Мы начинаем изучать JS и jQuery. Можно сказать, что это основная часть курса. Здесь мы обзорно посмотрим все, что нужно знать для front end разработки.

В своей работе мы используем только jQuery с дополнительными плагинами по необходимости. Мы не используем различные фреймворки типа Knockout и AngularJS. Вместо этого у нас есть свой небольшой фреймворк, который содержит все необходимые нам функции. В Учебном проекте мы подробно рассматриваем его на практике.

Основы JS

Переменные не имеют типа, вернее он определяется динамически:

```
var i =0;
var s = "String";
var t = {a:111, b: "str"}; -  //Объект
var ar = [111, 222, 333]; -  //массив
```

Важные функции:
parseInt - привести строку к числу
parseFloat - привести строку к числу с точкой
Math.round - округление числа
toFixed - задает формат вещественного числа.

Управляющие конструкции - это циклы, условия, множественный выбор.

Циклы:

```
for(var i=0; i<10; i++) {
}

var i=0;
while(i<10){
        // …
        i++;
}

foreach (var prop in obj){ //проход по свойствам объекта JSON.
        //…
}
```

jQuery цикл - можно проходить по массиву или по элементам цепочки (о цепочках будет дополнительно описано ниже):

```
$.each(array1, function(i, item){
```

```
// для массива
});
$('div').each(function(){
 // для цепочек
        alert($(this).html());
});
```

Условия:

```
if(i=10){
}else{
}
```

```
switch(i){
        case 1:
                // ...
                break;
        case 2:
                // ...
                break;
        default:
                // ...
                break;

}
```

Оператор IF в одну строку:

```
var i = (x=="22") ? 1 : 2;
```

Сложные условия и неявные ошибки:

```
if(x1>3 && x2<4  || (x3>3 && x4<65) )
if(x=3) // ошибка  -> правильно if(x==3)
if(x)  ... (это значит что если x - не пустой объект...)
x>0
x!="" x - непустая строка
```

Оптимизация условий и циклов:

- использование массива для конкатенации строк (это работает быстрее, чем объединять строки сложением):

```
var s = [];
s.push("s1"); s.push("s2");  s.push("s3");
var str = s.join("")
```

- если цикл длинный - то лучше for чем $.each
- ставьте первым условие которое наиболее вероятно при ИЛИ (x==0 || y<0) или наоборот при И (x==0 && y<0)
- как выйти из цикла $.each и for

для for - это break
для each - это лог переменная - isOut

```
var isOut = false;
$('.div').each(function(){
	if(isOut==true) return;
	...
	if(abc) then isOut = true;
});
```

Разберем, что такое JSON

JSON - объект в JS:

```
var obj = {
	a: 12,
	b: "sss",
	ar: ["111", "222"],
	c: {
		x1: 1,
		x2: 3
	}
}
```

JSON используется для передачи данных в AJAX запросах.
AJAX - технология, разработанная на JS, позволяющая совершать асинхронный обмен данными между браузером и сервером.

Для обращения к свойствам используется ".":

```
obj.a
obj.c.x1
```

Важные функции для работы с JSON:
JSON.stringify({a:1, b: 2}) - преобразует JSON в строку
eval("("+str+")") - преобразует строку в объект (заметьте, что обязательно строку оборачивать в дополнительные скобки).

Интересный момент - в качестве членов JSON-объекта могут быть функции. Именно этот момент мы используем для организации наших компонентов.

Общая структура компонентов такова:

```
var comp ={
	options: {
		a:0, b:0
```

```
	},
	init: function(options){
		comp.options = $.extend(comp.options, options); //инициализация свойств объекта
		// ...
	},
	method1: function(){},
	method2: function(){}
};
$(function(){
	comp.init({a:1, b:2}); //вызов-инициализация работы компонента
});
```

Все JS-компоненты в своих проектах мы строим именно по такому принципу. Есть центральный метод init, в котором происходит инициализация всего компонента и привязываются основные события к обработчикам. Остальные методы - это просто функции для вызова внутри или извне. Есть недостаток в такой организации - служебные методы не скрываются от внешней среды. В общем случае это не проблема - можно организационно решить, что все элементы в private - это скрытые члены.

Примечание: при выполнении практических заданий используйте именно такой шаблон.

jQuery

jQuery - это библиотека, которая значительно упрощает использование JS.
Философия jQuery основана на работе с цепочками. Сначала Вы выделяете группу элементов на странице, затем начинаете делать обработку этих элементов, причем каждая новая функция применяется к цепочке элементов. Например:

```
$('div.item').css('color', 'red').addClass('ttStep').show(200);
```

Выделение цепочек:
Вы можете выделить несколько элементов на странице с помощью расширенных CSS селекторов $('selector'):
$('div span') - выбрать все span которые находятся внутри div

Какие бывают селекторы:
name - выбрать все теги name
.class - выбрать все с классом class
#id - выбрать элемент с id=id
:checked - выбрать все отмеченные элементы(галочки, радиокнопки)
:visible - выбрать все видимые элементы
.class1>div - выбрать все div, которые находятся непосредственно в .class1

input[type=text] - выбрать текстовые поля
#select option[value=1] - выбрать option с атрибутов value=1 в комбике #select

Селекторов очень много - изучите эту тему досконально, т.к. в дальнейшем это сильно упростит вашу жизнь.

Задание 1. Google: jquery selector. Попробуйте на практике каждый селектор.

Зачем нам выделять элементы? Чтобы выполнять с ними операции. Пример:
$('div').hide() - скроет все Div на странице.

Работа с DOM
DOM - это структура Вашего HTML документа. С помощью jQuery Вы можете легко менять, добавлять/удалять элементы.
Все функции, которые мы здесь рассмотрим, необходимо будет еще изучить отдельно (Google: jquery using {function}).

Добавление элементов в тег (варианты):

```
$('body').append("<div class='s1'>1111</div>");
$("<div class='s1'>1111</div>").appendTo($('body'));
$('.d1').html("<div class='s1'>1111</div>");
```

Удаление элементов:

```
$('.d1').remove();
$('.d1').empty() - очищает элемент (аналог $('.d1').html(''));
```

Изменение атрибутов и значений элементов:

```
$('.t1').attr('data-x', 1212);
$('.t1').removeAttr('data-x');
```

Получение и установка значения элемента:

```
var s = $('.t1').html();      $('.t1').html('1111') //для html тегов (span, div и т.д.)
var t = $('.t1').val();       $('.t1').val('1111'); //для форм (текстовое поле и т.д)
```

Управление видимостью элементов, как можно скрыть элемент (плавно и сразу)

```
$('.el1').hide(200).addClass('hide').fadeOut(1000);
// скрытие (параметр - время анимации)
$("div").show().removeClass('hide').fadeIn();
// показать скрытый элемент
```

Задание 2. Найдите наиболее удобное руководство по CSS селекторам и jQuery функциям - в дальнейшем мы будем ссылаться в курсе на лучший найденный ресурс.

Работа с классами и стилями

Добавление и удаление классов, переключение класса:

```
$('.div1').addClass('c1 c2');
$('.div1').removeClass('c1');
$('.div1').toogleClass('c1');
//если был класс - то убирает его, если не было - то добавляет
$('.div1').hasClass('c1'); //проверяет есть ли класс у элемента
```

Очень часто классы используют, как индикаторы состояния. Например, класс hide можно использовать как индикатор видимости:

```
$('.s1').hasClass('hide'); //выделен ли элемент?
```

Соответственно в CSS задаем:

```
.hide{ diplay:none;  }
```

Работа со стилями css:

```
$('.div1').css('color', '#ccc');
$('.div1').css({'color': '#ccc', 'font-size': '12px'});
```

Работа с флажками:

```
input[type=checkbox] //выбрать все флажки
$('.ch').is(":checked") //как проверить - выделен ли флажок?
$('.ch').attr('checked', 'checked')
//или в нов версиях:
$('.ch').prop('checked', true); //установка / снятие  флажка
//как выбрать только выделенные флажки?
$('.ch:checked')
```

Работа с select:

```
select.class1 option:selected //выбрать выделенный элемент списка
$('select.class1 option[value=3]').attr('selected', 'selected')  / установить выбранный элемент в списке
//как выбрать только выделенные флажки?
$('select.class1 option:selected').val() - получить выделенное значение
```

Функции по работе с цепочками

Функция filter - для custom создания цепочек.

Все .item, где значение равно 1:

```
$('div.item').filter(function(){ return $(this).html()=="1" });
```

Функции parent, closest, find и их использование:

$('div.item').parent() - выбрать всех непосредственных родителей для .item

closest('.cont') - ближайший сверху, который подпадает под критерий

$('.s1').find('.item') или *$(".item", this)* - найти среди текущих элементов в цепочке (т.е. мы задаем контекст поиска элементов в рамках определенной группы, а не во всем html).

Данные функции позволяют вам делать очень гибкие выборки.

Задание 3. Напишите селектор, который позволяет выделить все Div, которые следуют непосредственно за параграфами, содержащими выделенные галочки.

Работа с событиями

Когда вы нажимаете на кнопку или отмечаете флажок - можно эти действия обрабатывать в jQuery.

Событие для флажка или select:

```
$('.ch').change(function(){});
```

Обработать клик по кнопке:

```
$('.btn').click(function(){});
```

Возможно привязать событие для еще не созданной кнопки (т.е. кнопка может создаться потом, после объявления обработчика, где-то в коде JS). Здесь обработчик привязывается к параметру $(...) (в данном случае в document), и затем идет отслеживание дочерних элементов в момент возникновения события:

```
$(document).delegate('.link1', 'click', function(e){});
```

Отмена действия по умолчанию и всплытия события:

e.preventDefault(); - отменяет действие по умолчанию

e.stopPropagation(); - заставляет прекратить распространение событие вверх по DOM структуре (Например, span находится div. У обоих есть свои обработчики клика. По умолчанию оба последовательно вызовут обработчики. Если для span вызвать stopPropagation - то для div обработчик уже не вызовется). Будьте осторожны с использование StopPropagation - иногда это может сказываться на вызове других элементов (вложенных в элемент с соответствующим обработчиком). Т.е. если какой-то обработчик не срабатывает и у вас используется stopPropagation - попробуйте его отключить и проверить работу без него.

События лучше объявлять в init и один раз - иначе у вас могут быть неявные ошибки по двойному срабатыванию обработчиков.

Как узнать, какую клавишу нажал юзер (с shift или без) - следует использовать для этого событие keypress (keyup, keydown):

```
$('.input1').keypress(function(e){
        alert(e.which);
        console.log(e); //f12 / console - смотрим объект e
});
```

Есть такая полезная штука, как анонимные функции. Они могут передаваться в качестве параметров в другие функции. Пример:

```
$('.sq1').hide(200, function(){
        $(this).remove();
});
```

Мы сначала скрыли элемент (в течение 200 мс), а потом вызвали функцию (callback), в которой удалили этот элемент.
Обратите внимание на this - это контекст функции. Очень часто в jquery-цепочках это какой-то конкретный выбранный элемент.

Также следует еще раз сказать о цепочках (за что многие и любят jQuery). Работая с одним набором, вы можете совершать различные действия над ним:

```
$('.s1').addClass('x1').removeClass('x2').hide();
```

Здесь для всех тегов с классом s1 мы сначала добавили один класс, потом убрали другой класс и в итоге скрыли элемент.

Лучше всего весь код jQuery запускать из функции загрузки страницы:

```
$(function(){
        // … когда страница полностью загрузится, это код начнет выполняться
})
```

Настоятельно рекомендую Вам очень хорошо изучить все функции jquery, селекторы и события. Это очень Вам пригодится, когда Вы будете создавать свои JS-приложения.

Чит листы для jQuery - http://www.sitepoint.com/10-jquery-cheat-sheets/

Работа с console

Если вы в браузере Chrome или Firefox нажмете F12 - то снизу появится панель с различной полезной информацией о работе страницы. На мой субъективный взгляд, панель в Chrome удобнее. Какие там есть полезные вкладки:
Elements - можно посмотреть реальный HTML, поменять его, посмотреть CSS, поэкспериментировать с ним. Можно выбрать на странице элемент и через правую кнопку мыши посмотреть, как он выглядит в HTML (пункт контекстного меню Просмотреть элемент)

Sources - показывает какие скрипты, стили загружаются. Возможность отладки по шагам.
Network
Console - вывод ошибок JS, а также различных Ваших сообщений, которые делаются в коде Вашего JS приложения с помощью console.log(text). При отладке приложения используйте именно эту функцию, а не alert для проверки промежуточных значений. Также Вы можете использовать ключевое слово debugger для пошаговой отладки в браузере - на этой точке браузер остановит выполнение скрипта и Вы сможете по шагам выполнить код и посмотреть все переменные.

Console Вы также можете использовать для трассировки Вашего приложения в плане оптимизации. У нас есть в библиотеках специальная функция as.sys.trace - она показывает сколько времени прошло с последнего вызова функции. Таким образом можно находить медленные участки кода.

Примечание. Лучше не вызывайте просто console.log(text). В IE объект console будет не опознан и вы получите ошибку. Вызывайте лучше следующим образом:
console && console.log(text). Если первое выражение будет false, то второе будет не выполняться.

Вообще используйте эти 2 паттерна:

- `var a = s && s.func1();` - проверка существует ли элемент s
- `var a = s || 'default value'` - установка значения по умолчанию (если s не определен).

Хороший прием - использование оберток.
Это относится не только в JS, но вообще к программированию в целом. Старайтесь для типовых функций (даже очень простых) использовать функции обертки. Мы немного касались этого раньше, когда говорили про проверку прав. Обертки тем хороши, что Вы всегда можете их потом поменять. Если же обертки нет - то вам придется менять поведение во многих местах программы.

Хороший пример - это вызов console.log. Представьте себе, что Вы наставили повсюду в своем приложении такие штуки. Потом выясняется, что в IE эта штука не применяется и выдает ошибку. Вы начинаете проходить по коду и заменять его на console && console.log(). Ну слава Богу, хоть и через Ж, но все такие проблему решили. Чуть позже выясняется, что хорошо бы все эти сообщения логировать через ajax запрос в базу. Опять проблема.
Чтобы этого не было, мы сразу создаем общий метод в глобальной вашей библиотеке

```
lib.log = function(text, options){
        options = options ||{}l
        console && console.log(text)
}
```

Зачем options? Затем, что в будущем возможны различные дополнительные опции. Например передача цвета в console.log или тип события (Ошибка, Опасность, Уведомление и т.д.). При этом если options не указан - то ему просто присваивается значение undefined (в коде мы его заменяем в этом случае на пустой объект, чтобы не было ошибки).
Далее Вы используете только эту обертку. Если что-то нужно будет поменять - Вы просто меняете этот метод, а не бегаете по всему коду.

Задание 4. Найдите, как окрашивать в разные цвета строки в console.log()

Задание 5. Найдите подобные чит листы для C#, Bootstrap и других технологий. Добейтесь того, что Вы можете что-то вразумительное сказать про каждую функцию. Пришлите по 3-5 ссылок для каждого найденного раздела.

На этом мы заканчиваем введение по jQuery. У Вас появилось довольно большое поле для экспериментов. Пробуйте создавать различные механизмы.

Задание 6. Реализуйте следующее в виде отдельных компонентов страницы с использованием предложенного шаблона для js-компонентов:

- добавление элементов на страницу по кнопке
- калькулятор на jQuery
- динамически сформированный выпадающий список
- часы или таймер (функции setTimeout и setInterval)

В следующей главе мы перейдем в технологии AJAX, которая позволяет управлять данными на странице, запрашивая данные с сервера незаметно для пользователя. Этот тренд уже длится около 13 лет (с 2002 года, когда это внедрили у себя Google в Gmail) и скорее всего AJAX-составляющая сайтов будет с каждым годом только расти. Очень вероятно, что веб-приложения в итоге полностью заменят настольные приложения.

Задание 7. Приведите свои доводы/размышления за или против этой теории (полное замещение настольных приложений).