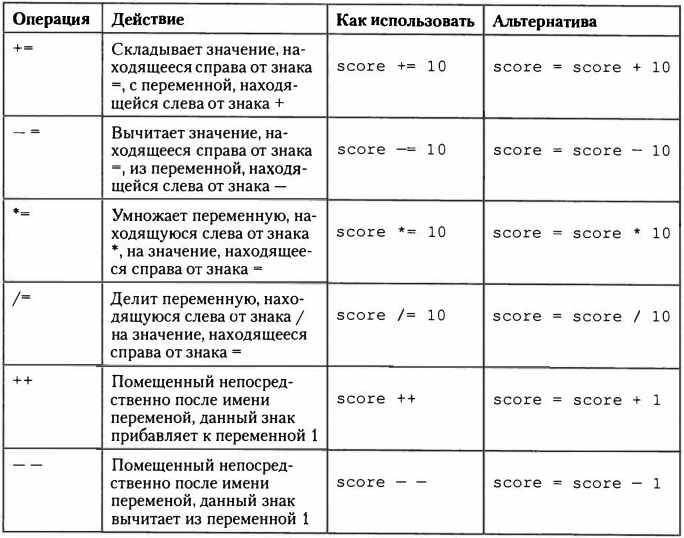
**ПРЕОБРАЗОВАНИЕ СТРОКИ В ЧИСЛО**

var numOfShoes = '2';

var nurnOfSocks = 4;

var totalitems = **Number(numOfShoes)** + numOfSocks; или **+numOfShoes** без пробела

**Математические Операции**



**Массивы**

Старый способ:

**var days = new Аrrау('Пн', 'Вт', 'Ср');**

Более приемлемый:

**var days = [ 'Пн', 'Вт', 'Ср', 'Чт'] ;**

**var playList = [];**

узнать количество переменных в массиве

**days.length**

доступ к последнему элементу :

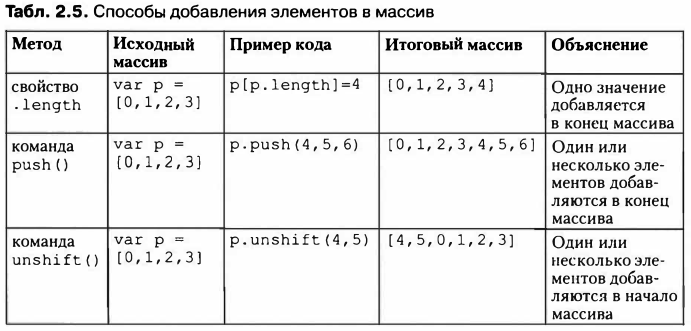
**days[days.length -1]** -1 используем в следствии того, что первый элемент начинается с **0** а не **1**

**days[days.length ] = ' Пт ';**  добавит в конец массива еще одно значение

**days.push(' Пт ');** тоже добавит еще одно значение в конец массива

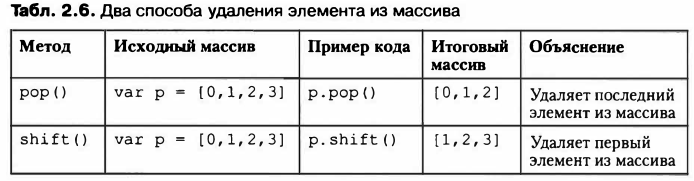
**days.push('Пт', 'Сб', 'Вс');** добавит в конец новые 3 значения

**days.shift**(**'Пт', 'Сб', 'Вс');** Добавит значение в самое начало массива

****

**Удаление элементов из массива**

Если вы хотите удалить элемент из начала или из конца массива, используйте методы **рор ()** или **shift ()**



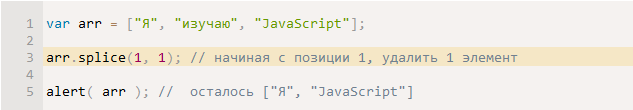
Универсальный метод **splice**

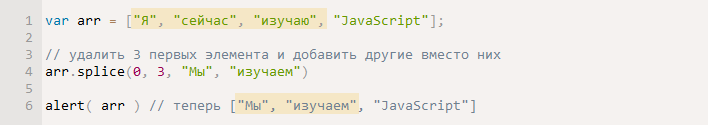
Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно.

Его синтаксис:

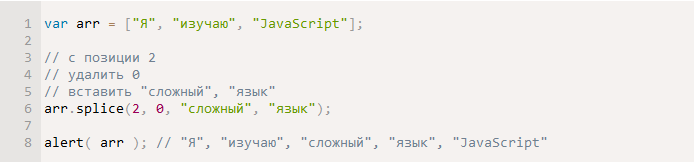
**arr.splice(index[, deleteCount, elem1, ..., elemN])**

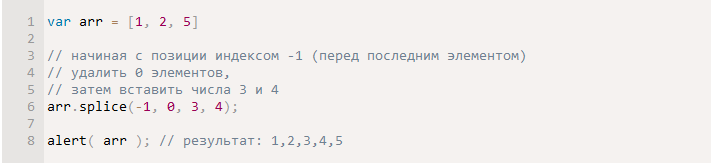
Удалить **deleteCount** элементов, начиная с номера **index**, а затем вставить **elem1**, ..., **elemN** на их место. Возвращает массив из удалённых элементов.





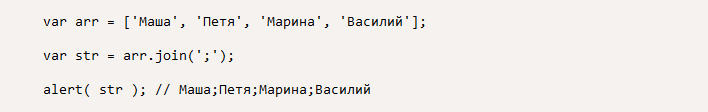




Допускается использование отрицательного номера позиции, которая в этом случае отсчитывается с конца:

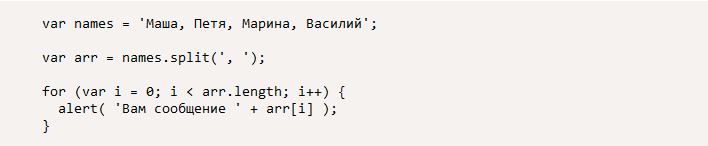
[Метод **join**](https://learn.javascript.ru/array-methods#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-join)

Вызов arr.join(str). Он берет массив и склеивает его в строку, используя **str** как разделитель.



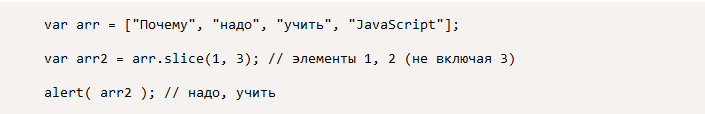
[Метод](https://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-split) **[split](https://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-split)**

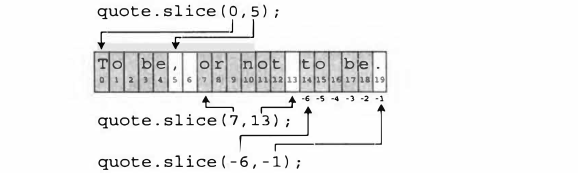
Метод split(s), который позволяет превратить строку в массив, разбив ее по разделителю s. В примере ниже таким разделителем является строка из запятой и пробела.



[Метод](https://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-slice) **[slice](https://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-slice)**

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

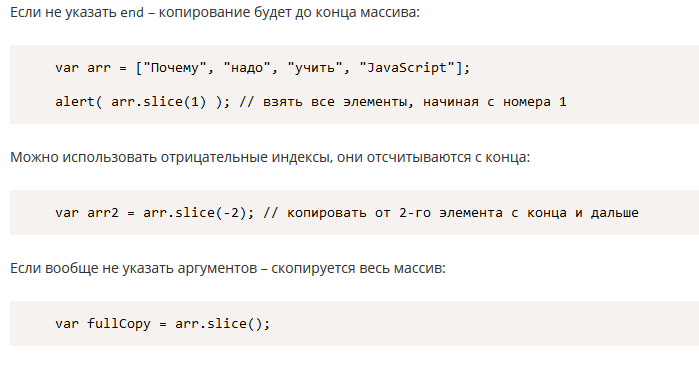




Метод **pop()**

Возвращает первый элемент массива и удаляет его, тем самым изменяя длину массива

var array = [10,1,2,3], firstItemArray = array.pop(); // firstItemArray = 10 и array = [1,2,3]

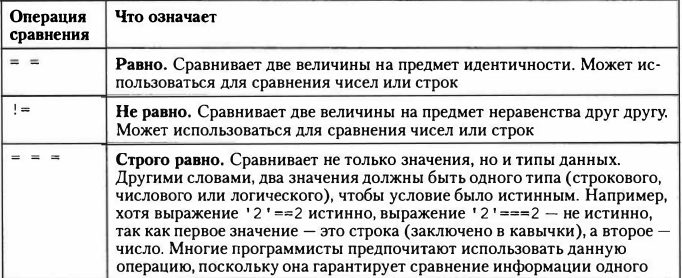


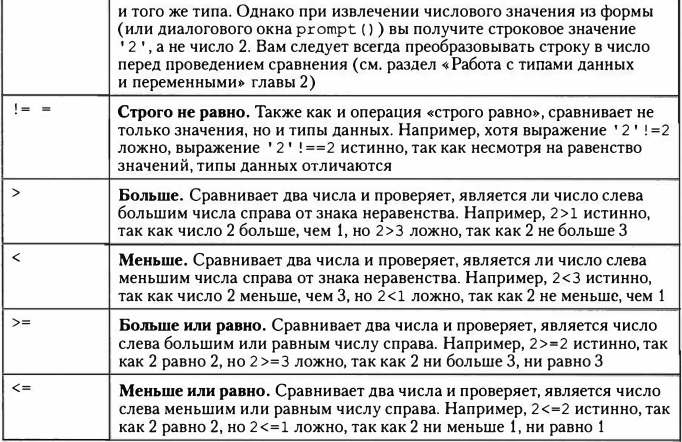
**Управляющие инструкции**

**if ( условие) {**

// Здесь совершается некое действие

**}**

****

****

**if (answer == 31) {**

**alert('Bepнo. У Сатурна 31 луна.');**

**numCorrect = numCorrect + 1;**

**}**

**else {**

**alert ("Неверно! У Сатурна совсем не столько лун. ");**

**}**

**------------------------------------------------------------------------------------------------------------**

**var fridayCash = prompt ('Сколько денег вы можете потратить?', '');**

**if (**fridayCash >= 1500**) {**

alert('Bы можете и поужинать, и пойти в кино.');

**}**

**else if** (fridayCash >= 1000**) {**

alert('Bы можете поужинать.');

**}**

**else if (**fridayCash >= 500**)**

alert('Bы можете пойти в кино.');

**}**

**else {**

alert('Пoxoжe, вы будете смотреть телевизор.');

**}**

**&& логическое «И»**

**if (**а > 1 **&&** а < 10**) {**

**// Значение а находится между 1 и 10**

аlеrt("Значение " + а + " принадлежит промежутку .-J

от 1 до 10");

**}**

Вы не ограничены всего двумя условиями. С помощью операции & & можно объединять сколько угодно условий:

**if** (Ь>0 **&&** а>0 **&&** с>0**) {**

**// Все три переменные больше 0**

**…**

**}**

Этот код проверяет три переменные, чтобы убедиться, что все они больше 0. Если хотя бы одна из них равна или меньше 0, код в скобках не будет выполнен.

**|| логическое «ИЛИ»**

**i f** ( key == 'n' **||** key == 'N' ) {

// Переход к следующему фото

}

Как и в случае с операцией логического “и”, вы можете сравнивать больше двух условий.

**if** (gas <= 0 **||** time <= 0 **||** cars <= 0) {

//игра окончена

}

При проверке множественных условий часто бывает сложно понять логику управляющих инструкций. Некоторые программисты группируют каждый набор условий в паре скобок, чтобы сделать логику более легкой для понимания:

**if** ( ( key == 'n' ) **||** ( key == 'N' ) ) {

//Переход к следующему фото

}

**! логическое «НЕ»**

**«НЕ»** может использоваться и сама по себе, чтобы изменить результат управляющей инструкции на противоположный, другими словами, с его помощью можно превратить истинное выражение в ложное, а ложное в истинное.

Операцию логического НЕ следует использовать, если вы хотите выполнить какой-либо код а основании отрицательного условия. Например, вы создали переменную с именем valid, содержащую логическое значение **true** или **false.**

Вы используете эту переменную, чтобы проверить, правильно ли посетитель заполнил форму ( например, поле не может быть пустым, в нем должен быть адрес электронной почты).

Если возникает проблема (допустим, поле пустое), вы можете изменить значение переменной **valid** на “ложь”: **valid = false**.

если вы хотите вывести на экран сообщение об ошибке и предотвратить отправку формы, то вы можете написать следующую управляющую инструкцию:

**if (!**valid**) {**

// Вывести на экран сообщение об ошибке и

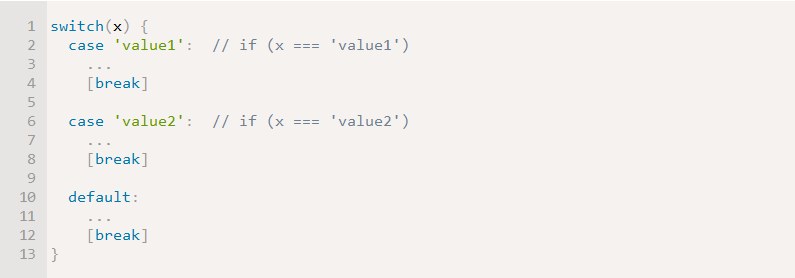
//не отсылать форму

**}**

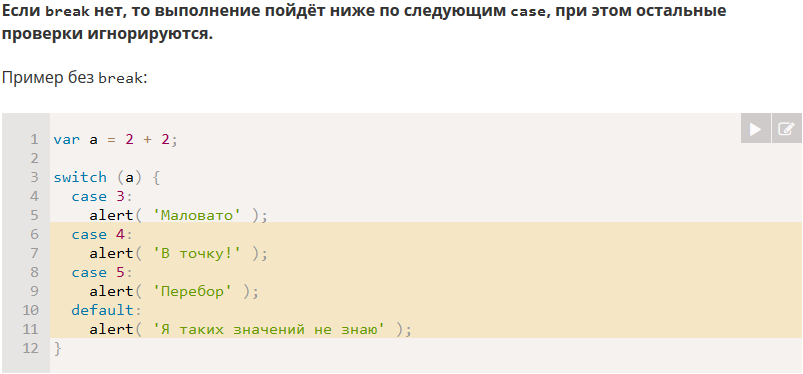
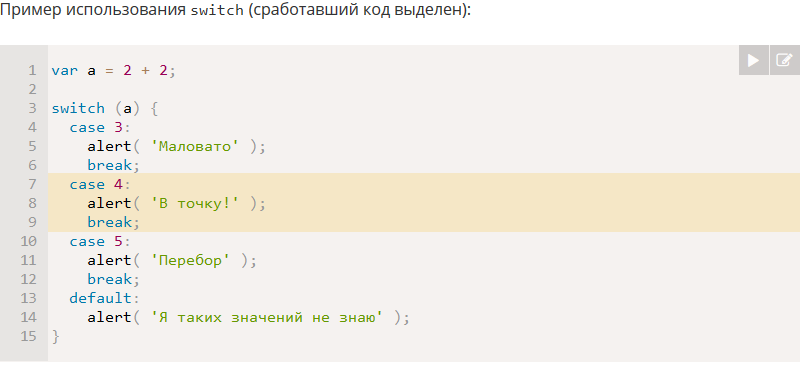
Условие **!valid** можно перевести как если не действительно, что означает: если значение переменной **valid** ложно, то условие истинно. Чтобы выяснить результат условия, использующего операцию логического «НЕ» просто оцените условие без операции «НЕ», а потом измените его на противоположное. Другими словами, если результатом условия является true, то операция **!** изменяет его на значение **false**, поэтому управляющая инструкция не выполняется.

# Конструкция switch

Конструкция switch заменяет собой сразу несколько if.



* 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**.



Циклы **whilе**

**while** (условие) {

// повторяющийся код javascript

}

**var num = 1;**

**while** (num <= 100) {

document.write('<p>Чиcлo ' + nurn + '</p><br>');

num += 1; // Или можем записать num ++

}

**Циклы и массивы**

**var days** = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];

**var counter** = 0;

**while** (counter < days.length) {

docurnent.write(days[counter] + ', ');

counter++;

}

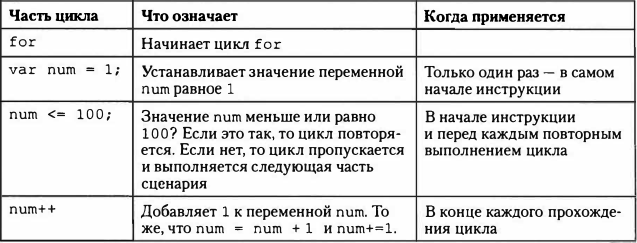
Выведет последовательно дни с понедельника по воскресенье

**Циклы FOR**

**for** (var num = 1; num <= 100; num++) {

document.write('Чиcлo ' + num + ' <br>');

}



var example = [ 'первый' , 'второй' , 'третий' , 'последний' ] ;

**for** (var j = example.length - 1 ; j >= 0; j--) {

document.write(example[j] + '<br>');

}

**Циклы do/while**

**do {**

**// повторяющийся код javascript**

**} while (условие) ;**

В данном случае цикл выполниться хотя-бы один раз, а заием следуя условию while либо продолжиться, либо прерветься.

**ФУНКЦИИ**

**function printToday()** {

var today = new Date();

document.write(today.toDateString());

}

Вызов функции **function printToday()**

Так же функции **можно передавать аргументы FixTax(12,2), FixTax(miniTax, quantity)**

Если необходимо получить значение функции тогда необходимо использовать **return,** также функция может вернуть **только одно значение** переменной и если необходимо что бы функция возвращала более одного значения, тогда необходимо использовать **массивы**

**var numQuant** = prompt("Enter quantity of items",""); //создание переменной в которой мы указываем количество элементов массива

**function array(quantity)** {

// создаем функцию которая повторяем запрос элементов массива "numQuant" количество раз

var check = alert("Вам необходимо ввести " + quantity + " элементов массива");

var newArr = []; //объявляем пустой массив

for (var i = 1; i <= quantity; i++) {

var x = prompt("enter somthing please","");

newArr.push(x);

}

return newArr; //возвращает значение переменной(в нашем случае массива) которое мы можем использовать дальше.

}

**document.write(** array(numQuant) **);** //Вызов функции с передачей ей аргумента

**Выбор селекторов**

JS:

**docurnent.getElernentByid('banner')**; //поиск елемента по ID (выберет первый селектор)

**document.getElementsByClassName('author')**; //поиск елемента по CLASS (выберет все селекторы)

**document.querySelectorAll('span.author');** //выберет все элементы по тегу SPAN к которым применен класс **author**

**JQ:**

**$ ('селектор')**

**$ ( '#banner' ), $ ( '#navBar а' ), $ ( 'body > р'), $ ( ' irng [alt] ' )**

**$ ('а [href^="mail to:as@as.as”]')** //и т.д. $ заканчивается на…,\* включает название.

**$(' .striped tr:even')** //even – четные, odd – нечетные.

**$ ( 'р: first') ; $('p:last'); $ ('а:not ( [href^ ="http: / /]) ')**

**$('а:соntаins(Щелкни по мне!)')** //Фильтр :contains () находит элементы, содержащие конкретный текст.

**$('div:hidden')** // :hidden находит скрытые элементы, к которым относятся элементы, обладающие свойством CSS display, установленным на значение none

**$ (":text")[0].focus (); // выберет первый элемент на странице с типом text**

**Связывание функций**

**$('#popUp').width(300).height(300);**

**Или разбиваем для удобства чтения:**

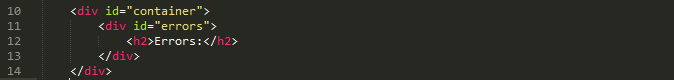
**$('#popUp').width(300)**

**.height(300)**

**. text('Сообщение')**

**.fadein(1000) ;**

**Добавление содержимого на страницу**

****

Функция **.html()**

**alert($('#errors').html()); -** этот код создает окно оповещения с текстом **"<h2>Errors:</h2>" соответственно**

**.html()** считывает содержимое тега с **ID=”errors”** и выводит его содержимое, если в скобках указать параметр, то функция заменит им содержимое тега**.**

**$('#errors').html('<p>**В данной форме четыре ошибки**</р>');** - Соответственно заменит **"<h2>Errors:</h2>"** на **'<p>**В данной форме четыре ошибки**</р>'.**

ПРИМЕЧАНИЕ

Если вы используете функцию ***html()*** или ***text()*** для извлечения НТМL-кода или текста из выборки jQuery, содержащей множество элементов, то НТМL-код или текст будет извлечен только из первого элемента выборки. Например, если вы примените код ***var divContents= $('div').html()***; к странице, на которой присутствуют 10 элементов ***div***, то в переменной ***divContents*** будет храниться НТМL-код только из первого элемента ***div*** на странице.

Тем не менее, при использовании функций ***html()*** или ***text()*** для вставки НТМL-кода или текста в выборку jQuery, все выбранные элементы окажутся затронуты этой вставкой.

Например, код ***$('div').html(‘<р>Привет</р>');*** заменит НТМL-код во всех элементах ***div*** на странице одним абзацем и словом «Привет».

Функция ***.text()***

похожа на работу функции ***.htrnl()***, но не принимает НТМL-элементы. Она полезна, если вам необходимо заменить текст, заключенный в элемент. Например, в коде в начале данного раздела вы видите элемент ***h2*** с заключенным в него текстом ***Errors:***. Допустим, после запуска программы для проверки формы на наличие ошибок вы захотели заменить текст ***Errors:*** на текст **“**ни одной ошибки не найдено**”**, в этом случае вы можете использовать следующий код:

***$('#errors h2').text(***'Hи одной ошибки не найдено***');***

*Функция* ***.append()***

добавляет НТМL-код в качестве последнего дочернего элемента выбранного элемента. Допустим, вы выбрали элемент ***div,*** но вместо замены его содержимого вы просто хотите добавить фрагмент НТМL-кода перед закрывающим тегом ***</div>.***

***$('#errors').append('<p>****B данной форме четыре ошибки****</р>'); -*** *добавит* ***<p></p>*** *с текстом перед закрвающим* ***div с ID=”errors”*** *не удаляя**тег* ***h2***

*Функция* ***.prepend() –*** *тоже что и* ***.append()*** *только добавляет в начало*

*Функция* ***.after() и before() –*** добавят элементы после или до элемента с ***ID=”errors”***

***$('#errors').after('<div id="newMessage"><p>****New Message****</p></div>');***

***$('#errors').before('<div id="newMessage"><p>****New Message****</p></div>');***

Способ **преобразования в DOM $(variable)** – если переменная variable содержит html код, он представляет собой просто набор символов(<div><h1>HEADER</h1></div>). Если мы к переменной variable применим функцию **$(variable)** то ее содержимое превратиться в DOM элемент и мы сможем применять к переменной функции JQuerry

var **$insertHTML** = **$(variable) //** $ в имени объявляемой переменной необязательный символ, говорящий что это oбъект и к нему применимыфункции :

$insertHTML.find(‘h1’).text(‘NewHeader’);

**Замена и удаление выборок**

**$('#popup').remove();** - удалит элемент с **ID=” popup”** и все что находится в нем.

**$('#product101').replaceWith('<p>**Дoбaвлeнo в корзину**</р>') -** заменит элемент с **ID=”#product101”** и все что находится в нем на **<p>**Дoбaвлeнo в корзину**</р>**.

**clone() -** позволяет копировать выбранный элемент.

**detach()** – удаляет содержимое со страницы но оставляет внутри переменной:

var $currentItem = $(this); // назначает переменной $currentItem значение текужего элемента

$currentItem.detach(); // удаляет элемент со страницы, но оставляет переменную в памяти браузера

$(‘’#newContainer).append($currentItem); // вставляет переменную в выбранное место

**Установка и чтение атрибутов элемента**

Функция ***addClass()*** добавляет элементу конкретный класс.

**$("a[href^='http:']").addClass('externalLink');** добавит ко всем ссылкам на странице класс **externalLink.**

------------------------------------------------------------------------------------------------------------------------------

Функция ***removeClass()*** противоположна функции **addClass().** Она удаляет указанный класс из выбранных элементов.

**$("a[href^='http:']").removeClass('externalLink'**); удалит класс **externalLink** из всех ссылок

**$("a[href^='http:']").addClass('externalLink').removeClass('simple');** добавит и удалит класс для всех ссылок

Функция **toggleClass() -** либо добавляет, либо удаляет класс

**$('#errors').click(function() { $("a[href^='#']").toggleClass('simple'); } );**

**Чтение и изменение свойств CSS**

Функция **сss()**

Вы можете использовать **функцию сss()** тремя способами; находить текущее значение свойства CSS определенного элемента, устанавливать свойство CSS определенного элемента или настраивать несколько свойств CSS одновременно.

**var bgColor = $('#main').css('background-color');** вернет значение цвета элемента с **ID=”main”**

**$('body').css('font-size', '200%')**; установит шрифт в 200%

**$( 'р.highlight').css('border', '1px solid blасk');**

**var baseFont = $('body').css('font-size');** //берет значение размера шрифта, например 16px

**baseFont = parseInt(baseFont);** // преобразует значение в число отсекая **“px”**

**$('body').css('font-size', baseFont \* 2);** // умножаем 16 на 2

**$('#highlightedDiv').css({ 'background-color’ : '#FFOOOO',**

**border: '2рх solid #FE0037' } ) ;**

Свойства можно не заключать в кавычки если в их именах нет дефиса или писать не **'font-size',** а **fontSize** и т.д.

Значение свойств если они без приставок **px, em** итд. писать также без кавычек

**{opacity : .8, color: '#DDD', boxSizing: 'border-box', 'font-size' : '2em';}**

**Чтение, установка и удаление атрибутов HTML**

Функции **attr () и removeAttr ()**

**аttr()** позволяет читать указанный атрибут НТМL­элемента.

**var imageFile = $('#banner img').attr('src');** вернет значение атрибута **src**

**$('#banner img').attr('src', 'images/newimage.png');** второе значение указываем если хотим установить атрибут

**$('body').removeAttr('bgColor');** удалит свойство **bgColor**

**Работа с каждым элементом выборки**

Функция .each()

Выполняет функцию для каждого элемента набора. Это означает, что она выполняется каждый раз, для каждого совпавшего элемента.

**var i = 1;**

**$('div').each(function() {**

**alert("We've just found the " + i + " div");**

**i++;**

**});**

Выведет окно с уведомлением столько раз, сколько **div** элементов она найдет

**this и $(this)**

**this** указывает на текущий элемент выборки



Т.е. говорит о том, что работа будет проводится с текущим элементом выборки. В данном случае мы клонируем ссылку со всем её содержимым и для того что бы вставить в другое место.

**$(this).find('li.target'.css("border", "3px double red"); find ищет среди всех потомков**

**$(this)**.**children('li.target').css("border", "3px double red"); children ищет только дочерние элементы**

$('li.target',this); короткая запись для find

**$(this).next(‘p’);**возьмет следующий элемент **‘p’** стоящий за **$(this)** на том же уровне DOM

**$(this).prev(‘p’);**возьмет предыдущий **‘p’** стоящий за **$(this)** на том же уровне DOM

[**nextAll(‘p’)**](http://jquery-docs.ru/Traversing/nextAll)**;** [**prevAll(‘p’)**](http://jquery-docs.ru/Traversing/prevAll)**;** тоже что и выше только выберет все элементы того же уровня

**ДЕЙСТВИЕ / РЕАКЦИЯ**

**События мыши**

1. **click.** Это событие запускается, когда вы нажимаете и отпускаете кнопку мыши.
2. **dblclick.** Данное событие запускается, когда вы дважды нажимае те и отпускаете кнопку мыши.
3. **mousedown**. Это первая часть щелчка - момент, когда вы нажимаете кнопку мыши, но не отпускаете ее.
4. **mouseup**. Событие mouseup - это вторая часть щелчка - момент, когда вы отпускаете кнопку мыши.
5. **mouseover**. Запускается, когда вы проводите указателем мыши по элементу страницы.
6. **mouseout**. Запускается, когда вы смещаете курсор с элемента.
7. **mousemove**. Это событие запускается, когда вы передвигаете курсор, то есть оно происходит постоянно.

**События документа / окна**

1. **load.** Запускается, когда браузер заканчивает загрузку всех файлов веб-страницы: самого НТМL-файла, связанных изображений, Flаsh-фильмов, внешних файлов CSS и JavaScript.
2. **resize.** Запускается, когда вы изменяете размер окна браузера, нажимая кнопку разворачивания окна, либо изменяете ширину окна браузера, перетаскивая его границу.
3. **scroll.** Запускается, когда вы используете полосу прокрутки или клавиатуру (клавиши PageUp, PageDown, Ноте, End и т. д.) либо прокручиваете веб-страницу с помощью колеса мыши. Если на странице нет полос прокрутки, событие **scroll** не запускается.
4. **unload.** Запускается, когда вы щелкаете по ссылке для перехода на другую страницу, закрываете вкладку в окне браузера или само окно. Это событие можно сравнить с “***последним вздохом***” программы JavaScript, и оно дает вам возможность совершить последнее действие, прежде чем посетитель покинет страницу.

**События форм**

1. **submit.** Запускается, когда посетитель отправляет данные формы с помощью щелчка по кнопке Submit (Отправить) или нажатия клавиши Enter, когда курсор находится в текстовом поле.
2. **reset.** Пусть и нечасто, но кнопка Reset (Отменить) может пригодиться вам, чтобы отменить любые изменения, сделанные в форме. Это событие возвращает страницу в состояние, в котором она находилась на тот момент, когда она была загружена.
3. **change**. Многие элементы формы запускают это событие при изменении их статуса, например, когда кто-нибудь нажимает переключатель или выбирает пункт из выпадающего меню.
4. **focus.** Запускается, когда вы переходите к элементу формы, щелкая по нему мышью или Таb
5. **blur**. Противоположно событию focus. Запускается, когда элемент формы выводится из фокуса

**События клавиатуры**

1. **keypress.** Это событие запускается в тот момент, когда вы нажимаете клавишу. Причем, чтобы оно сработало, вам необязательно эту клавишу отпускать. Событие keypress запускается снова и снова до тех пор, пока клавиша удерживается нажатой, поэтому данное событие позволяет проверить, удерживает ли посетитель клавишу нажатой.
2. **keydown**. Подобно событию keypress, оно запускается при нажатии клавиши. На самом деле оно запускается непосредственно перед запуском события keypress. В браузере Opera событие keydown запускается только один раз; в других программах работает так же, как и событие keypress, - запускается снова и снова, пока клавиша нажата.
3. **keyup.** Запускается, когда вы отпускаете клавишу.

**Расширенные события JQuerry**

**События наведения и смещения указателя мыши**

**$('#селектор').hоvеr(функция1(наведение), функция2(покидание));** аналог **hover** в CSS3

**$('#menu').hover(function() {**

**$('#submenu').show();**

**} , function () {**

**$('#submenu').hide();**

**}); //окончание события hover**

Или с обычными функциями:

**function showSubmenu() {**

**$('#submenu') .show();**

**}**

**function hideSubmenu() {**

**$('#submenu') .hide();**

**}**

**$('#menu').hover(showSubmenu, hideSubmenu);**

**Объект события**

В момент запуска события браузер записывает информацию о нем и сохраняет его в так называемом ***объекте события***. Объект события содержит данные, собранные в момент, когда произошло событие, такие, как координаты указателя мыши по горизонтали и вертикали, элемент, который инициировал событие, была ли нажата при запуске события клавиша Shift.

**$(document).click(function(event) {**

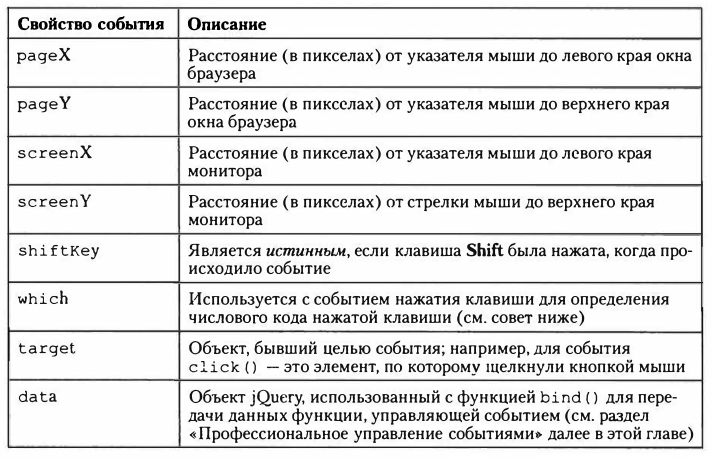
**var PosX = event.pageX,**

**PosY = event.pageY;**

**alert('X: '+PosX+ ' Y:'+PosY)**

**})**

Функция привязанная к событию **click** сохраняет **«объект события»** в переменной **event(**любое имя), а точечный синтаксис **event.pageY** и **event.pageX** возвращает координаты.



**СОВЕТ:**

Если вы попытаетесь получить доступ к свойству **which** объекта события, соответствующего событию **keypress()**, вы получите числовой код нажатой клавиши. Если вы хотите узнать, какая конкретно клавиша была нажата **(а, К, 9 и т. д.)**, вы должны преобразовать значение свойства **which** с помощью метода JavaScript, который превращает числовой код клавиши в конкретную букву или символ клавиатуры:

**String.fromCharCode(event.which)**

**$(document).keypress(function(event) {**

**var key = String.fromCharCode(event.which);**

**alert('Key is: '+key);**

**})**

**Отмена обычного поведения событий**

Функция **preventDefault()** позволяет отменить нормальное поведение браузера. Она является частью объекта события, поэтому доступ к ней осуществляется в рамках функции, управляющей событием.

|  |  |
| --- | --- |
| **$('#menu') .click(function(evt) {**  // "Умный" JavaScript-кoд вставляется сюда  **evt.preventDefault();** // не переходи по ссылке  **}) ;** | **$('#menu') .click(function(evt) {**  // "Умный" JavaScript-кoд вставляется сюда  **return false;** // не переходи по ссылке  **});** |

**Удаление события**

Иногда вам может понадобиться удалить событие, которое вы ранее присвоили элементу. jQuеrу-функция **off()** позволяет это сделать.

**$(document).keypress(function(event) {**

**var key = String.fromCharCode(event.which);**

**alert('You’ve just pressed: “'+ key +' ” button');**

**});**

**$('html').dblclick(function() {**

**$(document).off('keypress');**

**});**

Если нужно удалитьвсесобытия выбранного элемента, тогда мы не указываем в скобках параметр **$(‘a’).off()**

**Остановка события**

Можно представить ситуацию, когда к примеру у нас есть div с формой(к примеру) внутри формы есть кнопка отправить, мы присваиваем ей событие mouseover для того что бы менялся её цвет когда над ней находится курсор мыши(hover эффект), но к форме у нас тоже стоит привязка к mouseover для появления подсказки для чего эта форма нужна, так вот если мы наведем на кнопку, её цвет изменится, но и появиться подсказка от формы т.к. кнопка дочерний элемент для формы и соответственно сработает событие и для подсказки и цвета кнопки. Для предотвращения этого эффекта нужно использовать функцию **stopPropagation() и объект события** это отменит передачу события всем предкам элемента.

**$('form').mouseover(function() {**

**$(this).**//создание подсказки

**});**

**$('#button').mouseover(function(event) {**

**$(this). css('background-color','purple');**

event.stopPropagation();

**});**

**Профессиональное управление событиями**

**Литерал объекта (или объектная константа) –** это список пар свойство/значение. После каждого имени свойства вы добавляете двоеточие **(:),** за которым следует значение; пары **свойство/значение** разделяются запятыми**;** все это заключено в фигурные скобки **- { }**

**{ 'background-color' : '#EEDDFF', 'border' : '2рх solid #FE0037'}**

**{newMessage : 'My congratulations you’ve just won a prise!'}**

**Функции on() -**

**$('#селектор').оn(**'событие’**,** селектор, данные, имя функции**);** красным выделены обязательные

**Первый аргумент -** это строка, содержащая имя события **(mouseover** к примеру**)**

**Второй аргумент -** должен представлять собой допустимый селектор: **.callout** или **#alarm**

**Третий аргумент -** это данные, которые вы желаете передать функции (в форме литерала объекта либо переменной, содержащей литерал объекта) к примеру: **var linkVar = {message: 'Привет от ссылки'};**

**Четвертый аргумент -** другая функция (анонимная функция или обычная)

**$('селектор').on('событиеМыши', имяФункции);**

Функционально данный код идентичен следующему:

**$('селектор').сliсk(имяфункции);**

**var linkVar = {message: 'Привет от ссылки'};**

**var pVar = {mеssаgе:'Привет от абзаца'};**

**function showMessage(evt) {**

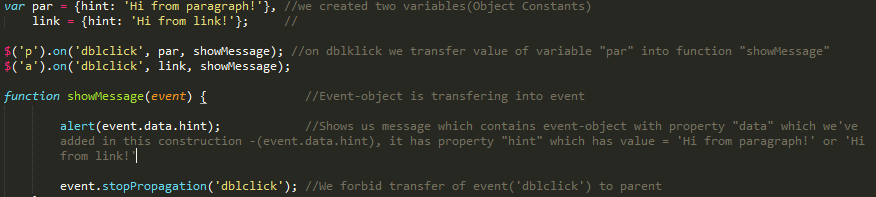
**alert(evt.data.message);**

**}**

**$('а').on('mouseover',linkVar,showMessage);**

**$(' р ').on('click',pVar,showMessage);**

этот код создает две переменные**: linkVar** в первой и **pVar** во второй строке. Каждая переменная содержит **объектную константу** с одним и тем же именем свойства **(message),** но с различным текстом. Функция **showMessage()** принимает **объект события** и сохраняет его в переменной **evt.** Эта функция запускает команду **alert(),** отображая свойство **message** (которое само является свойством свойства **data** *объекта события***).** Помните, что **message** - это имя свойства, определенного в объектной константе.



Мы создали две переменные **par** и **link** с значением **объектной константы**(как в передаче свойств css )

Далее создали два действия по двойному клику + передаем значение переменной par или **link** событию **event** функции **showMessage**.

Далее функция выводит сообщение со значением **объекта события(event)** к которому мы применили свойство **data**(см. таблицу выше) и конкретно значения свойства **hint**

**Другие способы использования функции on()**

|  |  |
| --- | --- |
| Можно использовать несколько событий для **on()**  **$(docurnent).on('click keypress', function() {**  **$('#lightbox').hide();**  **}); // завершение функции on** | Использование **объектной константы**  **$('#theElernent').on({**  **'click' : function() {**  // какое-либо интересное действие  **},** // окончание функции click  **'mouseover' : function() {**  // какое-либо интересное действие  **};** // окончание mouseover  **});** // окончание функции on |

**Делегирование событий с помощью функции on()**

**$('#селектор').on('событие мыши', селектор, данные, имяфункции);**

**Второй аргумент** может быть любым действительным селектором jQuery, например, селектором идентификатора, класса, элемента и т. д. Передача селектора функции **on()** может значительно повлиять на ее работу.

Это может потребоваться в случае с добавлением данных динамически. Если к примеру у нас есть список **ul** и подпункты li отсутствуют при первоначальной загрузке страницы то запись вида:

**$('ul li').on('click', function() {**

**//some code**

**});**

не сработает, т.к. элементов **‘li’** еще нет на странице и придется делегировать событие(заставить **'ul'** следить за тем когда появится дочерний **'li'**)

**$('ul').on('click', 'li', function() {**

**//some code**

**});**

в данном случае код сработает. Соответственно все действия будут применяться не к тегу **'ul'**, а к тегу **'li'**

**АНИМАЦИЯ И ЭФФЕКТЫ**

**ОТОБРАЖЕНИЕ И СОКРЫТИЕ**

**show()** показывает ранее скрытый элемент. Она не работает, когда элемент уже виден на странице. Если вы не устанавливаете значение скорости, элемент появляется моментально. Однако если скорость задана, например, ***show(1000****)* , то элемент анимируется от верхнего левого к нижнему левому углу.

**hide()** скрывает видимый элемент.

**toggle ()** переключает текущее состояние элемента. Если элемент в данный момент отображен, то функция *toggle()* скрывает его. Если элемент скрыт, она заставляет его появиться.

**Постепенное появление и исчезновение элементов**

**fadeIn()** показывает скрытый элемент. Сначала на странице появляется пространство для элемента (это значит, что остальные элементы страницы сдвинутся с места), затем элемент постепенно становится видимым.

**fadeOut() -** заставляет элемент исчезать, делая его прозрачным. Она не работает, если элемент уже скрыт.

**fadeToggle() –** переключает режимы **fadeIn()** и **fadeOut().**

**fadeTo(‘100’, .75) –** изменяет уровень **opacity** в отличии от предыдущих которые выставляют **display: none;** значение **времени** не обязательное, а уровень **opacity** указывать нужно. Более того если после применения **opacity** элемент скрыть какой-либо другой функией (**fadeOut()** к примеру, которая выставит для элемента **display: none;**) a потом вернуть её обратно, то степень непрозрачности не сброситься, а будет такой которую мы выставили до скрытия.

**Скользящие элементы**

**slideDown()** заставляет скрытый элемент появиться в поле зрения. Сначала появляется верхняя часть, и, по мере появления остальной части, то, что находится под элементом, сдвигается вниз.

**slideUp()** - удаляет элемент из виду, скрывая его нижнюю часть и двигая все, что находится ниже, вверх по мере того, как элемент исчезает.

**slideToggle() –** переключает **slideDown()** и **slideUp()**

**Тип записей для функций анимации**

**slideDown(**time**, ‘**typeOfAnymation**’,** function**(**выполниться после выполнения анимации**){** some code **} )**

**img. slideDown (1500, ‘linear’, functuion () {**

**caption. slideDown(4000, ‘linear’);**

**}**

**);**

**Анимация**

**animate (),** вы можете анимировать любое свойство CSS, принимающее числовое значение ( пикселы, em или проценты).

**$('#message').animate(**

{

left: '650рх',

opacity: .5, // если в значении только цифра, то пишем без кавычек.

fontSize: '24рх' // пишем кемелкейсом или в кавычках

} **,**

1500**,**  //скорость воспроизведения анимации

'linear'**,**  //тип кривой Безье

function() { //выполниться после анимации

Some code

}

**)** ;

Вы также можете задать свойство относительно его текущего значения, используя операции **+=** или **-=** как часть настроек анимации.

left:'+=50px'

Выполнение действия после завершения эффекта

Все в примере выше или связываем **animate ({some code}).fadeOut().animate({some code})**

**ВАЖНО**

Из-за особенности работы библиотеки с анимацией возможен эффект многократного повторения, представим что по эффекту **hover** у нас открывается полу спрятанная панель, так вот, если провести быстро над ней мышкой(несколько раз) в момент открытия, то анимация повториться столько раз сколько мы провели над ней мышкой и мы будем наблюдать как панель *многократно выезжает/заезжает*. Для устранения эффекта нужно использовать встроенную функцию **stop()** которая останавливает все анимации кроме той что необходимо выполнить в данный момент

**$('#dashboard').hover(**

**function() {**

**$(this).stop().animate({** left: 0 **}),**

**function() {**

**$(this).stop().animate({** left: ‘-100px’ **});**

Анимация при загрузке страницы выполнится сразу, поэтому если нам нужно что бы анимация запускалась по щелчку или т.п. тогда мы должны добавить класс с анимацией по щелчку или добавить **anirnation-play-state: paused;** в **CSS** и при щелчке изменять значение на **running**

**$('.progressBar').css('animation-play-state', 'running');**

**РАСПРОСТРАНЕННЫЕ ЗАДАЧИ, РЕШАЕМЫЕ С ПОМОЩЬЮ JQUERY**

**Смена изображений**

Меняем **src:**

**$('#photo').attr('src', 'images/newimage.jpg');**

**Если вы поменяете атрибут src изображения, используя язык JavaScript, то путь к иллюстрации будет основан на местоположении страницы, а не JavaScript-кoдa.**

Меняем атрибуты исходного изображения на атрибуты заменяемого

Image() – это специальный объект для работы с изображениями, он имеет свои атрибуты:

**src –** путь к файлу

**alt –** описание картинки

**width, height –** значение атрибутов высоты и ширины

**naturalHeight, naturalWidth –** возвращает исходный размер картинки

var newPhoto = new Image(); //присваиваем переменной значение обекта Image()

newPhoto.src = 'images/newimage.jpg'; //добавсяем в него ключ src со значением нового src

var photo = $('#photo'); //находим исходное фото

photo.attr('src',newPhoto.src); //меняем атрибут src на новый из объекта

photo.attr('width',newPhoto.width); //заменяем значение атрибута width на ширину объекта Image()

photo.attr('height',newPhoto.height); //заменяем значение атрибута height на ширину объекта Image()

или короче

$('#photo') .attr({

src: newPhoto.src,

width: newPhoto.width,

height: newPhoto.height

} );

**Смена изображений с помощью jQuery**

**replaceWith()** – заменит тэг целиком на указанный нами.

<img scr=”johnFace.jpg” alt=”face” id=”smbFace”>

$(‘#smbFace’).**replaceWith**(‘<img scr=”otherFace.jpg” alt=”face” id=”smbFace”>’)

Можно одновременно сохранить значение заменяемого изображения для дальнейшего использования:

var **oldImg** = $('#img2').replaceWith('<img src="1.png" alt="alt" width="100" height="100" id="img2">');

$('#img2').after(**oldImg**); //вставит старое изображение после замененного

replaceWith() – возвращает код который она впоследствии заменяет.

**Предварительная загрузка изображений**

**Нужно для загрузки изображений в кэш браузера для ускорения отображения**

**var newPhoto = new Irnage();**

**newPhoto.src = 'images/newirnage.jpg';**

Если изображений много, то используем массив:

**var preloadirnages = ['images/roll.png',**

**'images/flower.png',**

**'images/cat.jpg'**

**];**

**for (var i=O; I < preloadimages.length; i++) {**

**new Image().src = preloadimages[i]; //**создаем новый объект изображения и устанавливаем его атрибут **src**

**}**

**Или :**

**var preloadimages = ['images/roll.png',**

**'images/flower.png',**

**'images/cat.jpg'];**

**for (var i=0; i < preloadimages.length; i++) {**

**$('<img>').attr('src',preloadimages[i]);**

**}**

**$('<img>') - <>** указывают на то, что мы **создаем новый элемент img,** без кавычек будет просто **выборка**

**Смена изображений с hover**

**var newPhoto = new Image();**

**newPhoto.src = '2.png';**

**var oldPhoto = $('img').attr('src'),**

**oldHeader = $('h2').text();**

**$('#gallery').hover(**

**function() { $('img',this).attr('src', newPhoto.src);**

**$('h2',this).text('New Item'); },**

**function() { $('img',this).attr('src', oldPhoto);**

**$('h2',this).text(oldHeader); });**

**РАЗБОР “ПОЛЕТОВ”**

****

**25** мы создаем выборку с пустой функцией которая проходит по всем **img:** выборка находит первый тэг **img, 26** мы создаем переменную **originalSRC** которая содержит в себе атрибут **src** текущей картинки, **27** далее мы создаём пустой объект изображения **Image()** и передаем его значение(***пока пустое***) переменной **preloadImage(***для пред загрузки новой картинки в последствии***)**. Далее создаем переменную **imgExt** которая содержит в себе условие (регулярное выражение) для дальнейшего изменения текущего пути к картинке с помощью функции **replace**. **29** атрибуту **src(***пока пустого***)** нашего объекта изображения хранящегося в **preloadImage** мы передаем значение переменной **originalSRC**(путь к текущей картинке) и с помощью **replace** изменяем его(добавляем перед точкой **\_h**). **31** Мы вешаем функцию **hover** в которой при наведении мы меняем атрибут **src** на измененный нами (с **orange.png** на **orange\_h.png**) и при отводе мыши возвращаем старый. В целом данная программа, находит по одному img. Вычисляет текущий **src** картинки, создает объект **Image()** в который мызаписываем измененный функцией **replace** текущийпуть, навешивает на текущую картинку событие **hover** изменяющее атрибут **src** текущей картинку, передаёт текущее измененное значение пути **preloadImage.src** в функцию наведения мыши и передаёт старый путь **originalSRC** в функцию отведения мыши. После переходит к следующей картинке и делает тоже самое пока картинки не закончатся. Получается, что у нас на каждой картинке навешена отдельная функция hover, все три функции содержат одинаковые переменные но с разными значениями внутри каждой функции.

**УПРАВЛЕНИЕ ССЫЛКАМИ**

$('a[href^="http://"]').each(function() {

var linkWay = $(this).attr('href');

linkWay = linkWay**.replace**('http://', 'здесь на что заменить');

$(this).append(' ('+linkWay+') ');

})

Данный код найдет ссылки содержащие атрибут href^="http://" далее возьмет полное значение атрибута href, **присвоит переменной** **linkWay** **и удалит** 'http://**'** с помощью функции **replace()** .

.replace(‘*искомый текст*’, ‘*тот что мы вставим*’) – если второй параметр не указан, как в примере выше, тогда функция просто удалит указанное содержимое содержащееся в первом параметре.

Например **replace**(‘*http://*’,*’https://*’) удалит **http://** и заменит его на **https://**

**Запрет перехода по ссылке**

**Вариант 1:**

$('#showForm').click(function() {

$('#loginForm') .fadein('slow');

**return false;**

} ) ;

**#showForm** ссылка на страницу авторизации для тех у кого отключен **JS**. При клике на этой ссылке

мы отображаем форму входа и для того что бы тех у кого **JS** включен не перебрасывало на эту страницу и они могли использовать всплывающую форму мы запрещаем переход с помощью

**return false**, которая **обязательно должна стоять в конце функции** иначе она прервётся как только парсер дойдет до этой инструкции.

**Вариант 2:**

$('#showForm').click(function(event) {

$('#loginForm') .fadein('slow');

event**.preventDefault()**

} ) ;

event**.preventDefault()** предотвращает стандартное поведение элемента.

**Открытие внешних ссылок в новом окне**

**$**('a[href^="http://"]')**.attr('***target***', '***\_Ыank***');**

Просто меняем атрибут ссылки **target** на значение **\_blank**

**$**('a[href^="http://"]', 'a[href^="https://"]')**.attr('***target***', '***\_Ыank***');**

То же но с дополнительной выборкой **https**

К примеру у нас есть ссылки которые мы хотим открывать в той же странице (к примеру наши внутренние ссылки все абсолютные типа mcomp.dp.ua/akcii.html и т.д. )

var myURL = **location.protocol** + '//' + **location.hostname**

$ ('а [ href^="http://"]', а[href^="https://"]').**not**(**'**[href^ =" **'**+myURL+**'**"]**'**).attr('target', '\_Ыank');

**location.protocol –** вычисляет протокол текущей страницы к примеру **http:**

**location.hostname** – определяет доменное имя сайта к примеру **mcomp.dp.ua**

Мы создаем переменную myURL и помещаем в неё протокол **http:** плюс **//** плюс **mcomp.dp.ua**, соответственно значение переменной становиться [**http://mcomp.dp.ua**](http://mcomp.dp.ua)**.** Далее функция not() исключает из выборки те ссылки которые содержат атрибут **href текущего сайта** и добавляет к остальным ссылка содержащим **http:** и **https** атрибут для открытия в новой вкладке**.**

**Создание новых окон**

open(***URL***, ***name***, ***properties***)

var newWin = open('***http://www.google.com/***', '***theWin***', '***height = 200***, ***width = 200***');

Данный код открывает новое окно и сохраняет ссылку на него в переменной **newWin**

***URL*** – адрес сайта который мы хотим открыть

***Name*** – имя которое мы присваиваем окну

***Properties*** – свойства открытого окна



В примере мы использовали одно и то же имя **mySite** при открытии нового окна. Исходя из этого обе ссылки будут открыты в одном и том же окне с именем **mySite**. Если дать разные имена, то скрипт создаст отдельные окна

**Свойства окна:**

Свойства окна можно передавать двумя способами

**1:** open('***http://www.google.com/***', '***theWin***', '***height = 200***, ***width = 200***');

свойства отмечены красным цветом.

**2:** var windowProperties = '***height = 200***, ***width = 200***';

open('***http://www.google.com/***', '***theWin***', **windowProperties**);

свойства отмечены красным цветом.

**heiqht** – высота (пикселы,px)

**width** – ширина (пикселы,px)

**left** – позиция от левого края монитора (пикселы,px)

**top** - позиция от верхнего края монитора (пикселы,px)

**scrollbars** – наличие полосы прокрутки (yes, no)

**status** - контролирует отображение строки состояния в нижней части окна. (yes, no)

**toolbar** - определяет видимость панели инструментов, содержащей кнопки навигации, вкладки и другие элементы (yes, no)

**location** – строка для ввода адреса (yes, no)

**menuЬar** \_ относится к браузерам, имеющим меню в верхней части окна (yes, no)

**Использование ссылки на окно**

**close()** –закроет текущее окно. **windowName.close();** - закроет окно с именем **windowName,** можно использовать с событиями.

**blur()** - выводит окно «из фокуса». Это означает, что окно перемещается на задний план под все открытые окна.

**focus()** - противоположен методу **blur()** , помещает окно поверх всех других окон.

**moveBy()** - позволяет перемещать окно на заданное число пикселов вправо и вниз. Содержит два параметра, первый смещение в пикселях по **X**, второй по **Y:** **moveBy(100, 300)**. Перемещает окно **относительно текущей** позиции

**moveTo()**-позволяет перемещать окно в заданное место на экране. moveTo(0, 0) ; (левый верхний угол)

**resizeBy()** - изменяет ширину и высоту окна относительно текущего размера **resizeBy(100,200)**

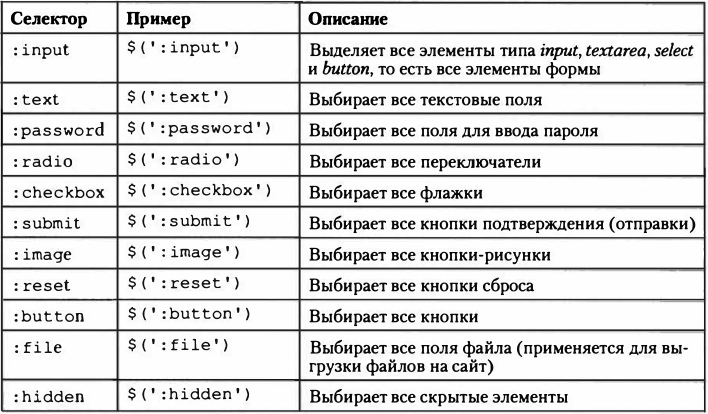
**resizeTo() -** изменяет размеры окна на заданные. **resizeTo(500, 800)**

**scrollBy()-** прокручивает документ внутри окна на заданное количество пикселов вправо и вниз **scrollBy(50,50)**

**scrollTo()** - прокручивает документ внутри окна на заданную позицию по отношению к левому и верхнему краям.

scrollTo(100, 200)если документ влезает целиком в окно, две последние функции недоступны

**УЛУЧШЕНИЕ ВЕБ-ФОРМ**

****

**:checked** выбирает все отмеченные или установленные элементы формы, то есть флажки и переключатели. **$**('input [name="shipping"]:checked')

**:selected** выбирает все отмеченные элементы **option** внутри списка или меню, позволяя вам узнать, какой выбор сделал посетитель (**элемент select**) **$**('#region :selected').val(); (не забываем пробел после **#region,** тк он родитель для **option** с селектором **:selected**) фильтр **:selected** может возвращать больше одного элемента.

**СЧИТЫВАНИЯ ЗНАЧЕНИЯ ФОРМ**

**vаl()** может как задавать, так и считывать значения элементов формы.

*var fieldValue = $('#email') .val(); //поместит значение формы в переменную*

*$('#email') .val(‘example@ui.ui’); //запишет значение в форму*

**Проверка активации кнопок и флажков**

**prop():** возвращает **true** или **false**

if ($( '#news').prop('checked')) {

// флажок установлен выполняем код. Иначе идем к else

} else {

// флажок сброшен. Выполняем код.

}

**События формы**

**submit()** – выполняется при нажатии кнопки **submit**(или клавиши **enter**) внутри формы. **Применяется только к формама**

$('signup') .submit(function()

If ($('#username').val() == ' ' ) { //если значение формы пустое, выведет

аlеrt('Введите имя в поле "Имя".'); // предупреждение и отменит отправку формы

**return false;**  //запрет отправки формы

}

}); // конец subrnit()

Или исполизуем event.preventDefault();

$('signup') .submit(function(event)

If ($('#username').val() == ' ' ) { //если значение формы пустое, выведет

аlеrt('Введите имя в поле "Имя".'); // предупреждение и отменит отправку формы

**event.preventDefault();** //запрет отправки формы

}

}); // конец subrnit()

**Активное состояние**

Состояние при котором курсор находится в поле и мы можем вносить информацию с клавы.

$('#usernarne').focus(function() {

var field = $(this);

if (field.val() == field.prop('defaultValue')) {

field.val('');

}

} ) ; //если поле имеет «значение по умолчанию», то мы при **focus** его стираем точнее устанавливаем в ‘’.

**defaultValue –** это значениеформы при первоначальной загрузке страницы

Код выше - это аналог placeholder

**Неактивное состояние**

blur() – запускается когда мы убираем фокус из поля ввода.

Можно использовать для моментальной проверки корректности ввода информации в поле.

$('#quantity').blur(function() {

var fieldValue=$(this).val();

if (isNaN(fieldvalue)) {

аlеrt('Пожалуйста, введите число');

}

} ) ; // проверит содержится ли в поле число и если нет выведет сообщение об ошибке

Если у нас несколько числовых полей, тогда можно назначить полям например класс .numberValue и применить выборку ко всем элементам с данным классом.

**Щелчок мышью**

$ (':radio').click( function () {

//функция будет применена к каждому переключателю при щелчке по нему

} ) ;

**Смена**

**change() –** срабатывает при изменении элемента, прекрасно применимы к выпадающим спискам.

$('#country').change(function() {

if ($(this).vаl(}=='Выберите страну')

аlеrt('Пожалуйста, выберите страну из данного раскрывающегося списка.');

}

)};

**Усовершенствование форм**

**Фокусировка на первом элементе формы**

$(document) .ready(function() {

$('#username').**focus**();

} ) ;

Поставит курсор в поле с ID=”#username” сразу после загрузки страницы.

$(document).ready(function() {

$(':text:first').**focus**();

} ) ;

Поставит курсов в первое текстовое поле страницы сразу после её загрузки.

**Отключение/включение элементов формы**

Чтобы отключить элемент формы, задайте для атрибута **disaЫed** значение **true**. Например, для отключения всех полей ввода формы можно использовать следующий код:

$(' :input').prop('**disaЬled**', true);

К примеру если вы одинок, то **отключаем** ввод возраста вашей пары:

**$** ( '#single').click( function () {

**$**('#yourCoulpeAge').**prop**('disaЬled', **true**);

} ) ;

Для **включения**:

**$**('#yourCoulpeAge').**prop**('disaЬled', **false**); //обязательно true или false без кавычек

**JQUERY UI**

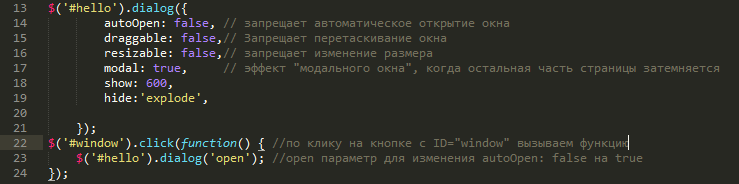
**ПРИМЕР ПОДКЛЮЧЕНИЯ:**

<link href="../\_css/jquery-ui.min.css" rel="stylesheet">

<script src="../\_js/jquery.min.js"></script>

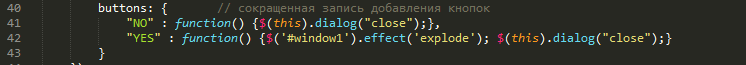
<script src="../\_js/jquery-ui.min.js"></script>

**dialog() –** вызов модального окна

****

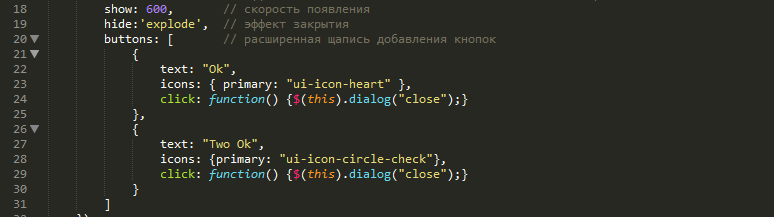
**Есть 2 способа записи, простои и расширенный**

**1-ый простой:**

****

Это обычный объект, в кавычках указывается текст кнопки, а после двоеточия функция, выполняемая при нажатии

**2-ый расширенный:**

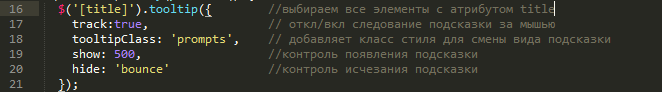
****

**Text** – значение текста кнопки**, icons** – стиль добавляемой иконки, встроенной в **JQuerry Ui,** и третий параметр **событие-функция**

**Всплывающие подсказки**

tooltip() – создан для организации всплывающих подсказок.

$('[**title**]').**tooltip**(); - этот селектор выберет все элементы содержащие атрибут title и организует всплывающие подсказки.



Если нам необходимо добавить в подсказку **HTML** код то это можно сделать 2-мя способами

**1:**

$('#element').tooltip({

**content**: '<img src = "me.png" alt = "Посмотрите на меня !"> '

} ) ; //покажет в всплывающей подсказке картинку

**2:**

Необходимо добавить на страницу скрипт(например перед закрывающим тегом) с ID в котором будет содержаться HTML код к примеру:

<script type="text/javascript" id=”prompt”>

<ul>

<li>item 1</li>

<li>item 2</li>

<li>item 3</li>

</ul>

</script>

И использовать его с подсказкой

$('#aim').tooltip({

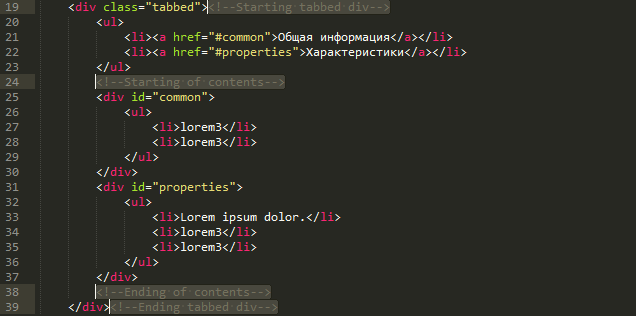
content: $('#prompt').html();

});

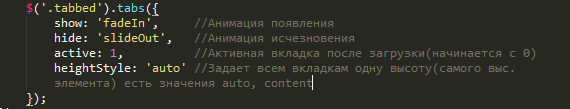
**ВАЖНО!!!**

Если у элемента к которому мы применяем метод tooltip() нет аттрибута title(можно пустой), то данный метод не покажет ничего!!!

**ПАНЕЛЬ С ВКЛАДКАМИ**

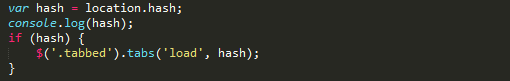
****

Создаем DIV с любым подходящим **class** или **id**. В него помещаем **неупорядоченный список** внутри элементов которого должны находится ссылку на **div** с **id** в которые мы помещаем содержимое необходимое для отображения в вкладках. Далее создаем **div**-ы с нужным содержимим и соответствующими **id**.

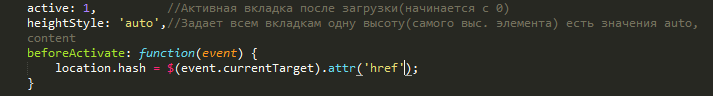


**Открытие вкладки по ссылке** (Только если не указан параметр active:)

Передаем кому либо ссылку, где после адреса указываем нужный **id** нужной вкладки site.com/phones/i35.html**#properties** и добавляем код который присваивает переменной **hash** значение адреса сайта - команда **location** к которой применен параметр **hash** отрезающий из адреса все что стоит до знака **#** и если значение переменной **hash** истинно(**true**), то ищет нашу таб панель и открывает её на вкладке указанной **$('.tabbed').tabs('load', hash)**; параметр **load** говорит открыть панель на ссылке с **id** в переменной **hash**

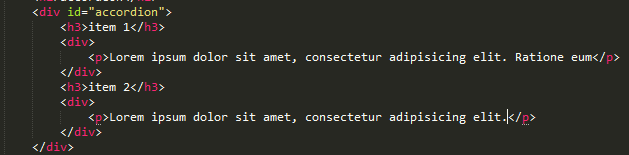


Код ниже выполниться до активации и добавит в конец адресной строки **id** текущей активной вкладки, для того что бы пользователь мог добавить закладку в браузер с открытой текущей вкладкой виджета

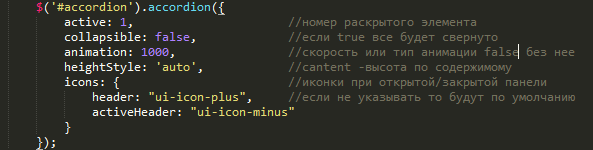
****

**Аккордеон**

accordion()

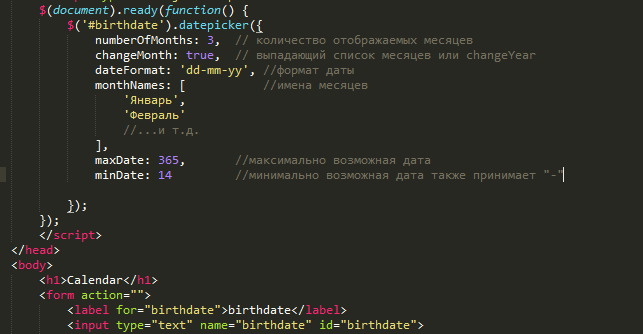


Вызов



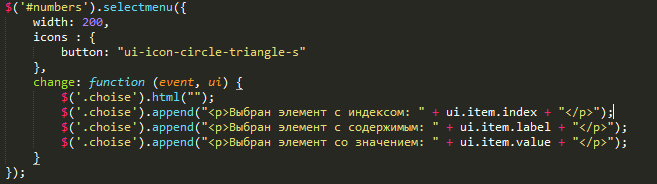
**КАЛЕНДАРЬ**

datepicker()



**Стилизация раскрывающихся списков**

Selectmenue()



**width** – ширина списка, **icons** – иконка списка, **change** – действие при смене значения списка

параметр change передает два события в функцию event – это обычное событие, а вот ui нас заинтересует:

**ui.item.index** – возвращает порядковый номер выбранного тега **option** (начинается с 0)

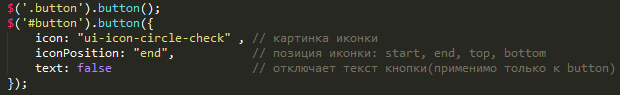
**ui.item.label** – возвращает текст внутри выбранного тэга **option**

**ui.item.value** – возвращает значение атрибута **value** выбранного тега **option**

Данная программа стилизует выпадающий список, задает ширину в 200 пикселов, потом добавляет значек и навешивает функцию при смене значения тега **select**. Точнее при выборе какого либо тега option программа ишет **div** с **id = chiose** удаляет всесь **html** код внутри него и вставляет порядковый номер выбранного элемента **option**, текст внутри него и его значение **value**.

**СТИЛИЗАЦИЯ КНОПОК**

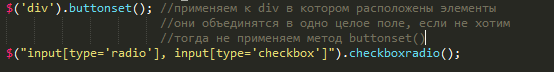
button()



Иконки применимы только к элементам **<button></button>** , а сам стиль можно применять ко всем элементам, включая кнопки форм.

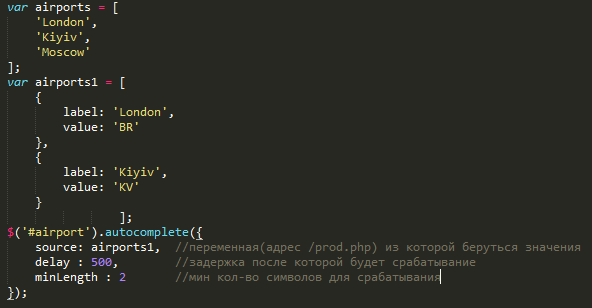
**СТИЛИЗАЦИЯ RADIO И CHECKBOX**

Checkboxradio() и buttonset()



**Автозаполнение**

autocomplete()

****

**2016-12-28_100047**

Есть три варианта передачи массива

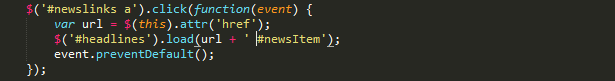
1 Создаеться простои массив как в переменной **airports** и подставляеться в параметр **source**

2 Создаеться массив с объектами airports1 с двумя значениями **label** и **value**, стоит помнить что выборка идет из значений параметра **label**, но в поле вставиться значение **value**

3 В параметр **source** мы вписываем адрес файла сервера который вернет нам необходимый массив

**AJAX**

Метод **load()**

****

В начале мы навешиваем функцию click на каждый элемент(ссылка а) списка #newLinks, далее мы создаем переменную url к которуу помещаем путь к странице с содержимым находящимся в атрибуте href ссылки , далее мы находим элемент с id headLines и с помощью метода load() заменяем его содержимое на содержимое по полученной ссылке и в конце предотвращаем переход по ней

ВАЖНО!

Метод load() загрузит всю страницу по ссылке и если нам нужна ее конкретная часть, то мы добавляем после ссылки id или class нужного элемента. В данном случае load(url + ‘ #newsItem’) **ОБЯЗАТЕЛЬНО УКАЗЫВАЕМ ПРОБЕЛ В КАВЫЧКАХ ПЕРЕД ЭЛЕМЕНТОМ**

**Методы qet() и post()**

$.get(url, data, callback);

$.post(url, data, callback);

**url -** это строка, содержащая путь к сер верному сценарию обработки данных

**data** - является либо строкой, либо литералом объекта JavaScript, содержащим данные, которые вы хотите послать на сервер

**callback** - является функцией обратного вызова, которая обрабатывает информацию, возвращаемую сервером

При выполнении функции **get ()** или **post ()** браузер посылает данные по указанному URL-aдpecy. Когда сервер возвращает ответные данные, браузер передает их функции обратного вызова, которая обрабатывает эту информацию и обновляет страницу.

$.post('rankMovie.php', 'rating=S');

Или

var data = {

rating: 5,

user: 'Вова'

}

$.post('rankMovie.php', data);

ОШИБКИ: $.get(url, data, successFunction).error(errorFunction); функция может вернуть какое-то сообщение об ошибке. НЕ РАБОТАЕТ c load()

**Возвращение XML:**

К примеру сервер вернул функции get XML файл вида

<?xml version="l.0" ?>

<message id="234">

<frorn>Boвa</frorn>

<content>Жaннa, давай вместе пообедаем.</соntеnt>

</message>

**Теги в XML мы придумываем сами**

$.get('xml.php', 'id=234', **processXML**);

function **processXML**(data) {

var messageContent=$(data).find('content').text();

} //выберет содержимое тега content в XML документе

**serialize()**

это функция конвертация данных для форм в строку запроса

<form method="post" action="login.php" id="login">

<input type="text" name="username" id="username">

<input type="password" name="password" id="password">

<input type="submit" name="button" id="button" value="Подтвердить" >

</form>

$('#login').submit(function() {

var formData = $(this).*serialize()*;

console.log(formData);

return false;

});

Вернет строку: *username=Max&password=141741* причем значения **username** и **password** берутся из атрибутов **name** каждого тега **input**

**Формат JSОN**

В сущности, **JSОN** представляет собой литерал oбъeктa javaScript, или коллекцию пар имя/значение. Вот пример данных **JSON**: обязательпо пара имя/значение в кавычках

{

“name” : ”Max”,

“gender” : ”male”

}

Когда веб-сервер отвечает на Аjах-запрос, он может возвратить строку в формате **JSON**. Сервер не посылает **JavaScript**-кoд: он посылает текст, отформатированный как объект **JSON**. Использовать код **JavaScript** невозможно до тех пор, пока строка не конвертирована в объект **JSON**. К счастью, библиотека **jQuеrу** предусматривает специальную функцию **$.getJSON()**, которая заботится обо всех деталях.

*$.getJSON(url, data, callback);*

*$.getJSON('contacts.php', 'contact=l23' ,processContacts);*

Данный код посылает строку запроса **contact=l23** в файл **contacts.php**, который на основе этой информации находит запрашиваемый контакт в базе данных и извлекает необходимые сведения. Результат отсылается браузеру и передается функции обратного вызова **processContacts**. Базовая структура обратного вызова выглядит так:

*function processContacts(data) {*

*}*

**Доступ к данным JSON**

Есть два способа доступа к данным объекта JSОN: *dot syntax* и *array notation*. **Dot syntax**, или точечный синтаксис осуществляеться как доступ к литералу:

*$.getJSON('contacts.php', 'contact=l23' ,processContacts);*

*function processContacts(data) {*

*}*

Файл *'contacts.php'* вернул к примеру строку

{ “name” : ”Max”,

“gender” : ”male” }

Функция *$.getJSON* преобразовала данные и передала аргументу data литерал, что аналогично

var data = {

“name” : ”Max”,

“gender” : ”male”

}

Теперь внутри функции обратного вызова мы можем получить доступ к объектам литерала

*function processContacts(data) {*

*data.name;// равно Мах*

*data.gender; // равно male*

*$(‘#user’).text(data.name);*

*}*

*Если у нас вложенный литерал объекта вида, то нужно его обрабатывать функцией $.each*

*var data = { “user1” : { “name” : “Max”*

*“gender” : “male” },*

*“user2” : {“name” : “Lena”,*

*“gender” : “female” }};*

*$.getJSON*(‘some.php’, ‘return 2 users’ , processing);

function processing(data) { //функция возврата получает литерал и присваивает его значение data

var name = ‘’, gender = ‘’; //объявляем две пересенные для вставки **html** к документ

$.each(data, function (user, info) { //user = user1, затем user2. Info = значению user1 затем user2

name = ‘<p>User Name: ‘ + info.name + ‘</p>’;

gender = ‘<p>User Name: ‘ + info.gender + ‘</p>’;

};

$(‘#someID’).html(name + user);

})

**Дополнительные возможности JQuery**

**Обход древа DOM:**

**$(this).find('.target').css("border","3px double red");** find ищет среди всех потомков

**$**(*'li.target'*,**this**); короткая запись для find

**$(this)**.**children('li.target').css();** children ищет только дочерние элементы

**$(this)**.**parent().css();** ищет **родителя** выбранного элемента

**$(this)**.**closest(‘#container’).css();** ищет 1го **предка** выбранного элемента с id **container**

**$(‘#container a’)**.**click(function() {**

**$(this)**.**siblings(‘a’).css();** ищет **элементы а того же уровня** выбранного элемента

**});**

**$(this).next(‘p’);**возьмет следующий элемент **‘p’** стоящий за **$(this)** на том же уровне DOM

**$(this).prev(‘p’);**возьмет предыдущий **‘p’** стоящий за **$(this)** на том же уровне DOM

[**nextAll(‘p’)**](http://jquery-docs.ru/Traversing/nextAll)**;** [**prevAll(‘p’)**](http://jquery-docs.ru/Traversing/prevAll)**;** тоже что и выше только выберет все элементы того же уровня

**ВНИМАНИЕ!**

<div>

<h1>Header</h1>

<p>text</p>

</div>

**$(‘div’).find(‘h1’).css(‘color’,’red’).find(‘p’).css(‘color’,’blue’);**

Данная выборка найдет **div** потом внутри его **h1** и поменяет цвет на красный, после этого она начнёт искать **p** внутри **h1** и соответственно не найдет, т.к. текущая выборка изменилась на **h1**. Во избежание этого необходимо использовать функцию **end()** и выборка вернется к самому первому значению!

**$(‘div’).find(‘h1’).css(‘color’,’red’).end().find(‘p’).css(‘color’,’blue’);**

**. append()** – вставит код в конец выбранного элемента

**. prepend()** – вставит код в начало выбранного элемента

**.after()** – вставит код в после выбранным элементом

**.before()** – вставит код перед выбранным элементом

**.replaceWith()** - полностью заменяет выбранные элементы (тег и все, что внутри) содержимым.

**.remove()** - удаляет выбранные элементы из модели DOM, фактически удаляя их со страницы.

**.wrap()** - заключает каждый элемент выборки в пару НТМL-тегов:

**$(‘p’).wrap(‘<div class=”wrapper”><div class=”inner-wrap”></div></div>’)**

**.wrapInner()** - окружает НТМL-элементами содержимое каждого элемента выборки.

**.unwrap()** - удаляет родительский элемент, окружающий выборку. Он не принимает аргументов!

**.empty()** - удалит все содержимое элементов выборки, но сама выборка при этом останется.

**Совершенствуем JS**

**Возвращаем длинну строки:**

var word = prompt('enter something','');

alert(word.length); // Вернет количество символов

**Изменение регистра строки:**

var word = “mAx” ,

check1 = word.toUpperCase(), // Вернет МАХ

check1 = word.toLowerCase(); // Вернет мах

**Поиск в строке: техника indexOf()**

string.indexOf(*'иcкoмaя* *строка'*);

var name = ‘Max.Ostepan’;

name.indexOf(‘.’); // Ищет точку в переменной и если не находит то возвращает -1 или возвращает число положения найденного запроса с начала строки

name.lastIndexOf(‘.’) // То же что и выше но ищет с конца, выведет туже цифру позиции(отсчитанную с начала строки)

**ищем тип браузера:**

var brouser = navigator.userAgent;

if (brouser.indexOf('Firefox') != -1) {

alert('You are using Firefox browser');

alert(brouser.indexOf('Firefox')); // Выведет 61

}

else if (brouser.indexOf('Chrome') != -1) {

alert('You are using Chrome browser');

alert(brouser.indexOf('Chrome')); // Выведет 76

}

**Navigаtor** -это один из объектов браузера, а **usеrAgent** - это свойство данного объекта.

**Работа с числами**

Если нужно преобразовать строку в число, то используем **Number(variable)** или **+variable**, однако, если переменная содержит не только цифры но и буквы(**20 years**) мы получим значение **NaN**, в этом случае можно использовать **parseInt(variable, 10)** для целых и **parseFloat(variable, 10)** для дробных чисел, число 10 я вляеться видом системы исчесления в которую необходимо преобразовать переменную 10-десятичная, 16-шестнадцатеричная и т.д.

**Проверка является ли переменная числом:**

isNan(variable) - проверит является ли переменная variable числом, к примеру если значение ‘+8.1’, ’-5’ или ‘10’ вернет значение true, в ином случае(’54 year’) значение будет false.

**Округление:**

Пригодится например если мы вычисляем значение в пикселах к примеру. Оно не может быть дробным и мы можем применить функцию **round**() метода **Math**

var number = 25.2,

finalNumber = Math.round(number); // будет равно 25, если значение после точки >= .5 округлит вверх, иначе округлит вниз

**Форматирование значений в валюте:**

Js не умеет выдавать числа вида 100.00 какие часто используются при указании цены, но мы можем преобразовать число в строку с необходимым нам количеством знаков после точки. **toFixed**():

var cost = 99,

totalCost = cost.toFixed(2) // выдаст нам 99.00 число в скобках указывает кол-во знаков после точки, если число дробное функция обрежет количество знаков до того какое указано в скобках

ВАЖНО! Можно применить только лишь к числу, если будет строка, тогда будет ошибка

**Генерация случайных чисел:**

Math.random()\*10 – генерирует число от 0 до 10 с дробным остатком вида 3.4448588848, 10 не выпадет никогда, максимум будет 9.999999, поэтому мы должны округлить полученное число с помощью Math.floor() которая округляет дробь в меньшую сторону 9.99999 будет равно 10 или Math.ceil() которая округлит в большую сторону 9.9999 будет равно 10, но тогда нуля мы никогда не получим:

Math.ceil(Math.random()\*6) будет давать значения от 1-6

Math.floor(Math.random()\*6) будет давать значения от 0-5

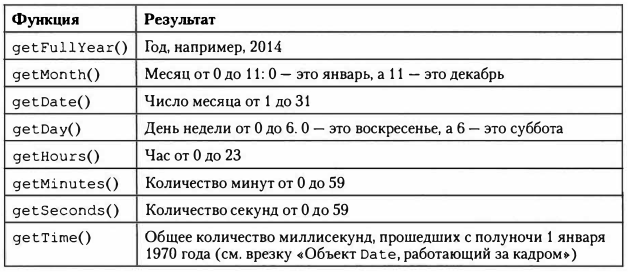
**Это можно использовать с массивами**

var array = [‘me’, ’5’, ’you’, ’hi’],

currentArrayItem = Math.floor(Math.random()\*array.lenght);

**Дата и время**

**var now = new Date();** - встроенный объект типа как и Image();



var currentDate = new Date(),

month = currentDate.getMonth(); //будет равен числовому индексу месяца который мы можем привязять к массиву с русскими названиями

var monthRus = [“Январь”,”Февраль”,”Март”, …];

alert(‘Сейчас месяц ’ + monthRus[month]);

**Оптимизация сценариев JavaScript**