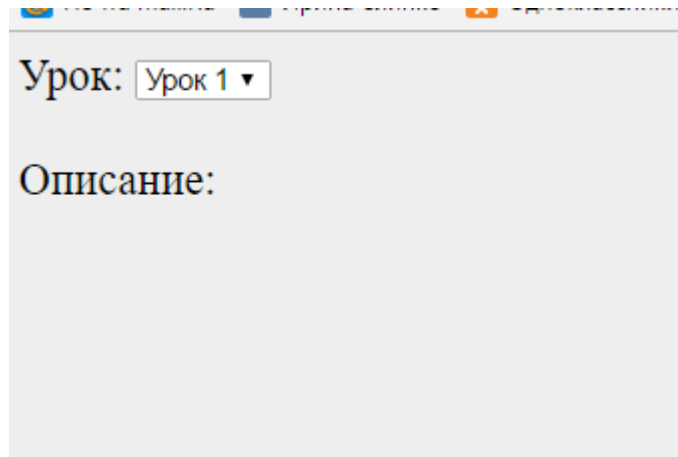


Рисунок 16. Исходная страница

Мы могли бы каждому пункту списка задать обработчик события - функцию, которая и будет задавать описания в зависимости от выбранного пункта. Но jQuery позволяет сделать это еще проще.

На странице `script.js` нам достаточно назначить функцию, которая будет срабатывать каждый раз, когда документ готов к работе (назовем ее `init`). Сама же функция `init` будет вызывать функцию каждый раз, когда выбран какой-либо пункт списка (т.е. наступает событие `change`).

Кроме того применим конструкцию `switch`, заменяет собой сразу несколько `if`. Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

Выглядит она так:

```
switch(x) {  
  case 'value1': // if (x === 'value1')  
    ...  
    [break]  
  
  case 'value2': // if (x === 'value2')  
    ...  
    [break]
```

```
default:
    ...
    [break]
}
```

- Переменная `x` проверяется на строгое равенство первому значению `value1`, затем второму `value2` и так далее.
- Если соответствие установлено – `switch` начинает выполняться от соответствующей директивы `case` и далее, до ближайшего `break` (или до конца `switch`).
- Если ни один `case` не совпал – выполняется (если есть) вариант `default`.

При этом `case` называют *вариантами switch*.

```
Урок: <select class="lesson">
    <option value='1'>Урок 1</option>
    <option value='2'>Урок 2</option>
    <option value='3'>Урок 3</option>
</select><br><br>
Описание: <div class="desc"></div>
```

```
$(document).ready(init);

function init(){
    $('.lesson').bind('change', desc);
}

function desc(){
    var op=$('.lesson').val();
```

```
switch (op)
{
    case '1': $('.desc').text('Первый урок по jQuery знакомит с основными понятиями и возможностями этой библиотеки.');
```

знакомит с основными понятиями и возможностями этой библиотеки.');

```
    break;
    case '2': $('.desc').text('Второй урок по jQuery знакомит с таким понятием, как селекторы.');
```

знакомит с таким понятием, как селекторы.');

```
    break;
    case '3': $('.desc').text('Третий урок по jQuery знакомит с таким понятием, как фильтры.');
```

знакомит с таким понятием, как фильтры.');

```
    break;
}
```

Другие методы обработки событий:

**one(type, fn)** - связывает обработчик события с самим событием, но выполняется он только один раз.

**Пример:**

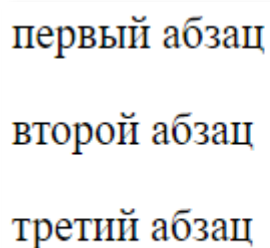
```
$(div).one('click', init);
```

Данная инструкция говорит браузеру, что при щелчке по блоку должна сработать функция `init`, но инструкции этой функции будут выполнены только один раз.

Если в нашем примере со списком уроков мы заменим `bind` на `one`, то функция `desc` сработает только один раз, т.е. при первом выборе урока из списка мы увидим описание, но далее, сколько бы мы не щелкали по пунктам списка, описание меняться не будет.

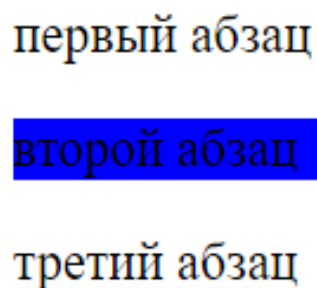
**hover(over, out)** - когда указатель мыши находится над объектом, срабатывает функция `over`, а когда указатель мыши выходит за объект, то функция `out`.

**Задание 2.** Создайте страницу по образцу(рисунок 17)



первый абзац  
второй абзац  
третий абзац

Рисунок 17. Исходная страница



первый абзац  
второй абзац  
третий абзац

Рисунок 18. Результат работы функции

Таким образом, когда указатель мыши над абзацем, его фон - синий, а при выходе - белый.

Можно применять краткую запись, т.е. использовать функции без названий или дать функциям имена. Например, так:

```
$(document).ready(init);  
function init(){  
  $('p').hover(hOver, hOut);
```



```

}

function hOver() {
...
}
function hOut() {
...
}

```

Выбирайте ту форму записи, которая вам больше нравится.

**toggle(fn1, fn2, ...fn)** - переключатель между функциями. Щелчок по элементу вызывает функцию fn1, повторный щелчок - функцию fn2, третий щелчок - функцию fn3 и т.д. **НаПример:**

```

$('селектор').toggle(
function(){$(this).css("свойство1", "значение1");},
function(){$(this).css("свойство1", "значение2");},
function(){$(this).css("свойство2", "значение3");},
)
);

```

Данная инструкция говорит браузеру, что при щелчке по абзацу, его фон станет синим, при повторном щелчке - белым.

**click(fn)** - функция fn связывается с событием click.

**Пример:**

```

$('div').click(

```

```
function(){$(this).css("background-color", "blue");}  
);
```

Данная инструкция говорит браузеру, что при щелчке по блоку, его фон станет синим.

**click()** - эмулируется возникновение события `click`.

### Пример:

```
$('div').click(  
function(){$(this).css("background-color", "blue");}  
);  
$('div:first').click();
```

Данная инструкция говорит браузеру, что при щелчке по блоку его фон станет синим и эмулирует это событие для первого блока.

Иными словами, вызов метода с функцией в качестве аргумента, например `click(fn)`, назначает обработчик события, вызов без аргумента, например `click()`, эмулирует возникновение этого события.

Аналогичные методы определены и для других событий, поддерживаемых javascript, например: `blur()`, `blur(fn)`, `focus()`, `focus(fn)` и т.д.

**Задание 3.** К списку уроков добавьте список ответов в форме теста(можно на новой, связанной с главной, страницей контроля знаний (переход по ссылке). При наведении курсора вопросы должны изменять внешний вид(подсветка, фон, шрифт...на Ваше усмотрение).Переход по ссылке на страницу контроля знаний допустим только один раз.

**Задание 4.** Для работы с шахматной доской нужно исправить функции добавления шахматного поля и расстановки шашек так, чтобы они добавлялись только один раз.

**Задание 5.** Создайте еще несколько абзацев в задании 2. Напишите функции для методов `toggle()`, `click(fn)`, `click()`, `blur()`, `blur(fn)`, `focus()`, `focus(fn)`

## Лабораторная работа 8. Визуальные эффекты. Методы видимости - `hide()`, `show()` и `toggle()`

Эти методы отвечают за видимость элементов, работают по принципу скрыть - показать:

`hide()` - скрыть,

`show()` - показать,

`toggle()` - показать, если скрыты и скрыть, если видимы.

### Задание 1.

- 1) Создаем простой блок, заливаем его любым цветом и назначаем для него класс или идентификатор.
- 2) Создаем две кнопки и для каждой назначаем свою функцию на клик, например, `hideDiv()` и `showDiv()`.

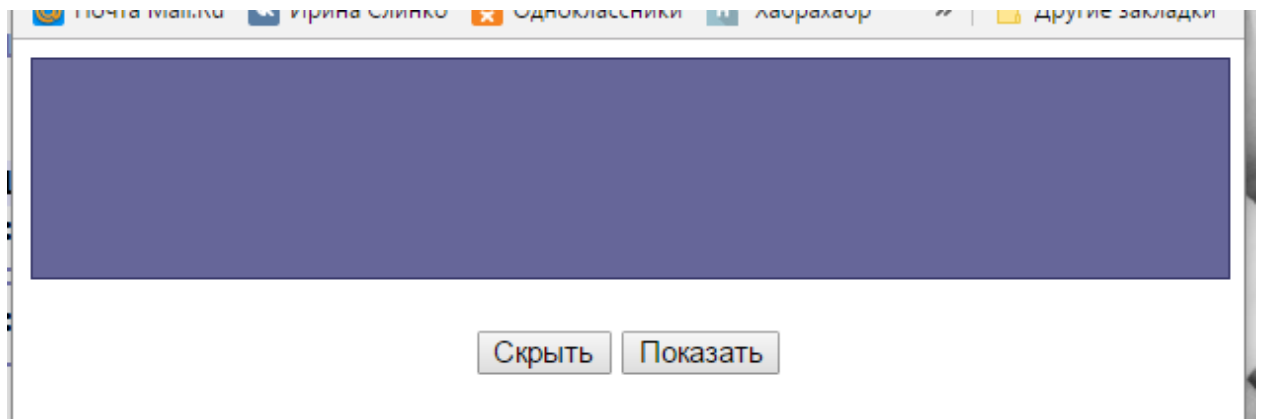


Рисунок 19. Исходная страница

Код функций (на странице `script.js`):

```
function hideDiv() {  
    $('селектор').hide();  
}  
function showDiv() {  
    $('селектор').show();  
}
```

Скрыть элемент можно и при помощи таблицы стилей, а не методом `hide()`.

Например, если бы мы хотели, чтобы при загрузке страницы наш блок был скрыт, то мы могли бы это сделать двумя способами:

Первый, использовать метод `hide()`:

```
$(document).ready(init);

function init(){
$('селектор').hide();
}
function hideDiv(){
$('селектор').hide();
}
function showDiv(){
$('селектор').show();
}
```

А второй - использовать свойство `display` CSS.

Результат будет одинаков.

Эти же методы можно использовать с анимацией:

`hide(speed, callback)` - скрыть,

`show(speed, callback)` - показать,

**`toggle`**(`speed, callback`) - переключить (показать, если скрыты и наоборот), где:

`speed` - скорость изменения высоты, ширины или свойства `opacity` (прозрачность) элемента. Может принимать три значения: `slow` (медленно), `normal` (нормально) или `fast` (быстро), а также значение в миллисекундах.

`callback` - функция, которая будет выполняться после завершения анимации. Ее присутствие вовсе необязательно.

**toggle** поочередно выполняет одно из нескольких заданных действий.

Имеет четыре варианта использования:

```
.toggle(handler1(eventObject), handler2(eventObject),  
[handler3(eventObject)])
```

Поочередно выполняет одну из двух или более заданных функций **handler**, в ответ на "клик" по элементу.

```
.toggle([duration], [callback])
```

Изменяет видимость выбранных элементов на противоположную (показывает/скрывает).

**duration** — продолжительность выполнения анимации. Может быть задана в миллисекундах или строковым значением 'fast' или 'slow' (200 и 600 миллисекунд).

**callback** — функция, которая будет вызвана после завершения анимации.

```
.toggle([duration], [easing], [callback])
```

**duration** — см. выше.

**easing** — изменение скорости появления/исчезновения (будет ли она замедляется к концу выполнения или наоборот ускорится).

**callback** — см. выше.

```
.toggle(showOrHide)
```

Только показывает (**showOrHide = true**) или только убирает с экрана (**showOrHide = false**) выбранные элементы на странице.

**Задание 2.** Для задания 1 применим анимацию. Скрывать и отображать блок будем медленно, используя метод `toggle()`. Для новой страницы создадим блок и одну кнопку (рисунок 20).

Примечание: для того, чтобы блок сворачивался и разворачивался относительно вертикальной оси страницы (или родительского элемента) нужно для его родительского элемента задать стиль выравнивания

```
text-align: -webkit-center;
```

Функция может выглядеть так

```
function hideShowDiv(){  
$('селектор').toggle('slow');  
}
```

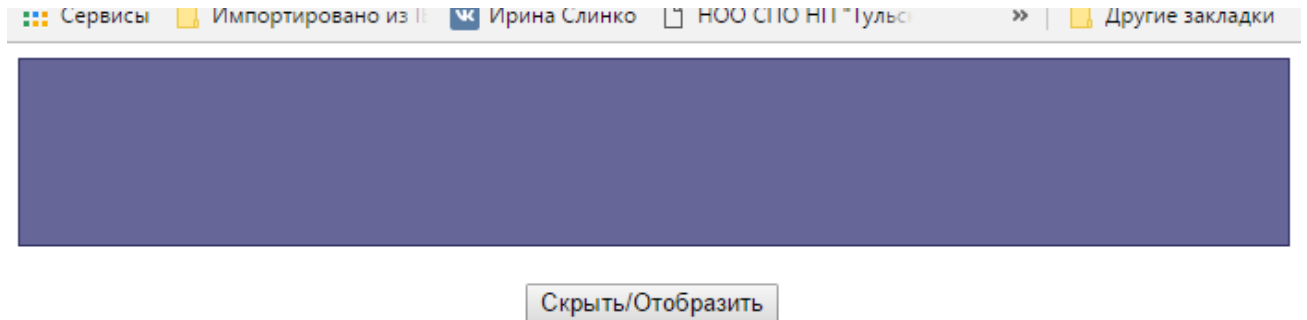


Рисунок 20. Исходная страница

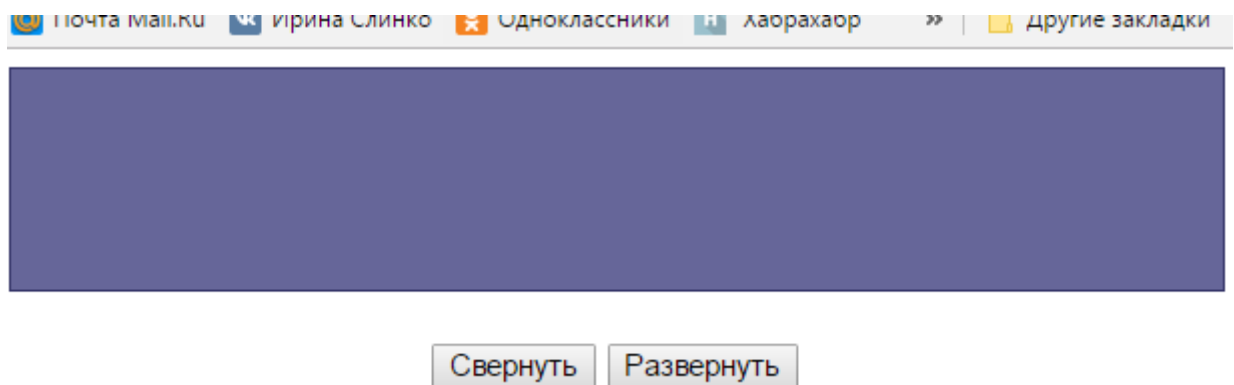
### Методы свертывания - `slideDown()`, `slideUp()` и `slideToggle()`

Эти методы также отвечают за видимость элементов, но работают по принципу свернуть элемент плавно снизу-вверх - развернуть элемент плавно сверху-вниз:

`slideUp()` - свернуть,

`slideDown()` - развернуть,

`toggle()` - развернуть, если скрыт и свернуть, если видим.



Эти методы также можно использовать с анимацией:

`slideUp(speed, callback)` - свернуть,

`slideDown(speed, callback)` - развернуть,

`slideToggle(speed, callback)` - переключить (развернуть, если скрыты и наоборот), где:

`speed` - скорость изменения высоты элемента. Может принимать три значения: `slow` (медленно), `normal` (нормально) или `fast` (быстро), а также значение в миллисекундах.

`callback` - функция, которая будет выполняться после завершения анимации. Ее присутствие необязательно.

**Задание 3.** Для предыдущих заданий, напишите функцию, которая будет скрывать и отображать блок за 7 секунд, используя эти методы.

Ситаксис функций может быть таким:

```
function slideToggleDiv() {
```



```
$('селектор').slideToggle(7000);  
}
```

### Методы исчезновения - **fadeTo()**, **fadeOut()** и **fadeIn()**

**fadeTo(speed, opacity, callback)** - уменьшает свойство `opacity` (прозрачность) к заданному значению,

**fadeOut(speed, callback)** - уменьшает свойство `opacity` (прозрачность),

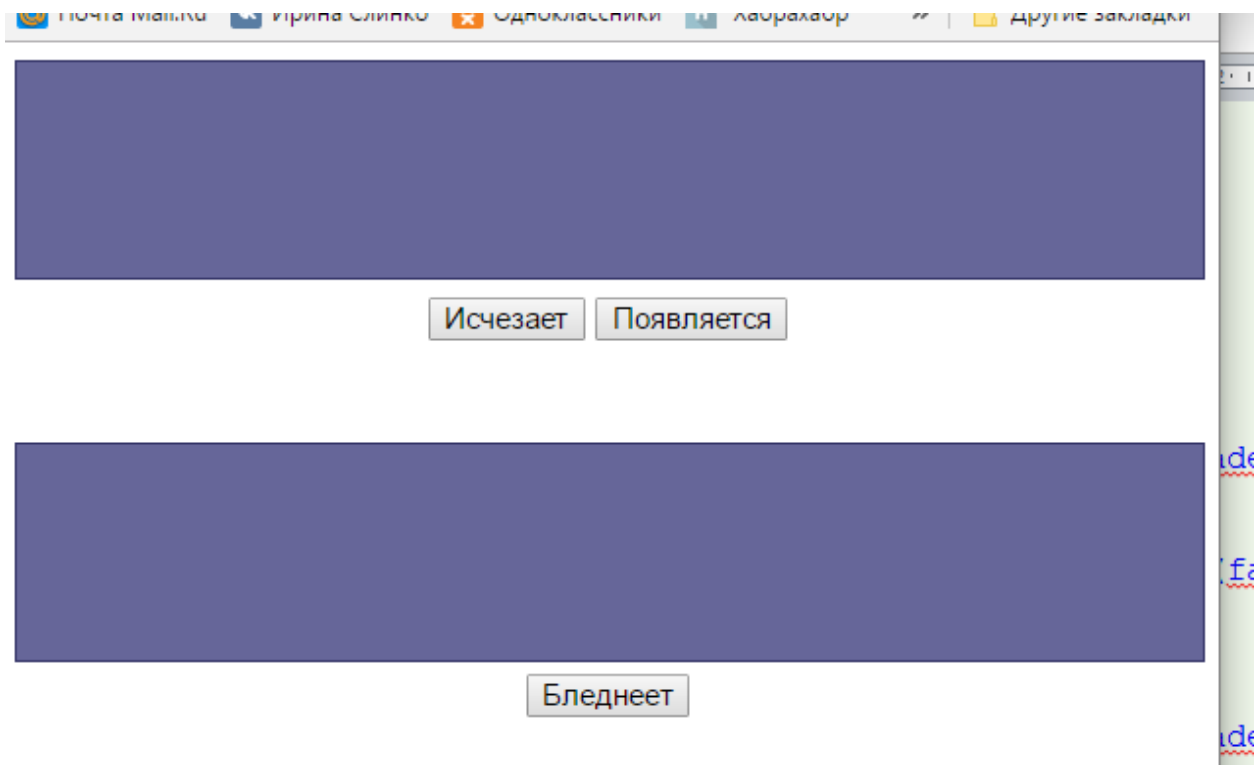
**fadeIn(speed, callback)** - увеличивает свойство `opacity` (прозрачность), где:

**speed** - скорость изменения прозрачности элемента. Может принимать три значения: `slow` (медленно), `normal` (нормально) или `fast` (быстро), а также значение в миллисекундах.

**opacity** - значение прозрачности, до которого оно будет уменьшено (число от 0 до 1).

**callback** - функция, которая будет выполняться после завершения анимации. Ее присутствие необязательно.

### Задание 4:



Код функций (на странице script.js):

```
function имя1 () {  
    $('селектор').fadeOut(5000);  
}  
function имя2 () {  
    $('селектор').fadeIn(5000);  
}  
function имя3 () {  
    $('селектор').fadeTo(5000, 0.5);  
}
```

## Лабораторная работа 9. Методы анимации - `animate()` и `stop()`

**`animate(params, options)`** - создает эффект анимации на любое числовое CSS свойство элемента. Единственным необходимым параметром

является объект с CSS свойствами. Этот объект похож на тот, что передается в метод `.css()`, за исключением того, что диапазон свойств ограничен.

`params` - атрибуты css, которые хотим анимировать ("width", "top", "border"..и др.).

`options` - свойства анимации (в том числе скорость).

`stop()` - останавливает анимацию.

### *Анимационные свойства и значения*

Все анимационные свойства должны быть анимированы при помощи *единичного числового значения*, за исключением случаев указанных ниже. Большинство не числовых свойств не могут быть анимированы при помощи базового функционала jQuery (Например width, height или left могут быть анимированы, но background-color не может, за исключением использования плагина jQuery.Color). Значения свойств рассматриваются как количество пикселей, если не указано иное. Единицы измерения em и % могут быть указаны, там, где это применимо.

В дополнение к свойствам стиля, некоторые не стилевые свойства, такие как scrollTop и scrollLeft, а также пользовательские свойства, могут быть анимированы.

Сокращенные CSS свойства (такие как font, background, border) не полностью поддерживаются. Например, если Вы хотите анимировать ширину границ, то стиль и изначальная ширина границы должны быть заданы заранее. Или, если Вы хотите анимировать размер шрифта, Вы должны использовать fontSize или CSS эквивалент 'font-size' а не просто 'font'.

В дополнение к числовым значениям, каждое свойство может принять строку: 'show', 'hide' или 'toggle'. Эти сокращения позволяют произвольную анимацию скрытия или отображения, которая учитывает тип отображения элемента. Для того чтобы использовать встроенное отслеживание состояния отображения свойства, строка 'toggle' должна быть задана в качестве значения анимируемого CSS свойства.

Анимированные свойства также могут быть относительными. Если значение задано с ведущей последовательностью символов **+=** или **-=**, то целевое значение вычисляется путем сложения или вычитания заданного числа от текущего значения свойства.

**Важно:** В отличие от сокращенных методов анимации, таких как `.slideDown()` и `.fadeIn()`, метод `.animate()` *не делает* скрытие элемента как часть эффекта. Например, в случае

`$("#someElement").hide().animate({height: "20px"}, 500)`, анимация будет запущена, но элемент *будет оставаться скрытым*.

### *Длительность*

Длительность задается в миллисекундах. Более высокие значения означает более медленную анимацию, а не быструю. Значение по умолчанию 400 миллисекунд. Строки 'fast' и 'slow' могут быть использованы для указания длительности 200 и 600 миллисекунд соответственно.

### *Функции обратного вызова*

Если заданы start, step, progress, complete, done, fail и always функции обратного вызова, то они будут вызваны для *каждого* анимируемого элемента; this в этих функциях будет указывать на анимируемый DOM-элемент. Если в выборке нет элементов, то функции обратного вызова не будут выполняться. Если анимируются несколько элементов, то обратный вызов выполняется один раз для каждого элемента, а

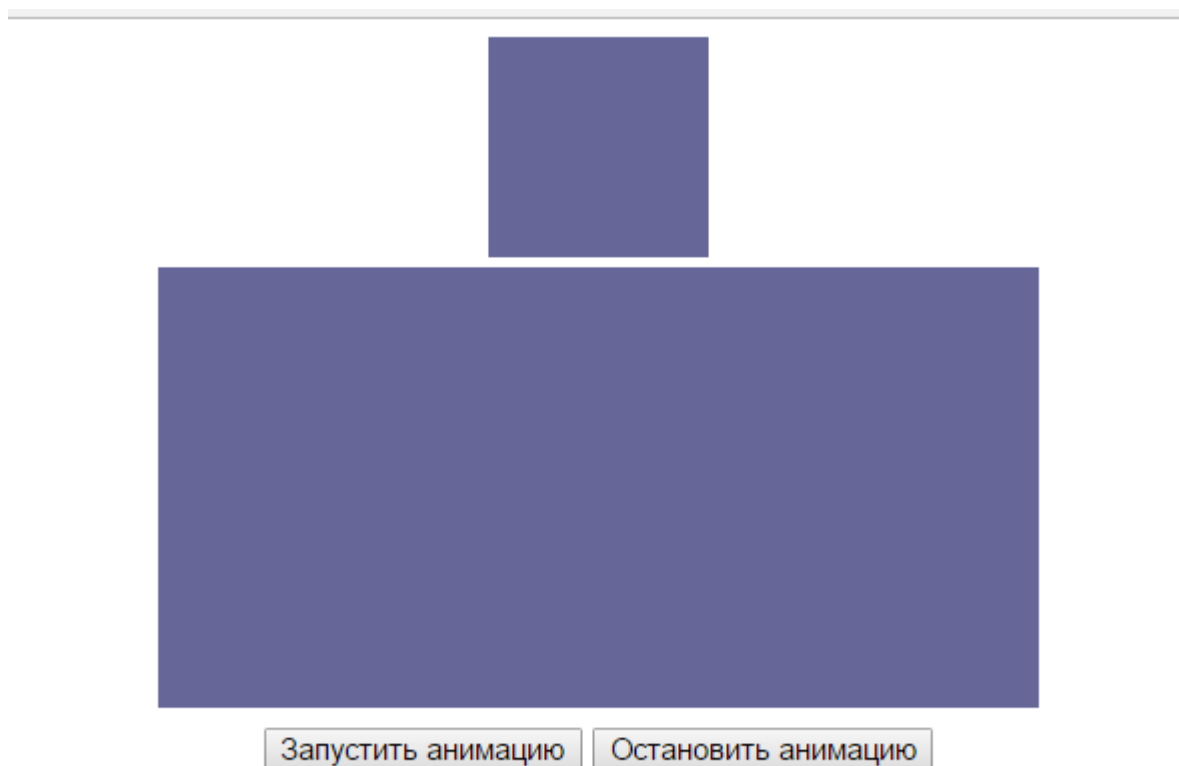
не один раз для анимации в целом. Используйте метод `.promise()` для получения объекта `Promise` и прикрепления функций обратных вызовов.

**Задание 1.** Создайте страницу по образцу:

Два блока с разными стилями.

Для блоков задать в стилях ширину и высоту.

Для кнопок создать функции для запуска и остановки анимации.



Примерный код функций (на странице `script.js`):

```
function имя1() {  
  $('блок1').animate({  
    width: "400px",  
    height: "200px"  
  }, 3000 );  
  $('блок2').animate({
```

```
width:"100px",  
height:"100px"  
, 3000 );  
}  
function имя() {  
$('блок 1').stop();  
$('блок2').stop();  
}
```

Несколько нюансов:

- свойства должны быть обозначены без пробелов, последующее слово с большой буквы, т.е. "borderWidth" вместо "border-width",
- поддерживаются только те свойства, значения которых выражаются числами.
- также в качестве значений свойств могут быть использованы значения "hide", "show" и "toggle".

Например, следующая функция:

```
function animateDiv() {  
$('селектор').animate({  
"height": "toggle"  
}, 1000 );  
}
```

будет при каждом обращении менять высоту от 0 до заданной в стилях и обратно.

**Задание 2.** Задайте анимацию для изображения(можно выбрать любое).

Для этого создайте два блока в одном из них разместим текст, в другом - изображение.

Для изображения нужно задать параметры, с которыми будет работать скрипт: ширину, высоту, относительное позиционирование и сдвинем его на 10 пикселей от левого края.

Зададим одновременную анимацию прозрачности, отступа слева и высоты картинки.

Скрипт будет следующим:

```
$(document).ready(function() {
    $(".clickme").click(function() {
        $(".im_1").animate({
            opacity: 0.25,
            left: "+=50",
            height: "toggle"
        }, 5000, function() {

        });
    });
});
```

Обратите внимание, что целевое значение свойства `height` равно `'toggle'`. Так как изображение было видно прежде, анимация сокращает высоту до 0 чтобы скрыть его. Второе нажатие затем обратит этот переход:

Свойство `opacity` картинки уже равно его целевому значению, так что это свойство не анимируется при втором клике. Так как значение свойства `left` является относительным значением, то картинка сдвигается далее направо во время второй анимации.

**Нажмите здесь**



### **Функция исчезания (Easing)**

Еще один параметр метода `.animate()` это строка с именем используемой метода исчезания (easing). Метод `easing` определяет скорость с которой анимация прогрессирует в различных точках в пределах анимации.

В базовом функционале jQuery поставляется дв такие функции: `swing`, используемая по умолчанию, и `linear` с равномерным изменением анимации.

/\*Больше easing функций доступны при использовании плагинов, в первую очередь jQuery UI suite.\*/

### **Функция исчезания для каждого свойства**

Можно также установить метод `easing` по отдельности для каждого анимируемого свойства при вызове `.animate()`. В первой версии метода `.animate()`, каждое свойство может принимать массив в качестве значений: первый член массива является названием CSS свойства, второй член массива имя функции. Если функция не определена для определенного свойства, то используется значение из метода `.animate()` или его значение



по умолчанию. Если аргумент не установлен, то используется значение по умолчанию `swing`.

Например, чтобы одновременно анимировать ширину и высоту с `swing easing` функцией и прозрачность с `linear` функцией:

**Задание 3.** Создайте копию задания 2. Зададим одновременную анимацию ширины и высоты с `swing` методом и прозрачность с `linear` методом.

Обратите внимание, что числовые параметры элементов не используются.

```
$( "блок1" ).click(function() {
    $( "#book" ).animate({
        width: [ "toggle", "swing" ],
        height: [ "toggle", "swing" ],
        opacity: "toggle"
    }, 5000, "linear", function() {
        $( this ).after( "<div>Анимация завершена.</div>"
    );
    });
});
```

**Задание 4.** Создадим страницу на которой покажем свойство `left` DIV элемента с относительным значением. Анимация должна работать при каждом нажатии на кнопки, это называют относительной анимацией в очереди.



Создайте две кнопки(с надписями `&laquo` и `&raquo` для стрелочек) и блок.

Скрипт будет примерно следующим:

```
$( ".right" ).click(function() {
    $( ".block" ).animate({ "left": "+=50px" }, "slow" );
});
```

```
$( ".left" ).click(function(){  
    $( ".block" ).animate({ "left": "-=50px" }, "slow" );  
});
```