
Содержание

Введение	1.1
Глава I - Старт	1.2
Hello World!	1.2.1
Переменные	1.2.2
# Типы данных в JavaScript	1.2.3
Работа с селекторами	1.2.4
#Цикл for	1.2.5
Генерация DOM-объектов	1.2.6
# Конструкция if	1.2.7
# alert prompt confirm	1.2.8
#COFFEE Логические операторы	1.2.9
#Массивы	1.2.10
# Применение флагов	1.2.11
# Оператор деления по модулю	1.2.12
# Повторения событий	1.2.13
# Цикл while	1.2.14
# Циклы с неизвестным количеством итераций	1.2.15
Объекты	1.2.16
# Тренировочные задачи	1.2.17
# Контрольная работа	1.2.18
Глава II - Основы	1.3
Массивы	1.3.1
# Добавляем DOM	1.3.2
Функции	1.3.3
# Ассоциативные массивы	1.3.4
Контрольная работа	1.3.5
Глава III - События	1.4
Способы установки обработчика	1.4.1
# Использование this в обработчиках	1.4.2
Объект event	1.4.3

Стандартные события в JavaScript	1.4.4
Всплытие событий	1.4.5
# Делегирование событий	1.4.6
# Обработчики событий по умолчанию	1.4.7
#Интервалы и задержки	1.4.8
# Debouncing	1.4.9
Интересные материалы	1.4.10
Контрольная работа	1.4.11
Глава IV - DOM	1.5
#COFFEE DOM, BOM и JS Объекты	1.5.1
# Типы узлов в DOM	1.5.2
# Навигация по DOM	1.5.3
# Создание и модификация узлов	1.5.4
# Работа с формами	1.5.5
# Работа с CSS	1.5.6
Задания для контрольной(part4/knowledgecheck.md)	1.5.7
Глава V - BOM	1.6
# AJAX и JSON	1.6.1
# navigator, screen, location	1.6.2
#COFFEE Продвинутая консоль	1.6.3
# history	1.6.4
# cookies	1.6.5
#COFFEE Создаем простейшее SPA	1.6.6
#Drag-n-Drop	1.6.7
#clipboardData	1.6.8
#FileReader	1.6.9
#COFFEE LocalStorage	1.6.10
Глава VI - Регулярные выражения	1.7
Синтаксис регулярных выражений	1.7.1
match	1.7.2
split	1.7.3
replace	1.7.4
Глава VIII - jQuery	1.8
Быстрый старт	1.8.1

#Работа с селекторами	1.8.2
#События	1.8.3
# Анимация	1.8.4
# AJAX	1.8.5
# Модификация DOM	1.8.6
# Работа с формами	1.8.7
# Плагины	1.8.8
#COFFEE Техника Map-Reduce	1.8.9
#COFFEE Promises	1.8.10
Код jQuery	1.8.11
ООП в JavaScript	1.9
Способы создания объектов	1.9.1
this	1.9.2
Замыкания	1.9.3
Паттерн Модуль	1.9.4
# Прототипы	1.9.5
Глава VII - Функциональное программирование	1.10
Promises	1.10.1
#COFFEE Рекурсия	1.10.2
# Контекст вызова функции	1.10.3
# Техника Map-Reduce	1.10.4
# Каррирование	1.10.5
Мемоизация	1.10.6
Глава IX - Веб-приложения	1.11
# Концепция	1.11.1
# "Инсталлируемое" веб-приложение	1.11.2
# Работа в оффлайне	1.11.3
# Работа с камерой	1.11.4
# WebRTC	1.11.5
# Service Workers	1.11.6
# Web Push	1.11.7
#Работа с файлами	1.11.8
COFFEE WebComponents	1.11.9

Глава X - NodeJS	1.12
#NodeJS Старт	1.12.1
Настройка Sublime для NodeJS	1.12.2
#Создание простейшего файл-сервера	1.12.3
#Работа с параметрами адресной строки	1.12.4
POST-параметры	1.12.5
Загрузка файла на сервер	1.12.6
Cookies в NodeJS	1.12.7
#События	1.12.8
Модуль fs. Работа с файлами	1.12.9
#Модули и require	1.12.10
NPM	1.12.11
async/await	1.12.12
process	1.12.13
Глава XI - Express	1.13
# Основы Express	1.13.1
Роутинг на Express	1.13.2
Express MiddleWare	1.13.3
Работа с GET-параметрами на Express	1.13.4
Работа с POST-параметрами на Express	1.13.5
Загрузка файла на сервер, с помощью Express	1.13.6
Cookies на Express	1.13.7
Unit-тесты	1.13.8
Pug	1.13.9
Глава XII - MongoDB	1.14
MongoDB Старт	1.14.1
Настройка Mongo GUI	1.14.2
MongoDB CRUD	1.14.3
Limit,skip,count,sort	1.14.4
Авторизация	1.14.5
Fulltext search	1.14.6
Геолокация	1.14.7
NoSQL injections	1.14.8
COFFEE making grabber	1.14.9

Создаем чат на NodeJS	1.14.10
MEAN	1.14.11
Паттерны	1.14.12
WebSockets	1.14.13
Материалы	1.14.14
Глава XIII - Angular	1.15
Angular Старт	1.15.1
ng-init	1.15.2
ng-repeat	1.15.3
Модули	1.15.4
Контроллеры	1.15.5
Сервисы	1.15.6
Директивы	1.15.7
Кастомные директивы	1.15.8
Scope директив	1.15.9
\$routeProvider, \$locationProvider	1.15.10
Factory	1.15.11
Фильтры	1.15.12
Unit-тесты в Angular	1.15.13
# \$watch \$digest \$apply	1.15.14
# \$broadcast, \$emit, \$on	1.15.15
# \$index,\$event,\$log	1.15.16
Компоненты	1.15.17
require, \$onInit	1.15.18
history, \$location, \$route	1.15.19
Глава XIV -Webpack	1.16
Webpack Start	1.16.1
Разделение исходников и результата	1.16.2
HtmlWebpackPlugin	1.16.3
CSS в Webpack'e	1.16.4
Babel	1.16.5
Webpack More	1.16.6
Минификация	1.16.7

Глава XV - React	1.17
React Старт	1.17.1
Компоненты и свойства	1.17.2
Работа с DOM-элементами	1.17.3
State	1.17.4
События	1.17.5
Рендеринг на сервере	1.17.6
Глава XVI - Redux	1.18
Redux Старт	1.18.1
Глава XVII - Vue JS	1.19
Vue Старт	1.19.1
Базовые директивы	1.19.2
Обработка событий	1.19.3
Экземпляр Vue	1.19.4
Шаблоны	1.19.5
Компоненты	1.19.6
Дополнения	1.20
#Инструменты JavaScript разработчика	1.20.1
#ESLint	1.20.2
Вдохновение	1.20.3
Ресурсы	1.20.4
Эксперименты	1.20.5
Codepens	1.20.6
Материалы для чтения	1.20.7
Книги	1.20.8
Вопросы к собеседованию	1.20.9
Работа с Git	1.20.10
Ветки на Git	1.20.11
Интеграция Slack с Github	1.20.12
Задания для практики	1.20.13

Странный JavaScript

Конспект лекций по JavaScript и NodeJS. Целью данной книги, является рассмотрение вопросов, связанных с использованием JavaScript в Web-разработке.

Если Вы уже умеете программировать можете смело пропускать Главу I

Данный курс предполагает, что Вы уже знаете HTML/CSS в соответствии со следующей программой <http://dmitrytinitilov.gitbooks.io/unexpected-html>

P.S. Данная книга не подходит для детских групп и применима только к людям достигшим совершеннолетия, так как содержит грубую лексику, а также сцены насилия(в том числе и над Вашими мозгами)

Глава I - Старт

Данная глава предназначена для тех, кто не программировал никогда и ни на чем. Если у Вас есть навыки программирования посмотрите типы данных в первой главе и работу с объектами во второй и переходите к третьей.

Hello World!

Создаем index.html с таким кодом

```
<script>
  alert('Hello world!');
</script>
```

Запускаем в браузере. При запуске у нас должно появиться всплывающее окно.

Переменные

Переменная в JavaScript оформляется с помощью конструкции var

```
var x=5;
```

Если мы хотим увеличить значение переменной с целым числом, то мы можем это сделать вот так

```
var x = 7;  
x=x+1;  
console.log(x); // 8
```

Hoisting

Речь идет о том, что если мы присваиваем переменной значение в коде, то она будет определена и выше, но со значением Undefined

```
console.log(x); // Undefined  
  
var x = 5;
```

<https://habrahabr.ru/post/127482/>

Типы данных в JavaScript

String

```
var a = 'Hello';

var b = "World!";

var c = a+' '+b; // Hello World!
```

Number

Integer

```
parseInt
NaN (Not a number)
Inf (Infinity)
```

Float

```
var x = 0.1 * 0.2;
document.write(x);
```

0.0200000000000004

Boolean

true 1,2,-1,'0' false 0

Objects

NULL

В JavaScript есть элементы, которые мы храним по ссылке, но ссылочного типа нет. Это вынуждает выделять целый тип под ссылки, которые никуда не ведут

Undefined

Определенная переменная, значение которой не задано получает значение undefined.

```
var x;
console.log(x); //undefined
```

Подробнее о NULL и Undefined <http://frontender.info/exploring-the-abyss-of-null-and-undefined-in-javascript/>

typeof

Преобразование типов <https://habrahabr.ru/post/312172/>

Интересная заметка об особенностях JavaScript <http://programmers-notes.blogspot.com/2012/02/javascript.html>

Работа с селекторами

Изначально основной задачей JavaScript'a было модификация страницы и реакция на действия пользователя.

document.querySelector

Допустим у нас есть div с классом `.block`

```
<div class="block">  
</div>
```

Мы хотим начать работать с ним через JavaScript. В этом нам поможет стандартная функция `document.querySelector`, которая по селектору возвращает DOM-объект.

```
var rect = document.querySelector('.block');
```

Важный момент состоит в том, что в `querySelector` мы можем применять любой CSS-селектор!

.innerHTML

Получения ссылки на DOM-объект для нас безусловно мало, хочется с ним, что-то сделать(для чего ж мы эту ссылку получали!?). Для начала давайте добавим текст внутрь

```
rect.innerHTML = 'теперь в прямоугольнике есть текст';
```

.className

Давайте поменяем класс у блока, сделать это можно вот так

```
rect.className = 'unblock';
```

Обратите внимание, что тут никаких точек не нужно, потому что это имя класса, а не селектора.

document.querySelectorAll

Но что делать, если у нас несколько элементов с одним классом. В этом случае `document.querySelector` вернет лишь первый из них.

Для решения этой проблемы есть функция `document.querySelectorAll`

Допустим у нас есть три div'a с классом `block`

```
<div class="block">
</div>
<div class="block">
</div>
<div class="block">
</div>
```

С помощью `querySelectorAll` можно получить коллекцию блоков

```
var blocksCollection = document.querySelectorAll('.block');
```

Если мы хотим работать с первым блоком, то нужно будет обратиться к нулевому элементу коллекции `blocksCollection[0]`

Общее количество элементов коллекции можно получить через свойство `.length`

```
var blocksCollection = document.querySelectorAll('.block');

blocksCollection[0].innerHTML = 'Первый элемент коллекции';
blocksCollection[1].innerHTML = 'Второй элемент коллекции';
blocksCollection[2].innerHTML = 'Третий элемент коллекции';

console.log('Количество элементов '+blocksCollection.length);
```

Написание собственной библиотеки

Допустим мы хотим, чтобы `div` с атрибутом `stripes=ok` становился полосатым. При этом, если пользователь захочет использовать нашу библиотеку такие `div`'ы ему придется подготавливать самому (о чем мы его предупредим в описании библиотеки... наверное). То есть пользователь подготавливает файл следующего вида

```
<div class="block" stripes=ok>
</div>
```

Теперь наша задача подготовить два файла `stripes.js` и `stripes.css`

`stripes.css` будет содержать `css`-класс, который и будет добавлять полосы

```
.stripes {  
    background-image: repeating-linear-gradient(-45deg,  
        transparent,  
        transparent 20px,  
        black 20px,  
        black 40px  
    )  
}
```

и stripe.js, который будет добавлять этот класс к div'ам с атрибутом stripes=ok

```
var block = document.querySelector('[stripes=ok]);  
  
//добавим наш класс к текущим  
block.className += ' stripes';
```

Теперь, чтобы наша библиотека заработала у пользователя, нужно, чтобы он подключил наши css и js-файлы к своему коду.

То есть у пользователя должен получиться вот такой вот код

```
<link rel="stylesheet" href="stripes.css">  
  
<div class="block" stripes=ok>  
</div>  
  
<script src="stripes.js">  
</script>
```

Практика:

1. Поменять цвет блока с помощью JavaScript
2. Есть десять блоков. Поменять блок с заданным номером.Номер задается через переменную i, либо вводится с клавиатуры через prompt
3. Есть список. С помощью js сделать так, чтобы подпункты скрылись.

for

Цикл for чем-то напоминает револьвер. У нас есть определенное количество шагов(выстрелов). На каждом шаге происходит полезное действие(например ищем следующую мишень). После каждого шага(выстрела) счетчик увеличивается(происходит переход к следующему патрону)

Вот наглядный пример из кинофильма Deadpool <https://www.youtube.com/watch?v=BYqjrMdkK8k>

В случае с циклом while мы и не знаем сколько шагов(выстрелов) нам понадобится. Например неясно, сколько раз будет выполняться вот такой цикл

```
while(people>0) {  
    kill();  
}
```

Но в случае с for'ом количество шагов как правило известно. Единственное, что программисты считают выстрелы... шаги с нуля.

```
for (shots=0;shots<bullets;shots++) {  
    kill();  
}
```

Но конечно обычно в качестве счетчика используют переменную i, но если это сделает Ваш код более наглядным Вы можете отойти от этого правила.

Вот более стандартный пример с выводом значения счетчика в консоль.

```
var n=10;  
  
for (i=0;i<n;i++) {  
    console.log(i);  
}
```

document.querySelectorAll

Допустим у нас несколько элементов с классом block и мы хотим всех их пронумеровать


```
var blocksCollection = document.querySelectorAll('.block');

for (i=0;i<n;i++) {
    blocksCollection[i].innerHTML = i;
}
```

document.write()

Попробуем выводить html-код из под javascript. Код ниже выведет 10 блоков

```
var n=10;
for (i=0;i<n;i++) {
    document.write('<div class="block"></div> ');
}
```

Практика:

1. Вывести 10 синих блоков. Седьмой блок сделать зеленым.
2. Вывести 10 блоков, так чтобы их цвета чередовались
3. Вывести шахматную доску 7 на 7
4. Возвести 2 в степень n
5. Возвести a в степень b

Генерация DOM-объектов

createElement и **appendChild** createElement - создает узел по названию тега
appendChild - переносит узел внутрь другого узла(см пример)

```
var btn = document.createElement("BUTTON");           // Создаем элемент <button>
var t = document.createTextNode("CLICK ME");           // Создаем текстовый узел
btn.appendChild(t);                                     // Добавляем текст в кнопку <button>

document.body.appendChild(btn);                       // Добавляем <button> в <body>
```

Обратите внимание, что если мы работаем с уже созданными элементами, то они просто переносятся, без создания копии!+

http://www.w3schools.com/jsref/met_document_createelement.asp

Генерация картинки

Еще один пример, затрагивающий генерацию элемента img

```
var image = document.createElement("img");

image.src = 'odessa.jpg'

document.body.appendChild(image);
```

Добавление блока внутрь существующего блока

Допустим у нас уже есть какой-то блок на странице

```
<div class="outer">
</div>
```

и мы хотим внутрь нашего блока добавить свежесозданный блок

```
var outer = document.querySelector('.outer');

var inner = document.createElement("div");
inner.className = 'inner';

outer.appendChild(inner);
```

То есть последняя строчка показывает нам, что мы вполне можем добавить один DOM внутрь другого с помощью `appendChild`.

Добавление нескольких элементов

```
for(i=0;i<10;i++) {  
    var smallBlock = document.createElement("div");  
    smallBlock.className = 'block';  
  
    document.body.appendChild(smallBlock);  
}
```

Обратите внимание, что мы в цикле каждый раз создаем объект через `createElement`. Дело в том, что `appendChild` не просто добавляет элемент в указанное место, а переносит также его со старого. То есть, если мы не будем создавать новый элемент, мы будем постоянно переносить блок со "старого" места на "старое"

Конструкция if

Оператор if позволяет выполнять код только при соблюдении определенных условий. Например

```
if (a>5) {  
    alert('Много');  
}
```

Код внутри фигурных скобок выполнится только при условии, что переменная a больше 5. Вообще этот код можно "прочитать" как если a больше 5, то выполнить alert.

Вот такой код выведет 'Много'

```
var a = 7;  
if (a>5) {  
    alert('Много');  
}
```

Для ситуаций, когда нам нужна какая-то реакция, когда условие все-таки не выполнилось можно использовать более расширенный вариант:

```
var a=7;  
if (a>5) {  
    alert('Много');  
} else {  
    alert('Мало');  
}
```

Сравнения в if'e это три знака равно. Например

```
var vse='ploho';  
if (vse === 'horosho') {  
    console.log('Ура!');  
}
```

При таком сравнении не происходит приведения типа переменной vse

Логические операторы

Допустим для нас критично выполнение двух условий, например a больше 5 и b меньше 11, тогда мы можем это записать как

```
if (a>5 && b<11) {  
}
```

Проверим находится ли а в промежутке от 10-ти до 30-ти

```
if (a>10 && a<30) {  
    console.log('а в промежутке');  
} else {  
    console.log('Увы, Вы не угадали с промежутком');  
}
```

Если нам нужна проверка, что переменная не равна какому-то значению нужно использовать оператор !=

```
if (vse!='ploho') {  
    console.log('vse ne tak ug ploho');  
    console.log('na segodnyashniy den');  
}
```

```
&& - и  
|| - или  
!= - не равно  
! - отрицание
```

Практика:

1. Вывести максимальное из трех чисел
2. Проверить могут ли три числа быть сторонами прямоугольного треугольника
3. Проверить могут ли три числа быть треугольником
4. Ввести логин, пароль — выдать правилен ли он
5. Выделить блоки, в которых числа больше определенных значений

alert prompt confirm

prompt открывает окно на запрос данных у пользоваля.

```
var s = prompt('Введите Ваше имя');  
  
alert('Добрый день, '+s);
```

prompt всегда возвращает строку. Чтобы преобразовать ее к числу у нас есть два варианта

```
var a = parseInt(prompt('Введите число'));
```

или более короткий

```
var a = +prompt('Введите число');
```

Практика:

1. Запрашиваем у пользователя два числа и выводим их сумму
2. Запрашиваем у пользователя числа, до тех пор пока не введет 0. Выводим сумму всех ранее введенных чисел
3. Запрашиваем у пользователя числа, до тех пор пока не введет 0. Выводим максимальное из всех ранее введенных чисел.
4. Запрашиваем у пользователя числа, до тех пор пока не введет 0. Из всех ранее введенных чисел выводим второе по величине число после максимального.
5. Задача угадай число (загадываем число, пользователь пытается угадать, вводя числа. Мы возвращаем больше или меньше)
6. Камень ножницы бумага (вариант вводится как буква, а компьютер выбирает свой через rand)

#COFFEE Логические операторы

Массивы

Допустим у нас есть массив чисел и мы хотим подсчитать их сумму. Для решения нам понадобится переменная `sum`, в которую мы будем накапливать нашу сумму. Изначально в ней будет 0

```
var arr=[6,8,5,11];
console.log(arr[1]); //8

console.log(arr.length); //4
console.log(arr[arr.length-1]); //11 , всегда выводится содержимое последнего элемента

var sum = 0;

for (i=0;i<arr.length;i++) {
    sum = sum + arr[i]; //на каждом шаге цикла добавляем к sum i-й элемент массива
}
```

Попробуем работать с блоками через `querySelectorAll`. Создадим 10-ть блоков. Сделаем это через Emmet, то конструкция `.block*10` должна превратиться в

```
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
<div class="block"></div>
```

Поменяем с помощью JavaScript классы в наших блоках

```
var blockList = document.querySelectorAll('.block');

for (i=0;i<blockList.length;i++) {
    blockList[i].className = 'changed';
}
```

Попробуем создать набор картинок `1.jpg`. Чтобы сделать это через Emmet наберите `img[src=1.jpg]*10` и нажмите `tab`. В результате получим


```










```

Поменяем названия этих картинок на 2.jpg

```
var imgList = document.querySelectorAll('img');

for (i=0;i<imgList.length;i++) {
    imgList[i].src = '2.jpg';
}
```

Практика:

1. Есть массив чисел. Подсчитать количество чисел большее 10
2. Поменять класс в каждом втором блоке.
3. Поменять класс в каждом третьем блоке.
4. Вывести номера внутри каждого блока.
5. Есть массив с названиями картинок. Вывести галерею картинок
6. Есть два массива с названиями картинок(жуки и бабочки). Вывести картинки так, чтобы жуки и бабочки чередовались
7. Есть зеленые блоки с числами внутри. Те блоки, у которых число внутри больше 10, должны стать красными
8. Вывести картинки в стиле Masonry Grid (пример сайт <http://pinterest.com>)

Применение флагов

Концепцию флагов проще всего понять на тюбике зубной пасты. Сейчас перед покупкой зубной пасты мы можем заглянуть под крышку и посмотреть, есть ли там защитная пленочка. Сам факт наличия этой пленки гарантирует нам, что этой зубной пастой никто не пользовался. Напротив ее отсутствие должно нас насторожить. Наличие этой защитной пленки несет для нас информацию, в равной степени как и ее отсутствие.

Если бы паста продавалась без защитной пленки (как это было ранее), мы могли бы только надеется на то, что нашей зубной пастой никто не воспользовался. То есть защитная пленка - это механизм, который сохраняет для нас информацию о событии использования пасты.

В JavaScript'e для сохранения информации нам не нужна защитная пленка. Мы можем просто создать переменную!

Как правило, для флагов используют переменную типа Boolean

```
var flag = false;
```

Изначально флаг может быть в одном состоянии, например флаг опущен, затем если произошло важное для нас событие состояние флага может измениться на флаг поднят. Флаг может быть изначально поднят, а если произошло что-то неудобное для нас, то мы его опустим.

Возьмем к примеру такую задачу. Есть массив, нужно узнать есть ли в нем пятерки.

Обычно учащиеся первым делом пишут вот такой вот код

```
var flag = false;

var arr = [8, 4, 5, 33, 5, 2, 11];

for (i=0; i<arr.length; i++) {
  if (arr[i]===5) {
    console.log('Ура! В массиве есть пятерка');
  }
}
```

Проблема этого кода в том, что если нам в массиве встретится две пятерки 'В массиве есть пятерка' выведется несколько раз. Если же пятерок не будет в массиве, для нас вообще не будет никакого сообщения.

```
var flag = false;

var arr = [8,4,5,33,5,2,11];

for (i=0;i<arr.length;i++) {
    if (arr[i]===5) {
        flag = true;
    }
}

if (flag) {
    console.log('Ура! В массиве есть пятерка');
} else {
    console.log('Пятерки в массиве не было');
}
```

Практика:

1. Вывести слова разделенные запятой
2. "Опавшие листья". Есть массив чисел. Нужно вывести числа в квадратах. Изначально квадраты зеленые, но если встретилось число меньше 8, квадраты становятся желтыми. Если встретилось отрицательное число, квадраты становятся серыми.
3. "Джекпот". Есть блоки с числами. Проверить, есть ли у нас в блоках три семерки подряд.
4. Есть блоки с числами. Проверить есть ли блоки, в которых семерка стоит сразу за пятеркой.

#C0FFEE Оператор деления по модулю

Вспомним из школьного курса такое понятие как остаток от деления. Например найдем остаток от деления 7 на 3. Для этого рассмотрим как мы можем представить семерку в ввиде троек.

$$7 = 3 * 2 + 1$$

то есть семерка делится на три два раза и единица остается в остатке. То есть остаток от деления 7 на 3 равен 1.

К счастью в JavaScript'e есть операция нахождения остатка от деления. Это %

$$7 \% 3 == 1 \quad 8 \% 3 == 2$$

Четные числа делятся на два без остатка, что означает, что остаток от деления на 2 любого четного числа будет равен нулю.

$$10 \% 2 == 0$$

Если мы хотим получить последнюю цифру в десятичной записи числа, нам нужно взять остаток от деления на 10. Происходит это потому, что любое положительное число без последней цифры заканчивается на ноль, а следовательно делится на 10 без остатка. Любое положительное число меньше десяти на 10 не делится, следовательно является остатком от деления на 10.

Например:

$$247 \% 10 == (240 \% 10) + (7 \% 10) == 0 + (7 \% 10) == 7$$

Практика:

1. Есть массив. Посчитать сколько в нем четных элементов.
2. Вывести предпоследнюю цифру числа.
3. Вывести цифры числа в обратном порядке

#COFFEE Повторения событий

Использование циклов призвано решить задачу повторения действий, но в JavaScript действия кроме программного кода может генерировать и пользователь.

Рассмотрим следующий код:

```
<div class="block" onclick="alert('Hi!!')">
</div>
```

При клике на блок, у нас начнет выводиться всплывающее окно.

alert является встроенной функцией, давай попробуем создать свою

```
<script>
function boom() {
    alert('BOOM!');
}
</script>

<div class="block" onclick="boom()">
</div>
```

Можно установить обработчик другим способом

```
<script>

var obj = document.querySelector('.block');

obj.onclick = boom;

function boom() {
    alert('BOOM!');
}
</script>

<div class="block">
</div>
```

Давайте попробуем подсчитать количество кликов

```
<script>
  var i = 0;
  function count() {
    i++;
    console.log(i);
  }
</script>

<div class="block" onclick="count()">
</div>
```

Практика:

1. Сделайте блок. Внутри него выведите число 0. При клике на блок увеличивайте число внутри него на единицу.
2. Есть блок. Если кликнуть на него три раза, блок меняет цвет.
3. Есть блок и массив картинок. При старте выводим на фоне "нулевую" картинку. При последующих кликах меняем фоновое изображение внутри блока на последующие фоновые картинки из массива.
4. Есть блоки двух цветов. Делаем три кнопки. При клике на первую остаются блоки только первого цвета. При клике на вторую остаются блоки только второго цвета. При клике на третью все блоки возвращаются на свои места.

Цикл while

Получим вечный цикл

```
var i=0
while (i<10) {

}
```

Чтобы избавиться от внешнего цикла добавим увеличение i во внутрь

```
var i=0
while (i<10) {
  i++; //i=i+1
}

console.log(i); // ?
```

```
var i=0
while (i<10) {
  console.log(i);
  i++; //i=i+1
}

console.log(i); // ?
```

```
var i=0;
while(i<5) {
  console.log(i);
  i++;
}
```

Практика:

1. Ввести имя с клавиатуры(prompt) и вывести его на экран (alert)
2. Ввести два числа вывести сумму
3. Ввести два числа, вывести больше из двух
4. Проверить делится ли число на 3

#Циклы с неизвестным количеством итераций

Пусть мы хотим, чтобы пользователь вводил числа до тех пор пока не введет ноль.

```
while(s=+prompt('Введите число')) {  
  console.log(s);  
}
```

Почему условие в крулых скобках while будет работать? Выражение с присвоением возвращает ту величину, которую мы присваиваем. То есть нам будет возвращаться то число, которые мы ввели в prompt. После чего это число будет приводиться к типу Boolean. Все числа отличные от нуля будут приводиться к true, и только ноль будет приводиться к false.

Подсчитаем количество введенных чисел

```
var i=0;  
while(s=+prompt('Введите число')) {  
  console.log(s);  
  i=i+1;  
}  
console.log(i);
```

1. Проверить есть ли единицы в записи числа
2. Проверить является ли число степенью двойки
3. Посчитать НОД двух чисел
4. Посчитать количество цифр в числе
5. Вывести цифры числа задом наперед
6. Игра угадай число. Компьютер "загадывает" число от 0 до 100. И спрашивает у пользователя числа, пока пользователь не угадает. После ввода числа, компьютер сообщает больше или меньше загаданное число, числа пользователя. Если пользователь угадал число, компьютер его поздравляет

Объекты

Создание

Допустим нам нужно хранить информацию о товаре. У каждого товара есть такие свойства как название товара, описание, цена и фотография товара.

Синтаксически такое описание товара будет выглядеть вот так

```
var product = {  
  name: 'бычки в томате',  
  description: 'черноморские бычки бережно приготовленные тетей Мусей в томате херсонских помидор',  
  price: 'бесценно',  
  image: 'tomato_bichok.jpg'  
}
```

Допустим мы хотим вывести в консоль название товара в product. Для этого нам нужно обратиться к свойству name объекта product. Делается это через точку (смотри пример внизу)

```
console.log(product.name);
```

Свойства

Допустим мы хотим поменять цену нашего продукта

```
product.price = 200; //тип свойства price при этом меняется со string на int
```

Мы могли бы пойти другим путем в создании объекта

```
var midii = {}; // создаем пустой объект  
  
midii.name = 'Мидии';  
midii.price = 100;  
midii.image = 'luzanovka_midii.png';
```

Методы

Допустим мы хотим вывести имя, цену и картинку на экран. Нам понадобится функция, но поскольку функция в JavaScript'e одновременно может быть еще и переменной, то добавим ее внутрь нашего объекта.

```
midii.getInfo = function() {  
    return this.name+ ' '+this.price+ ' '+this.image;  
}  
  
//вызовем наш метод  
midii.getInfo();
```

Наверняка у Вас возник вопрос, а что такое this? Чтобы ответить на него давайте мысленно представим что происходит. Мы вызвали функцию(метод) getInfo изнутри объекта midii. Метод возвращает нам имя, цену и картинку. Но имя, цену и картинку чего? Правильно нам нужны имя, цена и картинка ЭТОГО объекта из которого была вызвана функция getInfo. Вот this и является указанием на то, что мы берем свойства с ЭТОГО объекта.

Тренировочные задачи

В программировании, в начале пути многие сталкиваются с ситуацией, когда полностью не понимают, как работает программа, сколько бы они на нее не смотрели. Как правило, это происходит из-за того, что люди пытаются понять программу по ее внешнему виду. В то же время программа - это процесс, который довольно сложно осознать со стороны, если Вы не решали аналогичную задачу до этого.

Вместо того, чтобы пытаться понять структуру потоков программы со стороны нужно погрузиться в поток и двигаться в нем. Следующие задачи призваны оттренировать навык прогонять программу в голове. С улучшением этого навыка Вам будет также намного проще генерировать решения новых задач, так как Вы сможете предварительно "запускать" свои идеи в голове.

Во всех задачах Вам нужно предсказать их результат. В качестве проверки можете запустить их в браузере. Если понимания нет даже после этого - всегда можно воспользоваться пошаговой прогонкой в дебагере.

```
var x = 7;

x = x + 1

console.log(x);
```

```
var i = 7;

if (i == 7) {
    console.log('Hello');
    i++;
}

if (i == 8) {
    console.log('World');
}
```

```
var n = 7;
var limit = n-2;
for (i=0;i<n;i++) {
    if (i==limit) {
        console.log('Bingo!');
    }
}
```

```
var n=101;

for (i=0;i<n;i++) {

}
if (i==n) {
    console.log('Bingo!');
}
```

```
n=10;
for (i=0;i<n;i++) {
    if (i%4==0) {
        console.log('Bingo!');
    }

    if (i%7==0) {
        console.log('Bongo!');
    }
}
```

```
arr = [5,2,10,7,1];

for (i=0;i<arr.length;i++) {
    if (arr[i]===7) {
        console.log('Bingo');
    } else if (arr[i]===2) {
        console.log('Bongo');
    }
}
```

```
var b=9;
var n=10;
for (i=0;i<n;i++) {
    if (i==b) {
        console.log('Bingo');
    }
    b--;
}
```

Контрольная работа

Тут собраны различные примеры для контрольных работ

1. Есть массив чисел. Подсчитать сколько в нем чисел больше 5
2. Есть массив чисел. Подсчитать сумму всех чисел в нем, которые делятся на семь
3. Проверить является ли число степенью двойки
4. Есть блок. Сделать так, чтобы при клике его фоновый цвет поочередно менялся с первого на второй, со второго на третий, с третьего на первый и так по кругу.

Глава II - Основы

Массивы

```
var arr = [];
```

```
var fruits = ["Яблоко", "Апельсин", "Слива"];
```

```
alert( fruits[0] ); // Яблоко  
alert( fruits[1] ); // Апельсин  
alert( fruits[2] ); // Слива
```

Можно добавить еще один элемент

```
fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

В массиве может храниться любое число элементов любого типа. В том числе, строки, числа, объекты, вот например:

```
// микс значений  
var arr = [ 1, 'Имя', { name: 'Петя' }, true ];
```

// получить объект из массива и тут же -- его свойство alert(arr[2].name); // Петя

Вывод массива на экран

```
var arr = [12, 56, 3, 90, 34];  
for(i=0; i<arr.length; i++){  
    document.write(arr[i]);  
}
```

push и pop

```
var fruits = ["Яблоко", "Апельсин", "Груша"];  
  
alert( fruits.pop() ); // удалили "Груша"  
  
alert( fruits ); // Яблоко, Апельсин
```

```
var fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша
```

shift unshift

Поиск элемента в массиве

```
var a = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5];
alert(a.indexOf(3)); // 2
alert(a.lastIndexOf(3)); // 7
```

Удаление элементов из массива**

```
var a = [0, 1, 2];
alert(a.length); // 3
a[5] = 5;
alert(a.length); // 6
delete a[5];
alert(a.length); // 6: при удалении элементов length не изменяется.
a.length = 1;
alert([0 in a, 1 in a, 2 in a]); // true,false,false: элементы с индексами 1 и 2 удалены
```

splice

```
var a = [0, 1, 2, 3, 4, 5, 6];
// Удалим три элемента, начиная со второго(не забывает, что счет идет с нулевого элемента)
a.splice(2, 3);
alert(a); // 0,1,5,6
// Добавим два элемента перед последним
a.splice(-1, 0, 7, 8);
alert(a); // 0,1,5,7,8,6
// Заменяем второй и третий элементы на три строки
a.splice(1, 2, 'a', 'b', 'c');
alert(a); // 0,a,b,c,7,8,6
```

Практика:

1. Подсчет суммы элементов массива
2. Создать массив с названиями картинок. Вывести галерею картинок на экран
3. Найти максимальный элемент

4. Найти второе по величине число после максимума
5. Посчитать количество нулей в массиве
6. Проверить есть ли в массиве две единицы
7. «Курсор» указывает на нулевой элемент массива. Если в элементе массива j стоит число k , то курсор смещает на k позиций вправо (позиция $j+k$). Нужно узнать сколько будет таких смещений до того момента, когда курсор вылетит за границу.
8. *Даны номиналы монет, определить можно ли с их помощью выдать заданную сумму?
9. Найти простые числа
10. Найти самое редкое число в массиве
11. Задача на стек – пользователь вводит числа. Нужно вывести их в обратном порядке
12. Проверить правильность скобочной структуры

Добавляем DOM

document.write

Если мы хотим дописать html-код в body, можно воспользоваться методом document.write

```
<script>
  document.write('<div id="block"></div>');
</script>
//добавляет данный код в документ
```

document.getElementById()

document.getElementById - позволяет получить DOM-объект по его id

```
<div id="block">
</div>

<script>
var obj = document.getElementById("block");
</script>
```

Изменение встроенных CSS-свойств

Допустим у нас есть синий блок, но с помощью Javascript мы сделаем его красным

Для этого будем использовать свойство style DOM-объекта

```
<div id="block" style="width:100px;height:100px;background:black">
</div>

<script>
var obj = document.getElementById("block");
obj.style.backgroundColor='red';
</script>
```

Считывание значений у input'a

Допустим у нас есть input, и мы хотим считать оттуда значение.

```
<input type="text" id="info">
```

```
var obj = document.getElementById("info");  
console.log(obj.value); // выведет значение содержимого
```

innerHTML

Свойство innerHTML DOM объекта обращается к внутреннему содержимому тега, которое находится между открывающим и закрывающим тегами.

```
var obj = document.getElementById("block");  
obj.innerHTML = 'Этот текст будет внутри блока'
```

Если у нас внутри блока находится текст, то вернется его содержимое. Например есть вот такой HTML

```
<div class="outer">  
  <div class="inner">  
  </div>  
</div>
```

```
var block = document.querySelector('.outer');  
  
var str = block.innerHTML  
  
console.log(str); // <div class="inner"></div>
```

Работа с классами

Пример ниже выводит имя класса у блока с id "block1"

```
<div class="bad_block" id="block1">  
</div>  
  
<script>  
  var obj = document.getElementById("block1");  
  console.log(obj.className);  
</script>
```

Практика:

1. Есть два класса - первый окрашивает блок в синий цвет, другой в красный. Есть блок, окрашенный с помощью "синего" класса. Заменить его класс на "красный"
2. Есть четыре блока с одним классом. При клике на блок добавляем к нему класс, который добавляет у него границу.
3. Вывести картинку с помощью `document.write`
4. Вывести прямоугольник и поменять его цвет с помощью JavaScript
5. Вывести гистограмму
6. Вывести шахматную доску
7. Есть массив с названиями картинок, вывести галерею.
8. Генерируем блок произвольной ширины. Используем функцию `Math.rand\(\);`
9. Есть массив с цветами и массив с радиусами. Нужно вывести круги в произвольных местах экрана

Функции

Описание

```
function sayHi() {  
  console.log('Hi');  
}
```

Вызов

```
sayHi();
```

Передача параметров

```
function sayNum(x) {  
  console.log(x); //выдаст 5  
}  
sayNum(5);
```

Возвращение значений

```
function sayNum(x) {  
  return 2*x;  
}  
console.log(sayNum(5)); // выдаст 10
```

После первого return функция завершается

```
function sayNum(x) {  
  return 2*x;  
  return 3*x  
}  
console.log(sayNum(5)); // ?
```

Function Declaration

```
Sq(7); // ?

function Sq(x) {
  return x*x;
}

Sq(5); // ?
```

Function Expression

```
Sq(7); //

var Sq = function(x) {
  return x*x;
}

Sq(5);

var Sq2 = Sq;

console.log(Sq2(9)); //81
```

В данном примере мы видим, что функция может быть переменной

Создание через конструктор new Function

```
var adder = new Function('a', 'b', 'return a + b');

// now call the function
adder(2, 6);
```

Не самый распространенный способ создания функции, но тем не менее отлично показывающий, что функция является также еще и объектом! И этим свойством мы будем пользоваться в дальнейшем.

Локальные, глобальные переменные

```
<script>
var x=5; // глобальная

function sayNum() {
    var x=7; //локальная
    console.log(x); //7
}

sayNum();
console.log(x); // 5
</script>
```

Функция с вызовом на месте

```
(function() {
    console.log('Я существую');
})();
```

Такая конструкция позволяет выполнять код, не создавая новых глобальных переменных.

Например

```
var x = 5;

(function() {
    var x = 7;
    var a = 10;
})();

console.log(x); // 5
console.log(a); // Undefined
```

Формальные и фактические параметры

```
//формальные
function summ(a,b) {
    return a+b;
}

//фактические
summ(5,7); //12

summ(5); //NaN
//В недостающий параметр передается Undefined

summ(5,7,11); //12
```

Массив arguments

```
summ(5,7,11); //23

function summ(a,b) {
  var num =arguments.length;

  var sum=0;
  for (var i=0;i<num;i++){
    sum+=arguments[i];
  }
  return sum;
}
```

Практика:

1. Написать функцию, которая считает сумму двух чисел
2. Пишем функцию, которая считает площадь прямоугольника
3. Пишем функцию, которая считает расстояние между двумя точками (понадобится для agar.io)
4. Есть длины двух сторон прямоугольника. Посчитать площадь прямоугольника и его периметр (тренируем возвращение нескольких значений)
5. Сделать функцию, которая разбивает число на два множителя. Множители должны возвращаться из функции.
6. Изменение цвета DIV'a в зависимости от того какое число было введено
7. НОД
8. Сделать функцию, которая на вход получает массив координат и функцию для отрисовки фигур. Попробовать передать туда функцию, которая бы рисовала квадраты, а потом функцию, которая бы рисовала круги
9. Сделать функцию, которая считает количество своих вызовов (если пройдены замыкания)

Ассоциативные массивы

```
var MyObject = new Number();  
MyObject["id"] = 5;  
MyObject["name"] = "SampleName";
```

Если мы хотим пройтись по свойствам объекта то можно воспользоваться конструкцией for in:

```
for (key in MyObject)  
{  
    //key - ключ или названия свойства  
    //MyObject[key] - содержание записи по ключу  
    console.log(MyObject[key]);  
}
```

```
var arr={};  
  
arr["Олег"]='HR';  
arr["Леонид"]='Маркетолог';
```

Практика:

1. Массив имен профессий. Найти сколько есть сантехников
2. Вывести кто является сантехником
3. Массив логинов, паролей — проверить залогинен ли пользователь?
4. Посчитать количество людей по профессиям
5. Найти самую распространенную профессию
6. Вывести страны через ComboBox

Задания для контрольных

1. Есть массив чисел. Посчитать сколько в массиве семерок
2. Сделать функцию, которая на вход получает объект, содержащий адрес картинки, ее ширину, высоту, позицию(смещение от верха, смещение от левого края). Функция должна отобразить данную картинку.
3. Есть массив картинок (адрес). Одна из картинок повторяется несколько раз. Вывести галерею картинок, повторяющуюся картинку обвести в рамку(граница толщиной 10px).

Глава III - События

Способы установки обработчика

<http://javascript.ru/tutorial/events/intro>

Три способа задания обработчиков событий

1) Через HTML-атрибут

```
<input type="button" value="Нажми здесь" onclick="alert('Работает');" >
```

2) Через свойство DOM-объекта

Есть input

```
<input id="myButton" type="button" value="Нажми меня"/>
```

К нему добавляем обработчик

```
document.getElementById('myButton').onclick = function() {  
    alert('Спасибо')  
}
```

либо вот такой вариант

```
function doSomething() {  
    alert('Хватит кликать!');  
}  
  
//получаем DOM-объект  
input = document.getElementById('myButton');  
  
input.onclick=doSomething;
```

3) addEventListener

```
<input type="submit" id="buttonObj">
```

```
//код функции обработчика
function doSomething() {
    alert('Хватит кликать!');
}

//получаем DOM-объект
var buttonObj = document.getElementById('buttonObj');

buttonObj.addEventListener( "click" ,doSomething, false);
// ставим обработчик
```

Обратите внимание, что во втором и третьем случаях мы передаем функцию doSomething, а не ее вызов doSomething(). В третьем случае от названия события отбрасывается приставка on

Практика:

1. Есть три DIV'а . При клике на div выводится всплывающее сообщение. На каждом из DIV'ов используется один из способов задания обработчика
2. Блок. При клике на него он меняет цвет. При повторном клике цвет возвращается
3. Есть два div'а. Если кликнул на первый и на второй – выводится сообщение, что все хорошо
4. Есть массив логинов и форма ввода логина. При вводе логина пользователем, проверяем свободен ли данный логин или нет
5. При клике на блок увеличиваем счет
6. Есть много кружков. При клике на кружок, его счет увеличивается
7. Есть большой серый блок и маленькие цветные DIV'ы. При клике на маленький DIV, большой DIV выделяется выбранным цветом .
8. Выводим круги в случайных позициях со случайными числами внутри. При клике на круг число в нем уменьшается. Когда счет доходит до нуля, круг исчезает — счет увеличивается.
9. Сделать Cookie Clicker <http://orteil.dashnet.org/cookieclicker/>

Использование this в обработчиках

```
function func() {  
    this.style.backgroundColor = 'red';  
}  
  
var obj1 = document.getElementById('obj1');  
var obj2 = document.getElementById('obj2');  
  
obj1.onclick = func;  
obj2.onclick = func;
```

Практика:

1. Есть кружки с числами. При клике на кружок, число внутри должно уменьшиться на единицу. Использовать один обработчик для всех кружков.
2. Сгенерировать блоки. Установить на них всех(в цикле) один обработчик, который при клике на блок меняет его цвет.
3. Сделать предыдущее задание, чтобы активным был только один блок.
4. Есть четыре блока. При клике на один из них, блок становится «активным»(меняет цвет), остальные «деактивируются»
5. Есть меню с вложенными подпунктами. Сделать, чтобы при клике на пункт разворачивались его подпункты

Объект event

Когда происходит событие, браузер генерирует объект события(объект event) и передает его в обработчик ссылку на него. Для того, чтобы начать работать с этим объектом нужно указать его в параметрах обработчика.

Объект event позволяет получить дополнительную информацию о произошедшем событии:

event.type - сообщит нам тип произошедшего события: click, change, keydown

event.currentTarget - возвращает ссылку на объект, на котором в данный момент произошло событие.

В зависимости от того какое событие произошло в event'e могут быть какие-то специальные свойства. Например событие onclick передает в этот объект свойства event.clientX и event.clientY - координаты клика мыши относительно левого верхнего угла объекта, на который установлен обработчик.

Пример ниже будет выводить все рассмотренные свойства при клике на любую область экрана.

```
<div style="width:100%;height:100%;background-color:mediumseagreen" id="elem">
</div>

<script>
  elem.onclick = function(event) {
    // вывести тип события, элемент и координаты клика
    alert(event.type + " на " + event.currentTarget);
    alert(event.clientX + ":" + event.clientY);
  }
</script>
```

Подробнее о различиях в объекте event для разных событий в следующем разделе

Практика:

1. При клике на экране выводим координаты клика
2. При клике перемещаем круг на заданную позицию(используем CSS и transition)
3. Пользователь нажимает клавишу. Вывести код клавиши на экран
4. Есть массив логинов и форма ввода логина. При вводе логина пользователем, проверяем свободен ли данный логин или нет
5. Делаем игру с управлением игроком стрелочками

Стандартные события в JavaScript

onload - срабатывает, когда объект загрузился.

Если поставить событие onload на картинку, то onload сработает, когда картинка загружена.

Данное событие можно использовать, для того, чтобы получить возможность размещать JavaScript до body.

```
<script>
function func() {
    alert('Страница загружена');
}
</script>

<body onload="func()">
</body>
```

onscroll - срабатывает при скроллинге внутри элемента. Если нас интересует скроллинг внутри страницы, то это будет body

```
document.body.onscroll = scroller;

function scroller() {
    alert('Ура, я умею скроллить!');
}
```

Смещение скроллинга относительно верха сайта

Чтобы получить текущее смещение экрана от верха сайта используем свойство **window.pageYOffset**

Получение позиции элемента относительно верха сайта

```
var bodyRect = document.body.getBoundingClientRect(),
    elemRect = element.getBoundingClientRect(),
    offset    = elemRect.top - bodyRect.top;
```

Размеры окна

```
var w = window.innerWidth;
var h = window.innerHeight;
```

onkeypress, onkeydown, onkeyup - события для работы с клавиатурой

onkeydown - срабатывает, когда нажата клавиша.

onkeyup - когда она отпущена

onkeypress - когда напечатан символ

Допустим мы хотим перехватить нажатие клавиши "стрелочка вверх". Тогда можно воспользоваться следующим кодом.

```
<html>
<head>
<script>
function func(event) {
    if (event.keyCode=='38') {
        console.log('Нажата клавиша вверх');
    }
}

window.addEventListener('keydown', func, false);
</script>
</head>
<body>
</body>
</html>
```

Получение нажатого символа по коду

```
String.fromCharCode(evt.keyCode)
```

onchange - происходит, когда меняется значение элемента.

Например, когда у нас есть select-option элемент и мы выбираем у него какое-то другое значение

```
<select class="choose">
    <option value="Одесса">Одесса
    <option value="Амстердам">Амстердам
    <option value="Сан-Франциско">Сан-Франциско
</select>
<script>
var selectObj = document.querySelector('.choose');

selectObj.onchange = function() {
    console.log(this.value);
}
</script>
```

Если у нас есть input с type="file", то при выборе файла также сработает onchange

```
<input type="file" class="our_file">

<script>

var obj = document.querySelector('.our_file');

obj.onchange = function() {
    console.log('Вы выбрали файл');
}

</script>
```

Но при печати в input type="text" событие onchange вызываться не будет.

onresize - происходит при изменении размеров окна.

```
window.onresize = function() {
    console.log('Размеры окна поменялись');
}
```

onmousedown, onmouseup - события, которые возникают при нажатии клавиш мыши.

onmousedown - срабатывает, если мы нажали клавишу мыши(необязательно левую) над объектом.

onmouseup - срабатывает, если клавиша мыши была отпущена над объектом

В отличии от этих двух событий onclick срабатывает, когда клавиша мыши была нажата и отпущена над объектом

onmouseover,onmousemove,onmouseout

onmouseover - срабатывает, когда курсор мыши появляется над объектом

onmousemove - происходит, когда мы двигаем курсором мыши над объектом.

onmouseout - срабатывает, когда курсор мыши уходит за пределы объекта.

Генерация "искусственных" кликов на кнопке

Для того, чтобы сгенерировать клик нужно получить DOM-объект и вызвать от него функцию click()

```
var obj = document.querySelector('.button');

obj.click();
```

Практика:

1. Есть квадрат красного цвета. При загрузке документа, по событию onload меняем его на зеленый.
2. Сделать сайт высотой в несколько экранов. При скроллинге мы должны показывать пользователю счетчик - смещение от верха.
3. Делаем сайт со скроллингом. Когда пользователь доскролил до определенного момента срабатывает событие.
4. В предыдущем задании добиться, чтобы всплывающее окно срабатывало один раз.
5. По середине страницы есть блок. Когда мы до него доскроливаем он увеличивается в размерах сначала по высоте, потом по ширине.
6. Делаем сайт с шапкой и скроллингом. Прокручиваем сайт вниз. При скроллинге вверх должна появляться шапка сайта, при скроллинге вниз она должна скрываться(пример статей на сайте Medium)
7. Есть лента постов. При прокрутке, если шапка поста спряталась, переносим ее автоматически вверх (пример лента Instagram)
8. Есть форма с двумя полями: для ввода логина и для ввода пароля. Сделать скрипт, который поздравляет пользователя alert'ом если данные были введены правильно.
9. Есть поле для ввода текста. Под ним показывается число символов, которые пользователь может ввести. С каждым введенным символом число убывает на единицу.
10. Пользователь вводит в input имя, после чего input очищается, и пользователь может ввести имя еще раз и т.д. Под input'ом показываем введенные имена и число - количество раз, которое данное имя было введено.
11. Есть форма для выбора и отправки файла. Необходимо ее скрыть и заменить на кнопку "ЗАГРУЗИТЬ". При нажатии на кнопку загрузить должно появляться окно выбора файла. После выбора файла происходит его автоматическая отправка из формы.
12. Создать игру, в которой объект управляется клавиатурой. На экране появляются "полезные предметы", которые объект должен "собрать" (приблизится к ним достаточно близко).
13. 2048

14. Пример интерфейса со скролингом <http://codepen.io/tholman/pen/XJjvGO>
<http://www.framescollection.com/tunnelrats.html>
1. Реализовать интерфейс переключения страниц с помощью скроллинга как на приведенном сайте <https://www.aristidebenoist.com/>
 2. Пример с заполнением формы <http://codepen.io/zenu/pen/JKgEWZ>

Всплытие событий

Рассмотрим следующую ситуацию

```
<div onclick="alert('div');" style="width:100px;height:100px; background:cornflowerblue">  
  <button onclick="alert('button')">  
    Не нажимать!  
  </button>  
</div>
```

Если мы кликаем на кнопку, то формально мы кликаем и на div. Где же определить событие onclick?

В JavaScript эта проблема решается следующим образом:

1. Событие onclick происходит на кнопке
2. Потом оно переходит ("всплывает") к div'у
3. Событие переходит на внешние элементы. Если их нет, то к body.

В нашем примере выше сработают оба обработчика. И мы получим сначала 'button', а потом 'div'

event.target

Как отличить "оригинальное" событие от "всплывшего"? В объекте event есть свойство event.target, которое хранит ссылку на объект, в котором произошло событие изначально.

event.stopPropagation()

Если же мы хотим приостановить всплытие события то нам необходимо вызвать метод stopPropagation() из объекта event.

Практика:

1. Сделать блок и сделать вложенный в него блок. Поставить на оба блока обработчики клика. Поэкспериментировать с кликами.
2. Есть четыре квадрата, при клике на один из них, он становится активным(то есть меняют цвет), остальные деактивируются
3. Есть разноцветные квадраты. При клике на квадрат нужно вывести его цвет (через делегирование)

4. Внутри квадратов слова. При клике на div – выводим внутреннее слово
5. Есть круги с нулями. При клике на круг счет расчет. Реализовать через делегирование.
6. Есть набор объектов (кружочки, квадратики). При клике мы делаем объект «активным», при этом с остальных элементов «фокус» убирается

Делегирование событий

Всплытие событий и свойство `event.target` дают нам возможность воспользоваться техникой делегирования событий

Допустим мы делаем редактор графических объектов. Мы хотим, чтобы при клике на объект выводился его цвет.

Что мы делали раньше?

Делали обработчик под каждый объект. Затем каждому объекту его устанавливали.

После изучения `this`, мы смогли сделать универсальный обработчик. Но приходилось все-равно его добавлять к каждому объекту.

В случае с делегированием мы еще можно сократить этот процесс.

1. Делаем внешний блок. Помещаем все объекты внутрь него.
2. Ставим обработчик на внешний блок.
3. За счет всплытия события, происходящие на вложенных объектах, будут распространяться на внешний блок и запускать обработчик.
4. В обработчике получаем объект `event`. В свойстве `event.target` хранится ссылка на объект, на котором произошло событие.

```
<div id="outer">
  <div id="inner">
  </div>
</div>

<script>
var outer = document.getElementById("outer");

outer.onclick=function(event) {
  if (event.target==this) {
    console.log('Родительский элемент');
  } else {
    console.log('Кликнули на ребенка');
  }
}

</script>
```

Практика:

1. Есть разноцветные квадраты. При клике на квадрат нужно вывести его цвет (через делегирование).
2. Пользователь кликает на блок, выводится цвет предыдущего блока.
3. Внутри квадратов слова. При клике на div – выводим внутреннее слово.
4. Есть круги с нулями. При клике на круг счет растет. Реализовать через делегирование.
5. Есть набор объектов (кружочки, квадратики). При клике мы делаем объект «активным», при этом с остальных элементов «фокус» убирается
6. Есть разные кружки, с числами внутри. При клике на кружок число внутри уменьшается на единицу. Когда число становится нулем кружок исчезает. Ведем подсчет сколько кружков и какого цвета было убрано.
7. Есть список. В каждом пункте есть подпункты, но изначально они скрыты. Реализовать сворачивание/разворачивание подпунктов при клике.
8. Сделать галерею картинок

Обработчики событий по умолчанию

У нас есть ссылка на другой сайт, и мы хотим, чтобы при клике на нее не происходил переход, а выводился alert. Для того, чтобы отменить действие по умолчанию, используем функцию `event.preventDefault()`;

```
<a href="http://google.com" onclick="func(event)">Кликайте сюда</a>

<script>

function func(event) {
    alert('Гугл не пройдет!');
    event.preventDefault();
}

</script>
```

Иногда используют `return false`, вместо `preventDefault`, но на самом деле это равнозначная замена.

```
<a href="http://google.com" onclick="return func(event)">Кликайте сюда</a>

<script>

function func(event) {
    alert('Гугл не пройдет!');
    return false;
}

</script>
```

`return false` в функции обработчике равносильно отмене всплытия события и отмене действия по умолчанию. То есть вот такой вариант.

```
<a href="http://google.com" onclick="return func(event)">Кликайте сюда</a>

<script>

function func(event) {
    alert('Гугл не пройдет!');
    event.stopPropagation();
    event.preventDefault();
}

</script>
```

Практика:

1. Кликаем на ссылку – всплывает alert
2. Есть картинка замка и ссылка. Пока замок "закрыт" ссылка также заблокирована. При клике на замок, он открывается. После этого ссылка становится кликабельной.
3. Делаем трехстраничный сайт, на одной странице. Сажаем свои обработчики на ссылки. При клике на ссылку мы показываем пользователю соответствующую страницу
4. Скрываем стандартный `<input type="file">` и делаем автоматическую отправку формы. Подробнее это означает, что нам нужно сделать красивую кнопку, при клике на нее нужно передать этот клик на скрытый input. После того как пользователь выберет файл, должна произойти автоматическая отправка формы.
5. Есть поле два ввода текста. Под ним показывается число символов, которые пользователь может ввести. С каждым введенным символом число убывает на единицу. Добиться, чтобы после того как пользователь "потратит" все доступные символы, ввод блокировался(кроме нажатия клавиши `backspace`).

Интервалы и задержки

Задержки

setTimeout – вызывает функцию через заданное количество миллисекунд

Следующий пример использует анонимную функцию

```
setTimeout(function(){ alert("Hello"); }, 3000);
```

Но мы можем передать в `setTimeout` просто название функции. Код снизу через три секунды после запуска выведет в консоль 12.

```
function func_sum(a,b) {  
    console.log(a+b);  
}  
  
setTimeout(func_sum, 3000, 5, 7);
```

А этот код через две секунды выведет 'Hello World!'

```
setTimeout('console.log("Hello World!");', 2000);
```

Интервалы

Если мы хотим, чтобы код функции запускался регулярно воспользуемся функцией **setInterval**

Общее ее описание представлено ниже. Как и в случае с `setTimeout` параметры после задержки необязательны(указаны в квадратных скобках).

```
var intervalID = setInterval(func, delay[, param1, param2, ...]);  
var intervalID = setInterval(code, delay);
```

Следующий код будет выводить 'Beep, beep' каждую секунду

```
function sputnik() {  
    console.log('Beep, beep');  
}  
  
setInterval(sputnik, 1000);
```

Прерывание вызова по таймеру

`setInterval`, как и `setTimeout` возвращает `id`, по которому можно прервать отложенное выполнение. `clearInterval` позволяет отменить `setInterval`, `clearTimeout` может отменить `setTimeout`.

Общий вид этих функций таков

```
clearInterval(IntervalId)

clearTimeout(TimerId)
```

Давайте рассмотрим пример, который останавливает вызов "по интервалу" при нажатии на кнопку.

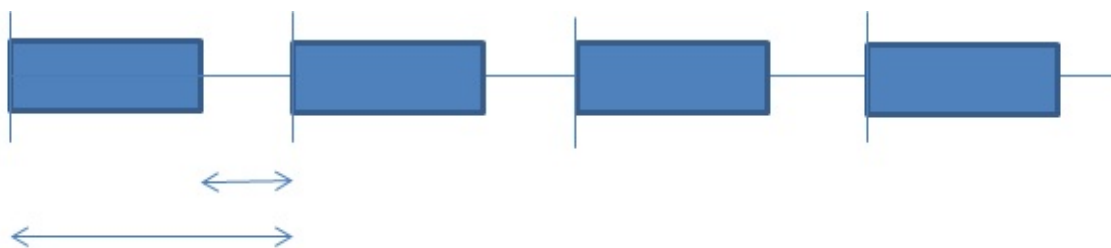
```
<button onclick="stopBeeping();">
Отправить в океан
</button>

<script>
function sputnik() {
    console.log('Beep, beep');
}

var timerId = setInterval(sputnik, 1000);

function stopBeeping() {
    clearInterval(timerId);
};
</script>
```

Очередь и наложение вызовов в `setInterval`



Поскольку на выполнение кода требуется время, то в реальности время между вызовами функции будет меньше чем указано в `setInterval`. Если же функции будут выполняться дольше чем идут вызовы, то вызовы будут накапливаться в очередь.

Рекурсивный вызов `setTimeout`

Для того, чтобы обеспечить гарантированное время между вызовами функции, используется следующая техника

```
var i=1;
var timer = setTimeout(function run() {
  console.log(i++);
  timer = setTimeout(run,2000);
},2000
)
```

По переменной timer мы можем прервать весь этот процесс.

Debouncing

Есть input. После того как мы перестали активно набирать, должен срабатывать alert. Техника состоит в том, что обработчик мы помещаем в таймер. При повторном вызове обработчика, таймер удаляется.

```
function debounce() {
  clearTimeout(timerId); //удаляем старый таймер, так как сгенерировалось //новое со
бытие
  timerId = setTimeout(fn_action,2000);
}
```

Практика:

1. Сделать блок, который мигает
2. Сделать блок, который начинает мигать при нажатии на клавишу
3. Добавить к предыдущему заданию кнопку stop, которая бы оставляла мигание
4. Нажимаем кнопку start несколько раз - проверяем работоспособность
5. "Бегущий элемент" - есть четыре блока. По очереди вспыхивает, то первый, то второй, то третий, то четвертый, потом опять первый
6. Делаем таймер обратного отсчета
7. Сделать автообновляющиеся часы. Использовать объект Date
8. Сделать слайдер. Переключение слайдера происходит автоматически, или по клику пользователя
9. "Тренировка быстрой печати". Пользователю показываются клавиши на экране, и он должен успеть их нажать, пока они не погасли. При каждом успешном нажатии пользователю начисляется бал.

Debouncing

Есть input. После того как мы перестали активно набирать, должен срабатывать alert.

```
function debounce() {  
    clearTimeout(timerId); //удаляем старый таймер, так как сгенерировалось //новое со  
    бытие  
    timerId = setTimeout(fn_action, 2000);  
}
```

Техника состоит в том, что обработчик мы помещаем в таймер. При повторном вызове обработчика, таймер удаляется.

Еще одна реализация

```
<textarea>Напишите тут что-нибудь...</textarea>  
<script>  
var textarea = document.querySelector("textarea");  
var timeout;  
textarea.addEventListener("keydown", function() {  
    clearTimeout(timeout);  
    timeout = setTimeout(function() {  
        console.log("Вы остановились.");  
    }, 500);  
});  
</script>
```

Практика:

1. Форма для ввода пароля с проверкой на правильность. Проверка должна срабатывать, только когда пользователь перестал набирать.
2. Всплывающая подсказка. Пока мы водим курсором по объекту ничего не происходит. Когда курсор неподвижен какое-то время - всплывает подсказка
3. Когда курсор останавливается, вокруг него растет круг.

Интересные материалы

Создание анимированного бургер меню и слайдера

<https://codepen.io/jorgecardoso/post/css-transitions-and-animations>

Задания для контрольной

1. Сделать блок, в котором есть ноль. При каждом втором клике на блок, к внутреннему содержимому должна добавляться единица.
2. Сделать блок-мигалку, в котором бы по очередно менялось бы три цвета
3. Есть блоки. При клике на блок его цвет перекрашивается. Выводить в консоль количество "перекрашенных" блоков. Реализовать через делегирование.

Глава IV - DOM и BOM

DOM, BOM и JS Объекты

<https://learn.javascript.ru/browser-environment>

Типы узлов в DOM

DOM-объект еще также называют узлом. Согласно спецификации есть 12-ть типов узлов. Нам понадобятся узлы всего лишь трех типов

`.nodeType==1` - html-элемент, то есть узел образованный тегом

`.nodeType==3` - текстовый узел

`.nodeType==8` - комментарии

```
<div id="block">
</div>

<script>
var type =document.getElementById("block").nodeType;
var nodeName =document.getElementById("block").nodeName;
console.log(type);// выведет 1
console.log(nodeName);// выведет DIV
console.log(tagName);// выведет DIV
</script>
```

Практика:

1. Есть блоки в перемешку с текстом - вывести `nodeType` всех элементов `body`

Навигация по DOM

<https://learn.javascript.ru/traversing-dom>

.childNodes - возвращает список всех узлов элемента

.children - возвращает список всех нетекстовых узлов

.parentNode - родительский узел

parentElement

.firstChild - первый узел-ребенок

.firstElementChild - первый нетекстовый узел

.lastChild - последний узел-ребенок

.lastElementChild - последний нетекстовый узел

.previousSibling - предыдущий соседний узел

.previousElementSibling - предыдущий нетекстовый сосед

.nextSibling **.nextElementSibling**

Конвертация из HTMLCollection в Array

```
var arr = Array.prototype.slice.call( htmlCollection )
```

```
var arr = [].slice.call(htmlCollection);
```

Поиск элементов в DOM

```
document.getElementsByTagName()
```

```
document.getElementsByClassName()
```

.querySelector - возвращает элемент, соответствующий указанному селектору

.querySelectorAll - возвращает список элементов, соответствующие указанному селектору

```
elementList = document.querySelectorAll(selectors);
```

```
<div class="block">
</div>
<div class="block">
</div>

<script>
    var first_block = document.querySelector("DIV"); //вернет ссылку на первый блок
</script>
```

Практика:

1. Есть блоки с одним и тем же классом. Четные блоки окрасить в один цвет, нечетные блоки окрасить в другой (использовать `querySelectorAll`).
2. Есть список блоков. Пронумеровать блоки в порядке возрастания (использовать `document.body.children`).
3. Есть родительский блок и есть дети. Дети одного цвета, родитель другого. При клике на область родительского блока, родительский блок и дети меняются цветами.
4. Есть четыре блока. При нажатии на клавиши перемещаем «выделение» на активном блоке
5. В предыдущем задании делаем блок верхнего уровня
6. Иерархическое меню. Перемещаемся по нему с помощью клавиш, при нажатии `enter` меню выпадает

Создание и модификация узлов

createElement и appendChild

createElement - создает узел по названию тега

appendChild - переносит узел внутрь другого узла(см пример)

```
var btn = document.createElement("BUTTON");           // Создаем элемент <button>
var t = document.createTextNode("CLICK ME");           // Создаем текстовый узел
btn.appendChild(t);                                     // Добавляем текст в кнопку <button>

document.body.appendChild(btn);                         // Добавляем <button> в <body>
```

Обратите внимание, что если мы работаем с уже созданными элементами, то они просто переносятся, без создания копии!

http://www.w3schools.com/jsref/met_document_createelement.asp

Наполнение элементов

Созданный с помощью createElement элемент можно наполнять через стандартные свойства DOM-объекта.

```
var banner = document.createElement('DIV');
banner.className = 'block';
banner.innerHTML = 'Летающий банер';
banner.style.position = 'fixed';
banner.style.top = '200px';
banner.style.left = '300px';
```

insertBefore

insertBefore, - добавляет элемент в список дочерних элементов родителя перед указанным элементом.


```
<div id="parentNode">
  <span id="childNode">foo bar</span>
</div>

<script>
//Создаем новый узел
var newNode = document.createElement("span");

//Добавляем перед childNode новый узел
parentNode.insertBefore(newNode,childNode);
</script>
```

removeChild - удаляет заданный узел из родительского узла

<https://developer.mozilla.org/ru/docs/Web/API/Node/removeChild>

Удаление элемента без указания его родителя

```
var node = document.getElementById("nested");
if (node.parentNode) {
  node.parentNode.removeChild(node);
}
```

cloneNode

<https://developer.mozilla.org/ru/docs/Web/API/Node/cloneNode>

```
<div id="outer">
  <div id="inner">
  </div>
</div>

<div id="outer2">
</div>
```

Если мы сделаем appendChild без клонирования, то получим следующую ситуацию

```
var inner = document.getElementById("inner");

var outer2 = document.getElementById("outer2");

outer2.appendChild(inner);
```

```
<div id="outer">
</div>

<div id="outer2">
  <div id="inner">
  </div>
</div>
```

Воспользуемся клонированием

```
var inner = document.getElementById("inner");
var clone = inner.cloneNode();

var outer2 = document.getElementById("outer2");

outer2.appendChild(clone);
```

```
<div id="outer">
  <div id="inner">
  </div>
</div>

<div id="outer2">
  <div id="inner">
  </div>
</div>
```

insertAdjacentHTML() <https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML>

Работа с атрибутами

setAttribute - устанавливает атрибут в элементе

<https://developer.mozilla.org/ru/docs/Web/API/Element/setAttribute>

Например, есть у нас кнопка

```
<button>Hello World</button>
```

с помощью javascript'a сделаем ее неактивной

```
var b = document.querySelector("button");

b.setAttribute("disabled", "disabled");
```

getAttribute() - возвращает значение указанного атрибута элемента. Если элемент не содержит данный атрибут, могут быть возвращены null или "" (пустая строка);

<https://developer.mozilla.org/ru/docs/Web/API/Element/getAttribute>

```
<input type="text" placeholder="здесь могла быть Ваша реклама">
```

```
var b = document.querySelector("input");  
  
var res = b.getAttribute("placeholder");  
  
console.log('и '+res); //и здесь могла быть Ваша реклама
```

hasAttribute - проверяет установлен ли атрибут. Возвращает true или false, в зависимости установлен наш атрибут в элементе или нет

```
<input type="checkbox" checked>
```

```
var b = document.querySelector("input");  
  
b.hasAttribute("checked");
```

removeAttribute - удаляет атрибут у элемента

```
<input type="checkbox" checked>
```

```
var b = document.querySelector("input");  
  
b.removeAttribute("checked");
```

Практика:

1. Есть кнопка. При нажатии на нее появляется круг. При клике на любой из созданных кругов, он удаляется.
2. Даем возможность пользователю добавлять новые элементы в выпадающее меню (ToDo List).
3. Игра. В произвольном месте экрана появляются кружки - нужно успеть на них кликнуть, чтобы заработать баллы
4. Есть кнопка. При нажатии на кнопку в документ добавляются кружки.

5. Сделать так, чтобы в предыдущем задании кружки перекрашивались.
6. Добавить в предыдущее задание вторую кнопку и блок. При нажатии на кнопку все перекрашенные круги перемещаются в блок.
7. Сделать телефонный справочник. Есть список людей и их номеров телефонов. Есть форма, которая позволяет добавлять новых людей.
8. Рядом с каждой записью добавить кнопку удаления.
9. Добавить возможность редактирования каждой записи.

Работа с формами

Получение значения из input'a

Допустим у нас есть поле для ввода текста

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

Чтобы получить значение того, что введено в input'e используем свойство `.value`

```
var obj = document.getElementById("name");  
  
console.log(obj.value);
```

Отправка формы

Допустим у нас есть следующая форма

```
<form action="#" method="GET">  
  <input type="text" name="login">  
  <input type="password" name="password">  
  <input type="submit">  
</form>
```

Мы можем отправить ее несколькими способами:

1. Отправка формы

```
var form = document.querySelector('form');  
  
form.submit();
```

1. Генерация клика на форме отправки

```
var submitBtn = document.querySelector('input[type=submit]');  
  
submitBtn.click();
```

События

`.focus` - при получении фокуса

```
var obj = document.getElementById("name");

obj.onfocus = function() {
    alert('Фокус здесь');
}
```

.blur - при потере фокуса

onchange

при изменении значения в SELECT'е и input type="file"

.checked

document.getElementById("myCheck").checked = true;

oninput

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
<input type="range" id="a" value="50">100
+<input type="number" id="b" value="50">
=<output name="x" for="a b"></output>
</form>
```

```
<form oninput="result.value=parseInt(a.value)+parseInt(b.value)">
    <input type="range" name="b" value="50" /> +
    <input type="number" name="a" value="10" /> =
    <output name="result"></output>
</form>
```

Дополнительные ссылки: Выразительный JavaScript про формы

<https://habrahabr.ru/post/245731/>

Проблемы со всплыванием focus, blur, change

http://www.quirksmode.org/blog/archives/2008/04/delegating_the.html

Практика:

1. Вводим данные в input. Набранный текст выводится рядом с input'ом
2. При вводе логина делаем автопроверку свободен ли он. Если логин свободен обводим input зеленой рамочкой.
3. Сделать всплывающие подсказки при вводе текста.
4. Сделать конвертор валют

5. Чекбоксы. Сделать кнопку, которая убирает выделение со всех чекбоксов
6. Сделать калькулятор

Работа с CSS

Получение значения свойства элемента

<https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle>

Для получения стилей элемента нам понадобится метод `getComputedStyle` объекта `window`. После чего мы получим объект со стилями элемента. Первым параметром этого метода должен быть DOM объект. Второй параметр предназначен для псевдоэлемента, если нам не нужен псевдоэлемент ставим `null`

```
var elem1 = document.getElementById("elemId");
var style = window.getComputedStyle(elem1, null);
```

Если мы все-таки хотим получить свойства из псевдоэлемента, тогда вызов будет выглядеть вот так.

```
var h3 = document.querySelector('h3');
result = getComputedStyle(h3, ':after').content;
```

После того как мы получили стили(`CSSStyleDeclaration`), можно вызвать от них метод `getPropertyValue`.

```
var elem = document.getElementById("elem-container");
var elemHeight = window.getComputedStyle(elem, null).getPropertyValue("height");
```

classList

<http://frontender.info/the-classlist-api/>

<https://developer.mozilla.org/ru/docs/Web/API/Element/classList>

Свойство `classList` позволяет более гибко модифицировать классы у элемента `.contains` проверяет, есть ли класс в нашем объекте

```
var obj = document.querySelector('.block');

obj.classList.contains('block') === true;

obj.classList.contains('bad_block') === false;
```

`.toggle` - добавляет класс, если его не было, убирает класс, если он был

Добавление правил

<https://davidwalsh.name/add-rules-stylesheets>

<https://developer.mozilla.org/ru/docs/Web/API/CSSStyleSheet/insertRule>

```
sheet.insertRule("body {margin:0px}");
```

CSS-переменные

<https://eager.io/blog/communicating-between-javascript-and-css-with-css-variables/>

Контрольная работа

Глава V -BOM

AJAX и JSON

Давайте рассмотрим JavaScript и HTML коды, а затем постараемся построчно их разобрать. Подразумевается, что JavaScript подключен к HTML коду.

```
function makeXHR()
{
    var xhr=new XMLHttpRequest(); //создание объекта для работы с асинхронными запроса
    ми

    xhr.onreadystatechange=function() {
        if (xhr.readyState==4 && xhr.status==200) {
            document.getElementById("myDiv").innerHTML=xhr.responseText;
        }
    }

    //формирование запроса
    xhr.open("GET", "ajax_info.txt", true);
    //отправка запроса
    xhr.send();
}
```

```
<div id="myDiv">
    <h2>Аjax поменяет этот текст</h2>
</div>
<button type="button" onclick="makeXHR()">Послать AJAX</button>
```

Для работы с AJAX нам понадобится объект от конструктора XMLHttpRequest. Любой браузер, поддерживающий AJAX содержит данный конструктор. Создаем мы такой объект строчкой

```
var xhr=new XMLHttpRequest();
```

Название объекта может быть любым, xhr выбрано просто, чтобы легче было запомнить (сокращение от Xml Http Request)

Далее пропустим пока что строчки для установки обработчика события. и перейдем сразу к формированию AJAX-запроса

Строка xhr.open подготавливает(но не отправляет) AJAX-запрос. Мы можем отправлять запросы двумя методами протокола http - GET и POST. В данном случае мы используем метод GET. Запрос пойдет к файлу ajax_info.txt, который находится в

той же папке, что и наш скрипт. Последний параметр `true`, говорит о том, что запрос будет асинхронным, то есть мы не будем приостанавливать выполнение скрипта, ожидая получения ответа от сервера

```
xhr.open("GET", "ajax_info.txt", true);
```

Ну и непосредственно отправка запроса на сервер

```
xhr.send();
```

Рано или поздно, если всё было в порядке, сервер пришлет нам свой ответ, и его нужно обработать. Когда данные поступают от сервера в браузере срабатывает событие `onreadystatechange`. Поэтому нам необходимо поставить на него обработчик

```
xhr.onreadystatechange=function() {  
    if (xhr.readyState==4 && xhr.status==200) {  
        document.getElementById("myDiv").innerHTML=xhr.responseText;  
    }  
}
```

Данные от сервера могут поступать порциями, затем будут проходить стадию подготовки. Каждый раз, при изменении состояния этого процесса будет вызываться событие `onreadystatechange`. Нас же интересует только тот момент, когда данные полностью загрузились и готовы для работы с ними. Это соответствует моменту когда `xhr.readyState == 4`. Строка `xhr.status==200` означает, что сервер, к которому мы обращались нормально отработал и сформировал свой ответ.

После того как данные получены, они находятся в поле `responseText` объекта `xhr`. В нашем примере мы загружаем их внутрь `div'a` с `id myDiv`.

Для того, чтобы протестировать работоспособность этого примера, скопируйте код, приведенный ниже, сохраните его в файле `index.html` и откройте этот файл в Mozilla Firefox (Chrome и Safari не работают с локальными файлами). Не забудьте создать файл `ajax_info.txt` - напишите там, например `Hello World!` Файл создавать лучше в Sublime или Notepad++, но не в Блокноте или WordPad, потому что могут возникнуть проблемы с кодировками, из-за которых ничего не будет работать.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    function makeXHR() {
      var xhr=new XMLHttpRequest(); //создание объекта для работы с асинхронными
запросами

      xhr.onreadystatechange=function() {
        if (xhr.readyState==4 && xhr.status==200) {
          document.getElementById("myDiv").innerHTML=xhr.responseText;
        }
      }
      //формирование запроса
      xhr.open("GET", "ajax_info.txt", true);
      //отправка запроса
      xhr.send();
    }
  </script>
</head>
<body>
  <div id="myDiv"><h2>Ajax поменяет этот текст</h2></div>
  <button type="button" onclick="makeXHR()">Послать AJAX</button>
</body>
</html>
```

Формат JSON

Как правило нам нужно получать от сервера более сложную информацию, чем просто строку. Если у нас есть разнородная информация, ее необходимо как-то структурировать. Как один из вариантов такого структурирования был придуман формат XML. Ему обязан своим названием объект XMLHttpRequest и сам протокол AJAX (Asynchronous JavaScript And XML). Но поскольку он довольно громоздкий, со временем он был заменён форматом JSON. Пример формата JSON можно посмотреть ниже.

Внимание! Для того, чтобы JSON парсился без ошибок, в нем не должно быть переносов строк

Хорошая подсветка для JSON - Monokai JSON+

```
{
  "firstName": "Остан",
  "lastName": "Бендер",
  "address": {
    "streetAddress": "Большая Арнаутская 23а",
    "city": "Одесса",
    "postalCode": 65000
  },
  "phoneNumbers": [
    "+380 (48) 728-10-68",
    "+380 (48) 777-02-42"
  ]
}
```

Если присмотреться, то JSON очень напоминает способ создания объекта с помощью литеральной нотации. Из отличий - мы должны везде использовать двойные кавычки: для названий свойств это обязательно, для значений исключение делается только для значений-чисел (посмотрите на `postalCode` в примере).

Функции внутри JSON'a не описываются. Если нам нужен объект, мы также можем использовать фигурные скобки, если нам нужен массив, то используем квадратные скобки.

JSON.parse

JSON вряд ли бы стал сильно популярным, если бы не одно его свойство. Строку из JSON очень просто перевести в JavaScript объект. Для этого используется метод `parse` объекта JSON. Выглядит это преобразование примерно вот так.

```
var obj = JSON.parse(json_string);
```

Заменяем содержимое нашего файла `ajax_info.txt` на JSON следующего вида

```
{"name": "Остан", "lastname": "Бендер"}
```

Если мы оставим строку с `xhr.responseText` без изменений (код ниже)

```
document.getElementById("myDiv").innerHTML=xhr.responseText;
```

То при нажатии на кнопку мы увидим наш JSON-код. Это вряд ли то, чего мы хотели добиться! Изменим немного код этой строки

```
var person = JSON.parse(xhr.responseText);
document.getElementById("myDiv").innerHTML=person.name+' '+person.lastname;
```

Теперь у нас должно вывестись Остап Бендер вместо Hello World! Значит Вы таки да освоили JSON ;)

JSON.stringify

Иногда нужно JavaScript-объект преобразовать в JSON. Для этого используется метод JSON.stringify. Например

```
var obj = {};

obj.a = 5;
obj.b = 7;

var str = JSON.stringify(obj);

console.log(str); //{"a":5,"b":7}
```

Поля с методами в объекте либо убираются при таком преобразовании, либо значение такого поля будет null

Примеры JSON-файлов

Мы можем создать JSON состоящий из массива объектов

```
[
  {"name": "Mark", "surname": "Twain"},
  {"name": "Lev", "surname": "Tolstoy"},
  {"name": "Antoine", "surname": "de Saint-Exupéry"}
]
```

Допустим мы получили этот JSON в переменную str. После того как мы ее распарсим, мы получим массив с тремя элементами, в роли которых будут объекты

```
var obj=JSON.parse(str);

console.log(obj.length); //3

console.log(obj[1].surname); //Tolstoy
```

Практика:

1. Создаем JSON файл со свойствами прямоугольника – шириной, высотой и фоновым цветом. С помощью AJAX считываем его. Парсим с помощью JSON.parse и используем полученные данные для вывода прямоугольника на экран.
2. Есть JSON-файл в нем хранятся имена людей. Получаем данные из файла с помощью AJAX. Вывести квадраты с именами людей внутри.
3. Создаем JSON файл продукта. В нем должно содержаться название, описание, цена и адрес картинки.
4. Создаем JSON, в котором будут храниться информация о товарах. Получить его AJAX'ом и вывести товары в виде каталога.
5. Создать Single Page Application, имитирующее трехстраничный сайт

navigator, screen, location

navigator.platform

navigator.userAgent

Получение координат

```
var x = document.getElementById("demo");

function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
```

screen width - ширина экрана height - высота экрана

```
<script>
    width=document.body.clientWidth; // ширина
    height=document.body.clientHeight; // высота
    alert ("Разрешение окна клиента: "+width+"x"+height);
</script>
```

location location.href=URL

```
location.href = "mailto:someone@example.com";
```

будет пытаться открыть почтовый клиент

location.assign - перенаправляет на текущую страницу

location.replace

location.reload

```
location.reload();
    assign()
    replace()
```

```
document.getElementById("demo").innerHTML =
"Page location is " + window.location.href;
```

document

document.activeElement

history

.back()

.forward

.go(-1) (равносильно back)

pushState – добавляет записи в историю. Первый параметр содержит объект, который говорит о состоянии

```
history.pushState({foo: 'bar'}, 'Title', '/baz.html')
```

onpopstate – срабатывает при изменении history (нельзя отменить действие кнопки back с помощью prevent default) в event onpopstate передается свойство state

```
window.onpopstate = function(event) {
    alert("location: " + document.location + ", state: " + JSON.stringify(event.state));
};
```

replaceState – заменяет текущее состояние stateObject

Изначально в history хранится пустой объект {}, вот его то и неплохо при запуске страницы поменять

Практика:

1. Сделать так, чтобы при изменении размеров окна, нам выводились его текущие размеры

Продвинутая консоль

<https://habrahabr.ru/post/114483/>

1. Сделать цикл и замерить сколько времени он выполняется
2. Сделать assert проверку на выделение чекбокса

history

history - объект браузера из BOM, который отвечает за очередь истории.

С history может работать как сам браузер, например при переходе с одной страницы на другую, так скрипты страницы, добавляя в history свои записи

Следующие три функции перемещаются по записям истории.

.back() - перейти на одну страницу в истории назад

.forward() - перейти на одну страницу в истории вперед

.go(n) - перемещает пользователя на n страниц вперед. go(-1) равносильно .back()

Если же нам нужны свои записи в истории, то нужен будет метод pushState

.pushState – добавляет записи в историю. Первый параметр содержит объект, который говорит о состоянии

```
history.pushState({'foo': 'bar'}, 'Title', '/baz.html');
```

onpopstate – срабатывает при изменении history (нельзя отменить действие кнопки back с помощью prevent default) в event onpopstate передается свойство state, в котором хранится объект, указанный первым параметром pushState.

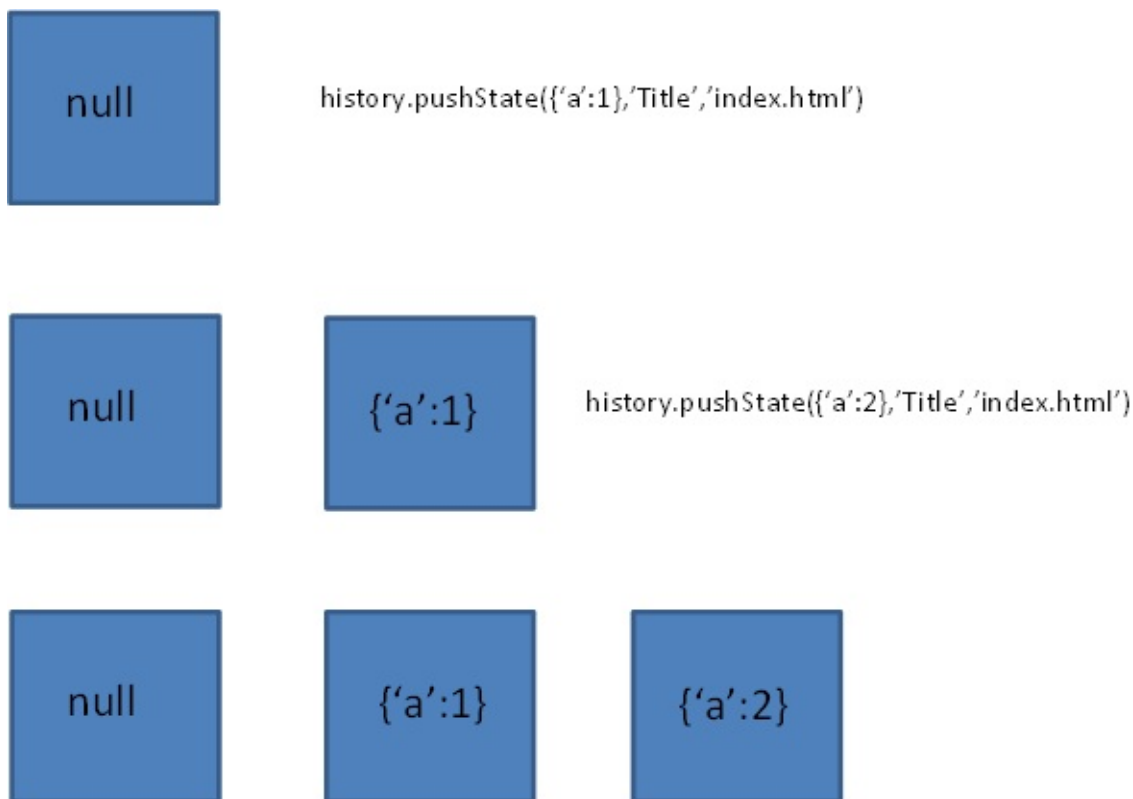
Для примера, приведенного выше в event.state будет находиться объект {'foo':'bar'}, т.е. в event.state.foo будет находиться значение bar

```
window.onpopstate = function(event) {  
    alert("location: " + document.location + ", state: " + JSON.stringify(event.state));  
};
```

replaceState – заменяет текущее состояние stateObject

```
history.replaceState({'foo': 'bar'}, 'Title', '/baz.html');
```

Изначально в history хранится пустой объект {}, вот его то и неплохо при запуске страницы поменять(см картинку ниже)

**Пример:**

Ниже приведен пример с блоком, который меняет свой при клике. Наша задача добавить пользователю возможность отменять свои действия.

Реализация состоит из трех этапов:

1. Реализовать функциональность и реакцию на действия пользователя.
Отображение должно зависеть от данных в "состоянии". Это может быть отдельный объект или набор переменных
2. Реализовать добавление объекта состояния при каждом действии пользователя с помощью `history.pushState`. Это активирует кнопку Back в браузере, но действия всё еще не будут отменяться.
3. Реализовать отмену действий пользователя при помощи установки обработчика на событие `window.onpopstate`

```
<style>
  .blue{
    width:100px;
    height: 100px;
    background-color: blue;

  }

  .red {
    width: 100px;
    height: 100px;
    background-color: red;
  }
</style>

<div class=blue>
</div>
```

Добавим к этому коду JavaScript

```
var block = document.querySelector('.blue');

history.replaceState({'color': block.className}, '', '');

block.onclick = function() {
  if (this.className=='red') {
    this.className='blue';
  } else {
    this.className='red';
  }
  history.pushState({'color': this.className}, '', '');
}

window.onpopstate = function(event) {
  block.className=event.state.color;
}
```

Практика:

1. Счетчик. Кнопка + увеличивает его на 1. Сделать, чтобы кнопка back позволяла вернуть меньшие значения счетчика.
2. Серый блок и три квадрата для выбора цвета. Сделать возможность отмену действия
3. Сайт с тремя страницами, реализован на одной html-странице со скриптом. Реализовать работу history
4. Есть интерфейс для выбора размеров и цвета круга. Реализовать выбор параметров и возможность их отмены

cookies

Запускать под сервером. Локально работает только под Mozilla

Записывает куку userName со значением Vasya

```
document.cookie = "userName=Vasya";
```

Записывает куку cn со значением 2 и fn со значением 3

```
document.cookie='cn=2';  
  
document.cookie='fn=3';
```

Чтобы переписать куку нужно ее повторно переустановить

```
document.cookie='cn=1';  
  
document.cookie='cn=2';
```

То есть кука cn станет равной двум

Полный формат записи кук

```
<name>=<value>[;expires=<date>][;path=<some_path>][;domain=<domain_name>][;secure]
```

Пример с использованием свойства path

```
<script type="text/javascript">  
    document.cookie = "name1=11; path=/;";  
    document.cookie = "name2=13; path=/;";  
    alert(document.cookie);  
</script>
```

Установка куки на час

```
var now = new Date();
var time = now.getTime();
time += 3600 * 1000;
now.setTime(time);
document.cookie =
'username=' + value +
'; expires=' + now.toUTCString() +
'; path='/;
```

Считывание значения cookie

```
function read_cookie(key)
{
    var result;
    return (result = new RegExp('(?:^|; )' + encodeURIComponent(key) + '=(^;]*)')
    .exec(document.cookie)) ? (result[1]) : null;
}
```

Либо использование библиотеки jQuery

<https://github.com/carhartl/jquery-cookie>

using jquery-cookie

I find this library helpful. 3.128 kb of pure convenience.

add script

```
<script src="/path/to/jquery.cookie.js"></script>

//установка значения
$.cookie('name', 'value');

//считывание значения
$.cookie('name');
```

Практика:

1. Сделать выбор цвета у прямоугольника и его сохранение
2. Делаем страницу, которая считает количество своих загрузок
3. Делаем проверку логина пароля. Если пользователь ввел правильно, выводим кубок
4. Делаем два блока с выбором цвета. После перезагрузки страницы цвета блока сохраняются.
5. Пишем Томагочи

Создаем SPA

Drag-n-Drop

Создадим картинку и блок

```
<div style="width:100px;height:100px;background:red" id="block" >
</div>


```

Давайте попробуем перетащить эти объекты. Перетаскивание не удастся, но не удастся по-разному. Обратите внимание, что картинка все-таки перетаскивается, но возвращается назад. В то время как блок просто никак не реагирует.

Чтобы исправить эту ситуацию допишем в наш блок атрибут `draggable="true"`

```
<div style="width:100px;height:100px;background:red" id="block" draggable="true">
</div>


```

Теперь давайте сделаем два DIV'a

drag_object и **dropzone**

```
<!DOCTYPE HTML>
<html>
<head>
<style>
#drag_object {
width:100px;
height:100px;
border:1px solid #aaaaaa;
background:cornflowerblue;
}
#dropzone {
background:grey;
width:500px;
height:500px;
}
</style>

</head>
<body>
<div id="dropzone"></div>
<br>
```

```
<div id="drag_object" draggable="true" ondragstart="event.dataTransfer.setData('text/plain',null);" >
</div>

<script>

var dragged = null;

function dragStart(event) {
    dragged = event.target;
    // сохраняем ссылку на перетаскиваемый объект

    console.log('dragstart');
}
drag_object.addEventListener('dragstart',dragStart,false);

function dragOver(event) {
    console.log('dragover');
    event.preventDefault();
    //отменяем обработчик по умолчанию, который блокирует ondrop
}

function drop(event) {
    event.preventDefault(); //обработчик по умолчанию пытается перейти по ссылке

    console.log('drop');
    //event.target в этом событии содержит ссылку на место, куда "кидают" перетаскиваемый объект
    if ( event.target.id == "dropzone" ) {
        event.target.appendChild( dragged );
    }
}

//теперь установим обработчики на dropzone'у
dropzone.addEventListener('dragover',dragOver,false);
dropzone.addEventListener('drop',drop,false);

</script>
</body>
</html>
```

ev.dataTransfer.setData("text", ev.target.id);

```
<!DOCTYPE HTML>
<html>
<head>
<style>
    #drag_object {
        width:100px;
        height:100px;
        border:1px solid #aaaaaa;
        background:cornflowerblue;
    }

    #dropzone {
        background:grey;
        width:500px;
        height:500px;
    }

</style>
<script>

function drag_start(ev) {

    ev.dataTransfer.setData("text", ev.target.id);
    //сохраняем id перетаскиваемого объекта в dataTransfer
}

function drag_over(ev) {
    ev.preventDefault();
    //отменяем действие по умолчанию
}

function drop(ev) {
    ev.preventDefault();
    var data = ev.dataTransfer.getData("text");
    ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="dropzone" ondrop="drop(event)" ondragover="drag_over(event)"></div>

<br>
<div id="drag_object" draggable="true" ondragstart="drag_start(event)" >
</div>
</body>
</html>
```

Без dataTransfer перетаскивание не будет работать в Firefox!

Интересное чтение:

Простой пример <https://developer.mozilla.org/en-US/docs/Web/Events/dragstart>

<http://www.html5rocks.com/en/tutorials/dnd/basics/>

<http://html5doctor.com/native-drag-and-drop/>

<https://learn.javascript.ru/drag-and-drop>

Наложение эффектов при перетаскивании

<http://www.kryogenix.org/code/browser/custom-drag-image.html>

Перетаскивание файлов <http://www.thecssninja.com/javascript/gmail-dragout>

Чтобы срабатывал стандартный обработчик объект должен быть draggable. Картинки и так являются draggable объектами Нам нужно поставить preventDefault у dragover и drop

Перетаскивание элемента по экрану <http://stackoverflow.com/questions/6230834/html5-drag-and-drop-anywhere-on-the-screen> Реализация <http://jsfiddle.net/robertc/kKuqH/>

Практика:

1. Есть две области. Перетаскиваем объект из одной в другую
2. Есть область, есть много объектов. При перетаскивании объектов в область, номер в блоке увеличивается.
3. Перетащить объект в произвольное место
4. Сделать ползунок
5. Редактор ToDo-листов
6. Игра Civilization Wars

COFFEE FileReader

<https://developer.mozilla.org/ru/docs/Web/API/FileReader/readAsDataURL>

<http://www.javascripture.com/FileReader>

```
<input type='file' onchange='openFile(event)'\n<script>
```

```
</script>
```

#COFFEE LocalStorage

```
localStorage.setItem('myCat', 'Tom');
```

getItem, removeItem

```
window.addEventListener('storage', function(e) {  
  document.querySelector('.my-key').textContent = e.key;  
  document.querySelector('.my-old').textContent = e.oldValue;  
  document.querySelector('.my-new').textContent = e.newValue;  
  document.querySelector('.my-url').textContent = e.url;  
  document.querySelector('.my-storage').textContent = e.storageArea;  
});
```

Глава V - Регулярные выражения

При первом рассмотрении регулярные выражения кажутся очень запутанными, но в дальнейшем Вы будете с легкостью их читать.

Регулярные выражения - это тот механизм, который позволит Вам экономить кучу времени при работе со строками.

Синтаксис регулярных выражений

Создание регулярных выражений в JavaScript

```
var re1 = /ab+c/i;  
var re2 = new RegExp('ab+c', 'i');
```

Общее описание регулярных выражений. Большая часть материала подходит и под JS http://www.php.su/lessons/?lesson_17

search

Изучим синтаксис регулярных выражений на примере функции search

search проверяет, есть ли в строке подстрока, соответствующая регулярному выражению. Если есть, то выводится позиция первой найденной подстроки в строке. Если подстрока, соответствующая регулярному выражению не была найдена, то search вернет -1. Например,

```
var str = "This is my test";  
  
var res1 = str.search(/e/); //12  
  
var res2 = str.search(/i/); //2, так как счет идет с нуля  
  
var res3 = str.search(/a/); //-1
```

Попробуем написать регулярное выражение, которое находит слово внутри предложения

```
var str = "А пряников сладких и вкусных, всегда не хватает на всех!";  
  
var res = str.search(/пряников/);
```

Допустим мы хотим проверить, что строка является нужным нам словом, тогда нам пригодятся символы начала и конца данных ^ и \$ соответственно.

```
var password = 'swordfish';

var sentence = 'One of the words here is a swordfish';

password.search(/^swordfish$/); //0, пароль в точности соответствует слову swordfish

sentence.search(/^swordfish$/); // -1 . Да слово swordfish есть, но строка начинается не с него
```

Задавать регулярное выражение в виде слова и искать это слово в предложении - слишком простая задача! Для этой цели мы могли бы использовать метод `.substr`. Регулярные выражения намного мощнее! Например мы хотим, чтобы наш шаблон находил как слово `bingo`, так и слово `bongo`. Для этих целей мы можем воспользоваться конструкцией символьный класс! Как видно, оба этих слова различаются одной буквой `i` и `o` - поместим их внутрь квадратных скобок `[io]` - такая конструкция означает, что вместо нее может быть либо `i`, либо `o`. Само регулярное выражение будет выглядеть вот так `/b[io]ngo/`

```
var str = "Какая девчонка! bingo-bongo! Ради такой я завалю даже Кинг-Конга!";

var res = str.search(/b[io]ngo/); // будет найдено слово bingo
```

Допустим мы хотим узнать, упоминаются ли в строке нужные нам слова `king` и `queen`. Нам будет достаточно только одного слова. В решении этой задачи нам помогут подмаска `()` и конструкция или `|`

```
var str = "We were the kings and queens of promise";

str.search(/(king|queen)/); //будет найден king в kings
```

Давайте теперь рассмотрим такую задачу. Представим, что мы ищем слово из пяти букв, которое заканчивается на `'ня'`. Первые три буквы нам не важны, поэтому мы заменим их на символ `.` - то есть любой символ.

```
var re = /\.ня/;

var str = 'Одесская кухня таит немало сюрпризов для непосвященных';

str.search(re); // будет найдено слово кухня
```

Повторения

Допустим мы хотим, чтобы строка начиналась с какой-то буквы, а дальнейшие символы могут быть любыми. Любой символ - это точка, но как задать их неопределенное количество? Для этого используется *

```
var re = /^a.*/i

var str1 = 'Афанасий';

var str2 = 'Василий';

str1.search(re); // 0

str2.search(re); // -1
```

То есть .* - это любое количество любых символов. ^ - нам приходится использовать, потому что нам важно, чтобы строка начиналась с символа а. И важно добавить модификатор i, чтобы наш шаблон находил и большие буквы тоже.

Если нас интересуют строки, в которых должна быть одна и более буквы а, тогда нужно воспользоваться символом +

```
var start = 'Без паники!';
var stop  = 'aaaaa';
var bad   = 'bbbbbb';

var re = /a+/; // одна и более буквы а

stop.search(re); // 0
bad.search(re);  // -1

var re2 = /a*/;

bad.search(re2); // 0
```

Аналогично ? - это одно или ноль повторений.

```
var re = /(Great )?Odessa/;
```

{2,4} - задаст 2,3 или 4 повторения

{0,1} - тоже самое, что вопрос {0,} - тоже самое, что * {1,} - тоже самое, что +

Что характерно *, +, ? и даже {} можно применять как к отдельным символам так и к подмаскам.

```
var re = /(bla){3,3}/; //задаем три повторения

var important = 'blablabla';

important.search(re); //0
```

На самом деле мы могли бы упростить регулярное выражение до вида

```
var re = /(bla){3}/
```

Специальные символы в символьном классе

```
var re= /^[^ab]*/
```

Любые символы, кроме символов a и b.

```
var re=/[a-d]{7}/
```

Слово из семи букв, каждая из которых может быть a,b,c,d

Если промежутков несколько, то всё-равно выбирается один символ

```
var re=/[0-9a-o]/
```

Дополнительные техники

Вернемся к нашему регулярному выражению /...ня/

Вообще-то точка подойдет вместо любого символа в том числе и пробела, поэтому в следующей строке тоже будет найдено совпадение.

```
var re = /...ня/;

var str = "Вася и Таня хорошо проводят время, изучая регулярные выражения".

str.search(re); // будет найдена ' Таня'
```

Примеры ниже не совсем корректны, потому что \w - не включает в себя символы кириллицы.

Чтобы избавиться от такой проблемы заменим . на \w - буква, цифра или нижнее подчеркивание.

```
var re = /\w\w\wня/;

var str = "Вася и Таня хорошо проводят время, изучая регулярные выражения".

str.search(re); // -1
```

```
var re = /\w\w\wня/;

var str = "Здесь будет найдена простыня, хотя это не слово из пяти букв".

str.search(re); // таки да найдена
```

Заменим на

```
var re = /\b\w\wня(\w|$)/;
```

Где \b - символ на границе слова, \W - всё что угодно, кроме цифры, буквы и нижнего подчеркивания, а \$ - конец строки, то есть после 'ня' - либо какие-то символы, из которых слово не построишь, либо конец строки.

match

split

reg.exec

Практика:

1. Проверить есть ли в строке буква а
2. Есть ли в строке две буквы а подряд
3. Если в строке две буквы а
4. Есть ли в строке ровно две буквы а
5. Найти вхождение в строку упоминание времени, например 12:57 . Вывести на экран отдельно количество часов, отдельно количество минут
6. Выбрать строки, которые не содержат три х подряд
7. Проверить является ли строка номером телефона
8. Проверить является ли строка емейлом
9. Разбить предложение на слова
10. Убрать лишние пробелы из строки
11. Сделать транслитерацию
12. Вывести все слова содержащие буквы k либо две буквы l
13. Перевести все буквы к нижнему регистру
14. Заменить в строке числа на квадраты чисел

15. Узнать есть ли в строке повторяющееся слово

Дополнительные задачи <http://habrahabr.ru/post/167015/>

1. Выделяем повторяющиеся последовательности тегом

match

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/match

В отличие от `search` `match` позволяет находить несколько вхождений паттерна в строку, либо умеет разбивать исходную строку на подстроки за счет подмасок в регулярном выражении

Работа с подмасками

Допустим мы хотим найти числа, разделенные двоеточием

```
var str = 'Есть у нас вот такая строка с числами 175:27 и всё';  
  
var res = str.match(/(/d+):(/d+)/);
```

В `res` мы получим массив `['175:27','175','27']`, то есть значение первой подмаски пойдет в первый элемент массива, значение второй подмаски пойдет во второй элемент массива.

Поиск нескольких вхождений

Иногда нам нужно найти все вхождения нашего паттерна в строку. Для этого существует модификатор **g** в регулярных выражениях

```
var str = "Все эти пары чисел 489:80, 34:26, 89:17" будут найдены";  
  
var results = str.match(/\\d+:\\d+/g);
```

`results` будет массивом с элементами `['489:80','34:26','89:17']`

Практика:

1. Найти упоминание времени в строке, например `19:40`. Обратите внимание, что записи вроде `25:67` не являются записями о времени

split

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/split

str.split позволяет разбить строку на подстроки. С помощью регулярного выражения можно задать шаблон для разделителя

Допустим у нас есть предложение, в котором слова разделены пробелами. Мы хотим получить массив строк.

```
var str = 'Хорошо живет на свете Винни Пух';  
  
var arr = str.split(' ');
```

Если бы мы вызвали split(""), то получили бы массив букв

Конструкция, которую любят на собеседованиях

```
split('').reverse().join('')
```

Практика:

1. Разбить строку на слова. В качестве разделителя могут выступать пробел, запятая, точка

replace

str.replace - находит подстроки, которые нужно поменять и выполняет замену, на строку или на результат работы функции

```
str.replace('нецензурные слова', '***');
```

str.replace можно использовать для удаления каких-то включений из строки. Для этого нужно сделать замену на пустую строку

```
str.replace('спам', '');
```

В качестве первого параметра может быть регулярное выражение

```
str.replace('/xxx/', '');
```

Параметры в строке замены Можно использовать следующие специальные сочетания в строке замены для работы с найденной подстрокой:

\$& - вставляет совпавшую подстроку \$` - вставляет часть строки, которая предшествует совпавшей подстроке \$' - вставляет часть строки, которая следует за совпавшей подстрокой \$n или \$nn, где n/nn - десятичные цифры Вставляет n-ю скобку в совпадении, если первый аргумент - объект RegExp

\$\$ - вставляет "\$"

Функция замены

Функция replace в качестве второго параметра может принимать функцию замены. В эту строку передается подстрока, соответствующая шаблону, значения подмасок, позиция подстроки в исходно строке и сама исходная строка. Результат, который вернет функция и будет подставляться вместо исходной строки.

Пример функции замены

```
function replacer(found, p1, p2, offset, s) {  
  
    return p1 + ", " + p2;  
  
}
```

found - найденная строка, которая соответствует шаблону

p1, p2 - подстроки, которые соответствуют шаблону

offset - это позиция найденной строки в исходной строке

s - исходная строка, для которой мы вызываем str.replace

Напишем функцию, которая меняла бы два числа между, которыми плюс на их сумму. То есть строка "Великая загадка Вселенной равна 13+32-ти" должна замениться на "Великая загадка Вселенной равна 45-ти".

Для этого нам понадобится регулярное выражение, которое бы находило такую конструкцию. Мы ищем числа разделенные знаком вопроса. Число, если говорить о представлении в строке, это одна и более цифр, то есть `\d+`. Плюс между цифрами придется заэкранировать. То есть получим следующее регулярное выражение:

```
/\d+\+\d+/
```

Нам нужно будет доставать числа из найденной подстроки, поэтому нам понадобятся подмаски

```
/(\d+)\+(\d+)/
```

Далее нам понадобится функция замены следующего вида

```
function sumUp(pattern_string, first, second) {  
    return parseInt(first)+parseInt(second)  
}
```

Далее нам необходимо вызвать str.replace

```
var str = "Найдем результат 23+52 и выведем на экран";  
  
var result = str.replace(/(\d+)\+(\d+)/, sumUp)
```

Обратите внимание, что str.replace не меняет исходную строку, а лишь возвращает результат замены.

Модификатор g

Если мы запускаем запрос без модификатора g, то замена происходит только по первому найденному значению. Модификатор g позволяет сделать замены во всей строке.

Предыдущий пример будет выглядеть вот так:

```
var result = str.replace(/(\d+)\+(\d+)/g, sumUp)
```

Практика:

1. Убрать все цифры из строки
2. Убрать лишние пробелы между словами в предложении
3. Найти в строке числа и заменить их на квадраты чисел

Глава VIII - jQuery

jQuery подарило миру универсальность работы кода, удобные фичи и море готовых плагинов.

Несмотря на улучшение согласованности стандартов JS в разных браузерах, реализации многих механизмов из jQuery в стандарте ECMAScript, он, jQuery, все еще остается очень удобным инструментом для решения самых разнообразных задач.

P.S. В качестве отличных справочников по jQuery рекомендую

<https://oscarotero.com/jquery/> и <https://api.jquery.com/>

Быстрый старт

Подключение

Мы можем подключить jQuery с сервера Google

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
</script>
```

Либо скачать файл с <http://jquery.com/download/>

и подключить его по локальному пути

```
<script src="js/jquery.min.js">
</script>
```

После подключения у нас будет доступна переменная \$. Для проверки откройте консоль разработчика и вбейте туда \$

Работа с DOM-элементами

Через \$ мы можем обращаться к отдельным элементам через CSS-селекторы

```
<style>
#block {
    height:200px;
    width:200px;
    background:red;
}
</style>

<div id="block">
</div>

<script>
$('#block').css('height'); //вернет нам '200px'

$('#block').css('height','400px');//высота блока станет 400px

$('#block').html('Новый текст внутри блока');// добавляем новый текст внутри блока
</script>
```

Установка нескольких свойств


```
var cssValues = {  
  "color": "red",  
  "font-size": "25px"  
}  
$("p").css(cssValues);
```

Установка обработчиков событий

```
$('#block').on('click', function(event){  
  alert('Test');  
});
```

Практика:

1. Есть синий блок. С помощью jQuery поменять его цвет на красный.
2. Есть блок. Сделать так, чтобы при кликах блок менял цвет с первого на второй. Со второго на первый.
3. Есть блоки. При клике на блок выводим цвет блока, на который кликали в предыдущий раз
4. Игроку показывается круг в произвольном месте экрана. Если игрок успел кликнуть на круг в течении двух секунд, то он зарабатывает балл.

Работа с селекторами

Через \$ мы можем обратиться к DOM-объектам, используя CSS-селекторы

Например \$('div') - обратиться ко всем div'ам на странице

Если мы хотим обратиться ко второму элементу на странице, то мы можем воспользоваться функцией eq, которая позволяет обращаться к элементу по его номеру. Нумерация начинается с нуля!

```
$( "div" ).eq( 2 ).css( "background-color", "red" );
```

\$("div").length - вернет нам количество элементов в коллекции

\$('#block') -позволит обратиться к элементу с id block

\$('.red_block') - получит коллекцию объектов с классом

Мы можем записать внутри функции \$ DOM-объект - и в результате получить jQuery объект со всеми функциями.

```
var obj = document.getElementById("block");  
$(obj).css("height", "200px");
```

Обращение к атрибутам data-

```
<div id="superid" data-foo-bar="123"></div>
```

```
$('#superid').data('fooBar'); // вернет 123
```

Сделает фон всех нечетных элементов красным

```
$( "li" ).not( ":even" ).css( "background-color", "red" );
```

Выберет пятый элемент

```
$( "li" ).eq( 4 ).css( "background-color", "red" );
```

Дополнения:

1. Скорость выполнения разных jQuery запросов <https://habrahabr.ru/post/103174/>

Практика:

1. Есть четыре блока. Сначала "зажигается" первый. Потом он гаснет, зажигается второй. Потом гаснет второй, зажигается третий. То есть блоки по очереди зажигаются с 1-го по 4-й. После четвертого очередь переходит опять к первому и все повторяется.

События

Стандартные события

```
$('.class').click(function(){  
    //что-то делаем  
})
```

Работа с события в общем случае

```
// создаем обработчик  
$('.class').bind('click', function(){  
    // что-то делаем  
});  
  
//или  
  
// создаем обработчик  
$('.class').on('click', function(){  
    // что-то делаем  
});  
  
  
// вызываем обработчик  
$('.class').trigger('click');  
  
// удаляем все обработчики для данной коллекции объектов  
$('.class').unbind('click');
```

Можно создать свое событие

```
var body = document.body;  
  
$(body).bind("abracadabra",function() {  
    alert('Hi');  
});  
  
$(body).trigger("abracadabra"); //вызовет обработчик с Hi
```

Практика:

1. Кликаем на блок - выводится alert

2. Есть два блока. При клике на блок он меняет цвет. Если оба блока поменяли цвет - выводим сообщение
3. Есть галерея картинок. При клике на картинку она увеличивается.
4. Есть блоки и есть кнопка. При клике на любой блок, он меняет цвет. При клике на кнопку, все блоки восстанавливают свой исходный цвет.
5. Есть блоки, есть круги. Если активировано три блока или два круга, то выводим сообщение
6. Есть список тематик. Пользователю нужно выбрать 3 из 10. При выборе тематики блок с ней меняет цвет. Когда выбрано 3 блока из 10 дальнейших выбор блокируется, все блоки "сереют".

Анимация

Функции для скрытия/отображения

.hide()
.show()
.toggle()

https://www.w3schools.com/jquery/jquery_hide_show.asp

Покажет скрытые элементы за 300 миллисекунд

```
$( '.hidden' ).show( 300 );
```

```
$( '.hidden' ).show( 'slow' );
```

Функции сворачивания/разворачивания

.slideUp()
.slideDown()
.slideToggle()

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_eff_slideup_slidedown

Функции для появления/исчезновения

.fadeOut()
.fadeIn()
.fadeToggle()

https://www.w3schools.com/jquery/jquery_fade.asp

animate

http://www.w3schools.com/jquery/eff_animate.asp

Пример

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_eff_animate

```
$("#button").click(function(){  
    $("#div").animate({left: '250px'});  
});
```

Вариант вызова №1

properties - CSS свойства, которые мы анимируем

duration - продолжительность, задается в миллисекундах

easing - определяет будут ли ускорения, торможения в анимации(по умолчанию swing, можно поставить linear)

complete - функция, которая вызывает после завершения анимации.

Например:

```
$( "#clickme" ).click(function() {  
    $( "#book" ).animate({  
        opacity: 0.25,  
        left: "+=50",  
        height: "toggle"  
    }, 5000, function() {  
        // Animation complete.  
    });  
});
```

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

jQuery.fx

jQuery.fx.off - флаг, который разрешает или запрещает анимации

https://www.w3schools.com/jquery/prop_jquery_fx_off.asp

jQuery.fx.speeds - настройка скорости анимации

```
// переустанавливаем уже определенную скорость
jQuery.fx.speeds.fast = 50;

// создает новую скорость
jQuery.fx.speeds.turtle = 3000;

//Эта анимация проходит за 50 миллисекунд
$( '.hidden' ).hide( 'fast' );

//Можем применить нашу новую скорость
$( '.other-hidden' ).show( 'turtle' );
```

jQuery.fx.interval

Полезное чтение:

1. Объяснение базового варианта jQuery.animate <http://papermashup.com/the-jquery-animate-method-explained/>
2. Примеры для jQuery.animate <http://viralpatel.net/blogs/understanding-jquery-animate-function/>

Практика:

1. Сделать блок. При клике на кнопку, если он есть, он исчезает. Если его нет, то он появляется.
2. Сделать сворачивание-разворачивание элементов списка
3. Есть галерея картинок. При клике на картинку, она выезжает на середину экрана и увеличивается.
4. При клике на круг он должен увеличиваться

#COFFEE AJAX

```
$("#button").click(function(){
    $.ajax({url: "demo_test.txt", success: function(result){
        $("#div1").html(result);
    }});
});
```

```
$.ajax({
    url: url,
    type: "POST",
    data: data,
    success: success,
    dataType: dataType
});
```

```
$.ajax({
    url: '/ajax/example.html',           // указываем URL и
    dataType : "json",                   // тип загружаемых данных
    success: function (data, textStatus) { // вешаем свой обработчик на функцию success

        $.each(data, function(i, val) { // обрабатываем полученные данные
            /* ... */
        });
    }
});
```

Полезное чтение:

1. Загрузка картинки на сервер с помощью ajax
<https://stackoverflow.com/questions/19447435/ajax-upload-image>

Практика:

1. Передаем на сервер два числа - получаем сумму двух чисел
2. Получить с сервера список имен и телефонов пользователей в виде JSON-файла.

Модификация DOM

Навигация по DOM

.next() - аналог .nextElementSibling в jQuery <https://api.jquery.com/next/>

.prev() - аналог .previousElementSibling в jQuery

.parent(); - аналог .parent в jQuery

.children(); - получает коллекцию детей от заданного элемента

.each() - применяет функцию к каждому элементу коллекции.

<http://api.jquery.com/each/>

```
$( "li" ).each(function( index,element) {  
    console.log( index + ": " + $( this ).text() );  
});
```

Добавление контента внутрь блока

.append() - добавляет элементы внутрь блока после контента

```
$( '.container' ).append( "<p>Some text</p>" )
```

Следующий код переместит объекты h2 в container

```
$( ".container" ).append( $( "h2" ) );
```

.appendTo - отличается от .append тем, что добавляемый контент стоит в начале записи.

```
$( "<p>Test</p>" ).appendTo( ".inner" );
```

.html() - аналог .innerHTML для jQuery

.prepend() - добавляет содержимое внутрь блока перед контентом.

.prependTo()

Добавление ДО и ПОСЛЕ тегов

.before

```
$( ".inner" ).before( "<p>Test</p>" );
```

Добавит <p>Test</p> до .inner

.insertBefore

.after

Обертки

.wrap() - оборачивает каждый элемент из коллекции в блок обертку (n - элементов, n - оберток) <http://api.jquery.com/wrap/>

.wrapAll() - оборачивает сразу все блоки из коллекции в ОДНУ обертку для всех (n - элементов, одна обертка)

.wrapInner() - оборачивает контент внутри элементов.

Удаление

.detach - удаляет элементы из DOM, но возвращает их в качестве результата

.empty - удаляет внутреннее содержимое

.remove - удаляет элементы безвозвратно

.unwrap - удаляет родителей у элементов <https://api.jquery.com/unwrap/>

.clone - если делать appendTo без клона, то он сделает перенос, если делать с, то будет копирование

Работа с атрибутами

.attr() - позволяет получать и устанавливать значение атрибута <http://api.jquery.com/attr/>

.data() - позволяет получать значение data атрибутов.

```
$(obj).data('type');// считает значение из атрибута data-type
```

.val() - получает значение у input'ов

Модификация классов

.addClass() **.hasClass()** **.removeClass()** **.toggleClass()**

Практика:

1. Делаем, чтобы по клику на кнопку добавлялся красный, квадратный блок

2. Добавляем в предыдущем задании, чтобы при клике на красный блок, он становился синим
3. Делаем массив фотографий. Делаем кнопку, при клике на которую на экран добавляется очередная фотография. Это происходит до тех пор пока в массиве есть фотографии.
4. Добавляем кнопку, которая переносит все синие блоки в "корзину", то есть внутрь какого отдельного блока на экране
5. Делаем список с подпунктами. При клике на пункт, он разворачивается. Сделать возможность переноса подпунктов.
6. Игра "Заполни стакан". Есть блоки, есть стакан. При клике на блок, он перемещается в стакан. Цель заполнить стакан доверху. Если стакан переполнен - Вы проиграли.

#COFFEE Работа с формами

Обращение к input'ам разного типа можно делать через селекторы с двоиточеем.
Например

```
<input type="text">
```

Получим значения с этого input'a

```
$('#:text').val()
```

:button

:checkbox

:checked

:disabled

:enabled

:focus

:file

:image

:input

:password

:radio

:reset

:selected

:submit

:text

События форм

.blur()

.change()

.focus()

.select()

.submit()

Чтобы получить значение элемента, нужно сделать

```
$("select.foo").val();
```

Практика:

1. Есть массив "использованных" логинов. При вводе логина делаем проверку свободен ли он.
2. Реализовать такую же реакции на получение фокуса, как на <http://webfont.ru/>

Плагины

Допустим мы хотим добавить свою функцию в возможности jQuery

```
jQuery.fn.greenbar = function() {  
    $(this).css('background', 'green');  
};  
  
$('div').greenbar();
```

Поддержка цепочек вызова

//Пример цепочки из двух методов css и html \$("p").css("color","red").html("Новое содержимое"); Для того, чтобы Ваш плагин не "разрывал" цепочку методов необходимо, чтобы он возвращал ключевое слово this.

Пример

```
(function($){  
  
    /* Создаем плагин с именем adjust, который будет устанавливать размер текста  
    выбранных элементов равным 1.4em и передавать их дальше по цепочки методов. */  
    $.fn.adjust=function(){  
        /* Для того, чтобы поддерживать цепочку методов нужно вернуть из плагина  
        преобразованную группу выбранных элементов */  
        return this.each(function(){  
            $(this).css("fontSize", "1.4em");  
        });  
    }  
  
})(jQuery)
```

Список плагинов

50 потрясающих jQuery плагинов

<https://habrahabr.ru/post/175045/>

Lightbox для галерей <http://lokeshdhakar.com/projects/lightbox2/>

Плагины для скроллинга <http://www.webdesignerdepot.com/2013/12/25-free-scrolling-plugins-for-awesome-experiences/>

<http://plugins.jquery.com/>

<http://www.smashingmagazine.com/2011/10/essential-jquery-plugin-patterns/>

<https://remysharp.com/2010/06/03/signs-of-a-poorly-written-jquery-plugin>

<https://msdn.microsoft.com/en-us/magazine/ff696759>

jQuery UI <http://ajpiano.com/widgetfactory/#slide5>

<http://free.matthieu.com/jquery.parallax-scroll/demo.html>

<http://www.wisdomweb.ru/JQ/plugin.php>

Примеры <http://tutorialzine.com/2013/04/50-amazing-jquery-plugins/> Gridster.js jQuery Countdown

Практика:

1. Сделать плагин, который бы раскрашивал блоки в зависимости от их размера
2. Сделать плагин, который оборачивает все красные элементы в границу толщиной 10 пикселей. Добиться возможности цепочечных вызовов

#COFFEE Техника Map-Reduce

Promises

```
var div = $( "<div>" );

div.promise().done(function( arg1 ) {
  // Will fire right away and alert "true"
  alert( this === div && arg1 === div );
});
```

```
$( "button" ).on( "click", function() {
  $( "p" ).append( "Started..." );

  $( "div" ).each(function( i ) {
    $( this ).fadeIn().fadeOut( 1000 * ( i + 1 ) );
  });

  $( "div" ).promise().done(function() {
    $( "p" ).append( " Finished! " );
  });
});
```

Код jQuery

В данном примере мы рассмотрим, что полезного мы можем вытащить из кода jQuery

1. <https://benmccormick.org/2015/06/08/how-jquery-works-an-introduction/>
2. <https://quickleft.com/blog/18-surprises-from-reading-jquery-s-source-code/>
3. <https://www.paulirish.com/2010/10-things-i-learned-from-the-jquery-source/>

ООП в JavaScript

Подходы объекто-ориентированного программирования в JavaScript'e могут сильно удивить тех, кто знаком с ООП в таких языках как C++, Java, PHP.

Тем не менее JavaScript решает все стандартные задачи по ООП, не строгая типизация, прототипы и замыкания позволяют привести новые, интересные решения.

Способы создания объекта в JavaScript

4-ре способа создать объект в JavaScript

<https://habrahabr.ru/post/17613/>

1-й способ `new Object()`

`Object` - стандартный конструктор JavaScript

```
var person = new Object();
//создадим свойство
person.name = 'Petya';

person.getName = function() {
    return this.name;
    //this содержит ссылку на объект
}
```

2-й способ Литеральная нотация

```
var person = {
    name: 'Petya',
    getName: function() {
        return this.name;
    }
}
```

Мы можем объединить два способа в один

```
var person = {}; //вместо new Object

person.name = 'Petya';

person.getName = function() {
    return this.name;
}
```

3-й способ работаем с объектом как с ассоциативным массивом

```
var person = new Object();

person["name"] = 'Petya';

person["getName"] = function() {
    return this.name;
}
```

Кстати, это единственный способ работать со свойствами, содержащими пробел в названии.

4-й способ создание объекта через конструктор

```
function CreatePerson(name) {
    this.name = name; //сохраняем параметр в свойстве объекта
    this.getName = function() {
        return this.name;
    }
}

//от одного конструктора можно генерировать сколько угодно объектов
var person = new CreatePerson('Petya');
var person2 = new CreatePerson('Vasya');
```

Чтобы сгенерировать объект, конструктор вызывается через new

Обычно такой способ создания объектов вызывает трудности в понимании у начинающих. По сути происходит следующее. Когда мы вызываем функцию конструктор через new, создается пустой объект {}, внутрь него мы помещаем функцию конструктор и вызываем ее. То есть процесс выглядит вот так:

```
var person = {}

person.CreatePerson = function(name) {
    this.name = name; //сохраняем параметр в свойстве объекта
    this.getName = function() {
        return this.name;
    }
}

person.CreatePerson('Petya');
```

Как видите name передается в свойство this.name нового объекта. Функция сохраняется в свойство getName объекта.

Часто конструктор по ошибке вызывают без `new`. В этом случае он просто добавляет глобальные переменные.

```
var name = 'Vasya';

function CreatePerson(name) {
    this.name = name;
    this.getName = function() {
        return this.name;
    }
}

var obj = CreatePerson('Petya');

console.log(obj); //Undefined, потому что CreatePerson ничего не возвращает

console.log(name); //вернет Petya, потому что вызов конструктора изменил значение name

console.log(getName); //вернет код функции getName
```

Добиваемся, чтобы конструктор вызывался без `new`

```
function User(name, passwordHash) {
    if (!(this instanceof User)) {
        return new User(name, passwordHash);
    }
    this.name = name;
    this.passwordHash = passwordHash;
}
```

То есть, если у нас функцию `User` запустили через `new`, `this` будет указывать на новый объект. Если же `User` была вызвана как обычная функция, то мы генерируем вызов с `new` сами.

Методы и свойства

Традиционно, в ООП свойством называется переменная внутри объекта, а методом функция вызываемая из объекта. В JavaScript'e граница несколько размыта, потому что методом по сути является свойство, в котором хранится функция. Давайте посмотрим пример. Сделаем объект для "приветствия". То есть у нас есть два свойства приветствие и имя адресата, а также метод, который это приветствие возвращает.

```
var GreetingObj = {  
  
  greeting: 'Hello',  
  name: 'Petya',  
  makeGreeting: function() {  
    return this.greeting+ ' '+this.name  
  }  
}  
  
//Вызовем метод makeGreeting  
var result = GreetingObj.makeGreeting();  
  
console.log(result);
```

То есть greeting и name являются свойствами, а makeGreeting методом. Вызвать метод можно через объект GreetingObj.makeGreeting()

Хранение объектов

```
var obj = {}  
  
obj.a = 5;  
  
var obj2 = obj;  
  
obj2.a = 7;  
  
console.log(obj.a); //?
```

Контекст вызова функции

```
var person = {  
  name: 'Petya',  
  getName: function() {  
    return this.name;  
  }  
}  
  
var person2 = {  
  name: 'Vasya'  
}  
  
person2.getName = person.getName;  
  
person2.getName(); // ?
```

this и куда он указывает


```
var name = 'Strange';

var person = {
  name: 'Petya',
  getName: function() {
    return this.name;
  }
}

//скопируем метод в переменную
var getThatName = person.getName;

//вызовем получившуюся функцию
var result = getThatName();

console.log(result);//?

console.log(this.name);//?

console.log(this);//?
```

Дополнительные материалы:

Разбирается много странных ситуаций, что может хорошо продвинуть понимание. Без базового понимания, сюда лучше не лезть

<https://habrahabr.ru/post/119391/>

Много всего, но на этом этапе будет сложно

<https://habrahabr.ru/post/48542/>

Выразительный JavaScript: Тайная жизнь объектов

<https://habrahabr.ru/post/241587/>

Практика:

1. Сделать объект с двумя свойствами и одним методом, который выводит их сумму.
2. Сделать объект (через конструктор), который хранит ширину, высоту и фоновый цвет прямоугольника
3. Создаем отдельную функцию, в которую передается объект. Функция рисует его в два раза больше.
4. Конструктор Car (передаем скорость) + метод Go. Добавляем машину, которая телепортируется
5. Создаем функцию, в которую передаем два объекта от Car. Определяем, кто победил. Смотреть, чтобы не было много return
6. Объект со свойством и методом show, которое выводит этот x через alert
7. Реализовать BlackJack с другим игроком

8. Создать объект галерею, внутри которого есть массив с адресами фотографий, а также три метода: `printGallery` - выводит галерею на экран, `printImage(i)` - выводит *i*-ю фотографию из массива, `getNumberOfPhotos()` - возвращает количество фотографий в галерее.

this

Интересное чтение:

1. Очень подробно о this <https://zellwk.com/blog/this/>

Замыкания

```
function outer() {  
  
    var a=5;  
  
    return function(b) {  
        console.log(a+b);  
    }  
  
}  
  
var inner =outer();  
  
inner(7); //получаем 12
```

Мы можем переделать внешнюю функцию в анонимную, и устроить вызов на месте

```
var inner = (function() {  
  
    var a=5;  
  
    return function(b) {  
        console.log(a+b);  
    }  
  
})();  
  
inner(8);
```

Функция, которая считает количество своих вызовов

```
var countFn = (function() {  
  
    var counter=0;  
  
    return function() {  
        counter++;  
        console.log(counter);  
    }  
  
})();  
  
countFn();// в консоль выведется 1  
countFn();// в консоль выведется 2
```

Практика:

1. Сделать функцию, которая бы считала количество вызовов самой себя
2. Сделать "функцию-копилку". На вход получает число, а возвращает сумму ранее введенных чисел
3. Сделать "функцию-мигалку". Функция по очереди выводит то true, то false
4. Сделать функцию, которая генерирует функции для отрисовки либо квадратов, либо кругов

Паттерн модуль

Паттерн модуль является дальнейшим развитием замыканий. Только вместо одной функции, возвращается несколько функций объединенных в объект. При это данные внутри объекта являются открытыми(public), данные внешней функции являются закрытыми для внешнего использования(private), но по прежнему доступны для методов возвращаемого объекта.

```
var obj= (function() {  
  
    var a=5;  
    var b=7;  
  
    return {  
        sum:function() {  
            return a+b;  
        },  
        mult: function() {  
            return a*b;  
        }  
    }  
  
})();  
  
console.log(obj.mult());  
console.log(obj.a);
```

Допишем наш модуль

```
var obj= (function() {

    var a=5;
    var b=7;
    var super_kvadrat = function() {
        return a*a+b*b;
    }

    return {
        sum:function() {
            return a+b;
        },
        mult: function() {
            return a*b;
        },

        sumOfSquares() {
            return super_kvadrat();
        }
    }
})();

console.log(obj.mult());
console.log(obj.a);
console.log(obj.sumOfSquares());
console.log(obj.super_kvadrat());
```

```
var weekDay = (function() {
    var names = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"];
    return {
        name: function(number) { return names[number]; },
        number: function(name) { return names.indexOf(name); }
    };
})();
```

Можно пойти дальше и не генерировать объект внутри внешней функции, а создавать его снаружи (объект weekDay), и передавать ссылку на него через параметр внешней функции (exports). В ходе выполнения внешней функции изначально пустой объект exports наполняется методами.

```
(function(exports) {  
  var names = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"];  
  
  exports.name = function(number) {  
    return names[number];  
  };  
  exports.number = function(name) {  
    return names.indexOf(name);  
  };  
})(this.weekDay = {});  
  
console.log(weekDay.name(weekDay.number("Saturday")));
```

Дополнительные материалы:

<https://medium.freecodecamp.com/javascript-modules-a-beginner-s-guide-783f7d7a5fcc>

Практика:

1. Сделать модуль-галерею. Массив картинок - закрытые данные. Есть три публичных метода: добавление картинки в массив, метод, который выводит i-ю картинку, метод, который выводит всю галерею.

Прототипы

<https://learn.javascript.ru/prototype>

```
var animal = {
  eats: true
};
var rabbit = {
  jumps: true
};

rabbit.__proto__ = animal;

// в rabbit можно найти оба свойства
alert( rabbit.jumps ); // true
alert( rabbit.eats ); // true
```

hasOwnProperty

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty

Метод `hasOwnProperty` позволяет проверять является ли данное свойство собственным свойством объекта или же оно взято из прототипа.

```
rabbit.hasOwnProperty('jumps');//true

rabbit.hasOwnProperty('eats');//false
```

prototype

```
var animal = {
  eats: true
};

function Rabbit(name) {
  this.name = name;
}

Rabbit.prototype = animal;

var rabbit = new Rabbit("Кроль"); // rabbit.__proto__ == animal

alert( rabbit.eats ); // true
```

constructor

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/constructor

свойство прототипа `constructor` указывает на конструктор объекта, то есть

```
rabbit.__proto__.constructor === Rabbit
```

Иногда в коде Вы можете столкнуться со следующей строчкой.

```
Car.prototype.constructor = Car;
```

Что делает эта строчка? И зачем?

<http://stackoverflow.com/questions/9343193/why-set-prototypes-constructor-to-its-constructor-function>

<http://stackoverflow.com/questions/8453887/why-is-it-necessary-to-set-the-prototype-constructor>

Object.create

`Object.create` -создает новый объект, используя в качестве прототипа, объект указанный в параметрах.

```
// Shape — суперкласс
function Shape() {
  this.x = 0;
  this.y = 0;
}

// метод суперкласса
Shape.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
  console.info('Фигура переместилась.');
```



```
};

// Rectangle — подкласс
function Rectangle() {
  Shape.call(this); // вызываем конструктор суперкласса
}

// подкласс расширяет суперкласс
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;

var rect = new Rectangle();

console.log('Является ли rect экземпляром Rectangle? ' + (rect instanceof Rectangle));
// true
console.log('Является ли rect экземпляром Shape? ' + (rect instanceof Shape)); // true
rect.move(1, 1); // выведет 'Фигура переместилась.'
```

// Немного надоедает использовать .call каждый раз. Может воспользоваться bind? //
Точно! Давайте привяжем функцию call к контексту slice.

```
slice = Function.prototype.call.bind(Array.prototype.slice);
```

// Теперь slice использует первый аргумент в качестве контекста

```
slice([1,2,3], 0, 1); // => [1]
```

Полезное чтение

1. Чем работа через прототип отличается от работы через конструктор?
<http://stackoverflow.com/questions/8433459/js-why-use-prototype>
<http://stackoverflow.com/questions/1948042/reason-for-using-prototype-in-javascript>
2. Почему прототипное наследование действительно важно
<http://aaditmshah.github.io/why-prototypal-inheritance-matters>

3. Преимущества прототипного наследования перед классическим

<http://stackoverflow.com/questions/2800964/benefits-of-prototypal-inheritance-over-classical?rq=1>

Практика:

1. Есть квадрат. Нужно сделать его прототипом круга
2. Сделать массив кругов и квадратов. В цикле вывести их на экран

Глава VI - Функциональное программирование

Promises

<https://stackoverflow.com/questions/36490803/javascript-promises-and-settimeout>

```
function a() {
  return new Promise(function (resolve, reject) {
    resolve("hi from a!");
  });
}

function b() {
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      resolve("hi from b!");
    }, 5000);
  });
}

function c() {
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      resolve("hi from c!");
    }, 1000);
  });
}

a().then(function (resultFromA) {
  console.log("a() responded with: " + resultFromA);
  return b(); // return
}).then(function (resultFromB) {
  console.log("b() responded with: " + resultFromB);
  return c(); // return
}).then(function (resultFromC) {
  console.log("c() responded with: " + resultFromC);
});
```

Полезное чтение:

1. MDN о промисах в es6
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise
2. Огромное количество вариантов использования промисов
<http://2ality.com/2014/10/es6-promises-api.html>
3. Еще варианты использования http://exploringjs.com/es6/ch_promises.html

4. Создание цепочек вызовов в JS

<https://html5hive.org/how-to-chain-javascript-promises/>

Рекурсия

Рекурсия. Понятие стека вызовов. Stack overflow

Практика:

1. Факториал
2. Перевернуть строку
3. Через рекурсивную функцию вывести массив
4. Найти максимум
5. Проверить является ли число степенью двойки
6. Проверить является ли число палиндромом
7. Вывести n-е число Фибоначчи
8. Сумма цифр числа
9. Ханойские башни

Контекст вызова функции

<http://habrahabr.ru/post/199456/>

```
var a=5;

function func() {
    console.log(this.a);
}
****
func(); // 5

var obj = {a:7};
obj.func = func;

obj.func(); // 7
```

apply , call

// Создадим простой объект, чтобы использовать его в качестве контекста

```
var context = { c: 2 };

function median (a,b) { return (a+b)/this.c; }
median(5,7); //error

median.call(context,5,7); // 6

median.apply(context,[5,7]); //6

//аналогично
context.func = median;
context.func(5,7)

var bound_median = median.bind(context);

bound_median(5,7); //6
```

////////////////////////////////////

// // Теперь давайте немного усложним // // В Array.prototype есть замечательный метод slice. // При вызове на массиве он возвращает копию массива // от начального индекса до конечного (исключительно)

```
[1,2,3].slice(0,1);
// => [1]
// Мы берем slice и присваиваем его локальной переменной
var slice = Array.prototype.slice;
// slice теперь оторван от контекста. Из-за того, что Array.prototype.slice
// работает с данным ему контекстом «this», метод больше не работает
slice(0, 1); // => TypeError: can't convert undefined to object
slice([1,2,3], 0, 1); // => TypeError: ... // Но мы можем использовать apply и call, о
ни позволяют нам передавать нужный контекст
slice.call([1,2,3], 0, 1); // => [1] // Apply работает как call, но принимает аргумент
ы в виде массива
slice.apply([1,2,3], [0,1]); // => [1]
```

//////////

```
this.x = 9;
var module = {
  x: 81,
  getX: function() { return this.x; }
};

module.getX(); // 81

var getX = module.getX;
getX(); // 9, поскольку в этом случае this ссылается на глобальный объект

// создаём новую функцию с this, привязанным к module
var boundGetX = getX.bind(module);
boundGetX(); // 81
```

////////////////////

// Немного надоедает использовать .call каждый раз. Может воспользоваться bind? //

Точно! Давайте привяжем функцию call к контексту slice.

```
var slice = Array.prototype.slice;
var call_func = Function.prototype.call;

big_slice = call_func.bind(slice);

big_slice = Function.prototype.call.bind(Array.prototype.slice);
// Теперь slice использует первый аргумент в качестве контекста
big_slice([1,2,3], 0, 1); // => [1]
```

Дополнительное чтение:

Функция bind своими руками <http://jsraccoon.ru/interview-bind>

Практика:

1. Превратить метод Draw из предыдущего раздела в функцию, которая получает на вход объект круга и отрисовывает его.
2. Есть функция, которая считает сумму своих аргументов. На ее основе получить функцию, которая принимает на вход массив и считает сумму его элементов.

Техника map-reduce

map

```
var numbers = [1, 4, 9];
var doubles = numbers.map(function(num) {
    return num * 2;
});
```

reduce

```
[0, 1, 2, 3, 4].reduce(function(previousValue, currentValue, currentIndex, array) {
    return previousValue + currentValue;
});
```

Преобразование HTMLCollection в Array

```
var arr = [].slice.call(htmlCollection);
```

filter

```
var words = ["spray", "limit", "elite", "exuberant", "destruction", "present"];

var longWords = words.filter(function(word){
    return word.length > 6;
});
```

Полезное чтение:

1. Map, Reduce, Filter <https://danmartensen.svbtle.com/javascripts-map-reduce-and-filter>

Практика:

1. Посчитать сумму чисел
2. Посчитать сумму квадратов чисел
3. Есть блоки. Сделать кнопки для одновременного изменения цвета, поворота, скругления, восстановления обратно
4. Есть массив чисел. С помощью функции filter получить массив простых чисел

Каррирование

Каррирование заменяет адаптер

```
pow(i, j);

function square(i) {
  return pow(i, 2);
}
```

```
function mul(a, b) {
  return a * b;
};

var double = mul.bind(null, 2); // контекст фиксируем null, он не используется
```

Говорят, что double является «частичной функцией» (partial function) от mul

Еще пример

```
var greet = function(greeting, name) {
  console.log(greeting + ", " + name);
};
greet("Hello", "Heidi"); // "Hello, Heidi"
```

```
var greetCurried = function(greeting) {
  return function(name) {
    console.log(greeting + ", " + name);
  };
};
```

```
var greetHello = greetCurried("Hello");
greetHello("Heidi"); // "Hello, Heidi"
greetHello("Eddie"); // "Hello, Eddie"
```

В общем виде функция каррирования выглядит вот так

```
function curry(func, a) {  
  return function (b) {  
    return func(b,a);  
  };  
}
```

Практика:

1. Получить из функции row, функцию 2^n
2. Получить из функции, которая рисует прямоугольник, функцию, которая рисует квадрат.
3. Есть функция func(a,b) - сделать функцию x, которую можно вызывать x(a)(b), но ее действие было бы аналогичным

Мемоизация

```
var fibonacci = (function () { // Self-executing anonymous function
  var memo = [0, 1];          // local variable within anonymous function
  var fib = function (n) {    // actual fib function (takes one argument)
    var result = memo[n];
    if (typeof result !== 'number') {
      result = fib(n - 1) + fib(n - 2);
      memo[n] = result;
    }
    return result;
  };
  return fib; // return fib (fibonacci is now set to the function fib //defined above, which takes one argument)
})();

for(var i = 0; i <= 10; i += 1) {
  document.writeln('// ' + i + ': ' + fibonacci(i));
}
```

//////////

```
var memoizer = function(memo, formula) {
  var recur = function(n) {
    var result = memo[n];
    if (typeof result !== 'number'){
      result = formula(recur, n);
      memo[n] = result;
    }
    return result;
  }
  return recur;
}

var fibonacci = memoizer([0, 1], function(recur, n){
  return recur(n-1)+recur(n-2);
});
```

Практика:

1. Построить функцию факториал через memoizer
2. Построить решение задачи про пешку через memoizer

Глава IX - Веб-приложения

Концепция

<https://medium.com/@franciov/installable-pinned-or-progressive-apps-5b4997ecbf49#.1vcg1xjgn>

https://wiki.mozilla.org/Firefox_OS/Pinned_Apps

Концепция для Firefox

<https://medium.com/@berbaquero/installable-web-apps-b48fdbcf5915#.3f4vp7j67>

Native Web Apps

<https://blog.andyet.com/2015/01/22/native-web-apps/>

"Инсталлируемое" веб-приложение

<https://medium.com/@franciov/how-to-make-your-web-app-installable-8b71571605e#.e16g8xv4k>

"Установка" под iPhone <http://brolik.com/blog/installable-web-apps-open-web/>

Кратко о текущем состоянии спецификации <http://html5doctor.com/web-manifest-specification/>

Спецификация от w3c

<https://w3c.github.io/manifest/>

<https://www.w3.org/TR/appmanifest/>

Спецификация для Opera <https://dev.opera.com/blog/installable-web-apps/>

Спецификация для Chrome

<https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp-manifest-in-chrome-38-for-Android>

<https://developer.chrome.com/multidevice/android/installtohomescreen>

Спецификация от Mozilla <https://developer.mozilla.org/en-US/Apps/Build/Manifest>

Спецификация для IE <https://dev.windows.com/en-us/microsoft-edge/platform/status/webapplicationmanifest>

Работа в оффлайне

Добавление на разные платформы

<http://android.stackexchange.com/questions/57064/quick-way-to-add-chrome-or-internet-shortcuts-to-home-screens>

<http://diveintohtml5.info/offline.html>

Добавление на андроид <https://habrahabr.ru/post/275849/>

Offline Cookbook <https://jakearchibald.com/2014/offline-cookbook/>

#Работа с камерой

<http://stackoverflow.com/questions/8581081/how-to-access-a-mobiles-camera-from-a-web-app> <http://www.html5rocks.com/en/tutorials/getusermedia/intro/>

<http://stackoverflow.com/questions/6336641/html5-camera-access-through-browser-in-ios>
https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/API/Camera_API/Introduction

Старый вариант, но рабочий <https://davidwalsh.name/browser-camera>

Вариант для Opera <https://dev.opera.com/articles/media-capture-in-mobile-browsers/>

Примеры https://www.kirupa.com/html5/accessing_your_webcam_in_html5.htm

<https://davidwalsh.name/html5-camera-video-iphone>

<http://www.webcodegeeks.com/html5/html5-web-camera-example/>

<https://hacks.mozilla.org/2013/02/cross-browser-camera-capture-with-getusermediawebrtc/>

#COFFEE WebRTC

<https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

http://www.html5rocks.com/en/tutorials/webrtc/basics/?redirect_from_locale=ru

////////////////////

Другие ссылки

<https://webrtc.github.io/samples/>

<http://www.html5rocks.com/en/tutorials/webrtc/basics/>

<https://developer.mozilla.org/en-US/docs/Web/Guide/API/WebRTC>

<https://webrtc.org/>

<http://shinydemos.com/qr-code/>

https://github.com/gasolin/qrcode_scanner

<http://franklinta.com/2014/10/19/serverless-webrtc-using-qr-codes/>

////////

<https://webqr.com/index.html> <https://github.com/LazarSoft/jsqrcode>

<https://github.com/sembrestels/angular-qr-scanner> <http://dwa012.github.io/html5-qrcode/>

<http://softwarerecs.stackexchange.com/questions/11971/javascript-qr-code-scanner-that-can-handle-800-bytes>

Service Workers

Services Workers и CacheStorage <https://habrahabr.ru/post/279291/>

Примеры Service Workers <https://github.com/w3c-webmob/ServiceWorkersDemos>

<https://serviceworker.rs/>

<https://github.com/w3c-webmob/ServiceWorkersDemos>

Проверить какие service workers запущены

```
chrome://serviceworker-internals/
```

Смотрим рабочий пример

<https://jakearchibald.com/2014/using-serviceworker-today/>

<http://blog.lamplightdev.com/2015/01/06/A-Simple-ServiceWorker-App/>

<https://davidwalsh.name/offline-recipes-service-workers>

Web Push

<https://web-push-book.gauntface.com>

<https://hacks.mozilla.org/2016/01/web-push-arrives-in-firefox-44/>

<https://mobiforge.com/design-development/web-push-notifications>

#Работа с файлами

<http://xdan.ru/working-with-files-in-javascript-part-4-object-urls.html>

<https://msdn.microsoft.com/en-us/library/windows/apps/hh453196.aspx>

COFFEE WebComponents

https://medium.com/@luisvieira_gmr/understanding-web-components-d051baa66019

<http://www.sitepoint.com/responsive-web-components/>

Глава X - NodeJS

Набор курсов, книг и другой информации по NodeJS

<http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js>

Плагины Sublime под NodeJS <http://scottksmith.com/blog/2014/09/29/3-essential-sublime-text-plugins-for-node-and-javascript-developers/>

Структура папок в проекте под NodeJS

<https://stackoverflow.com/questions/5178334/folder-structure-for-a-node-js-project>

<https://gist.github.com/lancejpollard/1398757>

COFFEE NodeJS Старт

Нам понадобится NodeJS <https://nodejs.org/en/> и NPM <https://www.npmjs.com/>

Для проверки установки вбейте в командную строку

```
node -help
```

и

```
npm -version
```

Обе эти команды должны срабатывать в любой директории на компьютере. Если же нет, то Вам нужно настроить переменные среды. Правый клик на "Мой компьютер" -> Дополнительные параметры -> Вкладка "Дополнительно" -> Переменные среды -> Нам нужна переменная Path

В Path через точку с запятой добавляем путь, где лежит nodejs и данные npm. У меня это `C:\nodejs\` и `C:\Users\Tinitilov\AppData\Roaming\npm` соответственно

Создаем файл hello.js

```
var message = "Hello World";  
console.log(message);
```

переходим в папку с этим файлом и запускаем его

```
node hello.js
```

В консоль должно вывестись Hello World

Теперь попробуем запустить простой локальный сервер. Создадим server.js с таким наполнением

```
var http = require('http');

var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Hello Http');
});

server.listen(8080);
```

Для запуска пишем в командной строке

```
node server.js
```

Если всё хорошо, у нас не должно быть ошибок. Окошко с запущенным nodejs закрывать не нужно. Проверим запуск сервера в браузере. Пропишем в адресной строке

```
localhost:8080
```

Чтобы остановить запущенный nodejs-сервер из командной строки пишем `process.exit(0)` или `Ctrl+Break` или `Ctrl+C`

Запустим сервер, который будет выводить сколько раз был запущен данный сайт

```
var http = require('http');
var i=0;

http.createServer(function (request, response) {
  i++;
  response.write("Page was opened "+i+" times");
  response.end();
}).listen(8080);
```

В данном случае счетчик увеличивается на 2, за счет того, что браузер посылает запрос на получение страницы и favicon

Подборка материалов <http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js>

Настройка Sublime для NodeJS

Настройка Build System

Выбираем: Tools > Build System > New Build System

Для Ubuntu

```
{  
  "cmd": ["/usr/local/bin/node", "$file"],  
}
```

Для Windows

```
{  
  "cmd": ["C:/Program Files/nodejs/node.exe", "$file"],  
  "selector": "source.js"  
}
```

Подробная статья <https://pawelgrzybek.com/javascript-console-in-sublime-text/>

Для открытия в командной строке в Windows

```
{  
  "cmd": ["node", "$file"],  
  "selector": "source.js",  
  "windows" : {  
    "shell": true  
  }  
}
```

Подробнее тема <http://stackoverflow.com/questions/20844421/sublime-text-3-build-system-node-js-npm-module-not-executing>

Перезапуск сервера при обновлении кода

```
npm install -g nodemon
```

Для проверки того, что nodemon установился вбиваем в командную строку

```
nodemon -h
```

Если не находит команду, то скорее всего проблема с путями прм

Заходим в Tools\Build System\New build system

Меняем build system на

```
{
  "cmd": ["nodemon", "$file"],
  "windows" : {
    "shell": true
  }
}
```

Для запуска Build System нажимаем Ctrl+Shift+B

Управление процессами

Если получаем

```
events.js:161
  throw er; // Unhandled 'error' event
    ^

Error: listen EADDRINUSE :::8081
```

Значит наше приложение уже запущено на порту 8081

Управление процессами под Linux <https://www.howtogeek.com/107217/how-to-manage-processes-from-the-linux-terminal-10-commands-you-need-to-know/>

Остановка NodeJS процесса

Получение id-процесса под Ubuntu

```
ps -e|grep node
```

```
kill -9 XXXX
```

Где XXXX - номер процесса

<http://serverfault.com/questions/256331/how-to-stop-node-js-server>

Настройка NodeJS на production'e <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-16-04>

Настройка виртуального домена для проекта

В папке проекта создаем config.json

```
{  
  "host": "site.local",  
  "port" : "8081",  
  "url": "http://site.local:8081",  
  "serverName": "site.local"  
}
```

Для редактирования файла хостов в Ubuntu

```
sudo nano /etc/hosts
```

Прописываем

```
127.0.0.1    site.local
```

<http://www.ainixon.me/how-to-create-virtual-hosts-in-ubuntu/>

Структура папок проекта

#Создание простейшего файл-сервера

Создадим файл страницы

index.html

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>NodeJS - это серверный javascript</title>
</head>

<body>
  NodeJS - это серверный javascript.
</body>
</html>
```

Добьемся, чтобы при запросе к серверу у нас открывался файл index.html . Нам понадобится модуль для работы с файлами fs

server.js

```
var http = require('http');
var fs = require('fs');

http.createServer(function (request, response) {
  fs.readFile('index.html', function (err, data) {
    if (err) {
      response.end("File wasn't found");
    }
    response.end(data);
  });
}).listen(8007);
```

Дополнительные материалы: <https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/>

<http://ru.code-maven.com/http-client-request-in-nodejs>

Практика:

1. Создайте трехстраничный сайт с навигационным меню

Работа с параметрами адресной строки

Дальнейшие действия потребуют модуль `url`. Модуль встроен в `nodejs`, так что его не нужно устанавливать с `nodejs`.

```
var url = require("url");
```

Работа с GET-параметрами

Допустим у нас есть строка

```
http://test.com?page=25&foo=bar
```

После вопроса у нас в строке находится два GET-параметра `page` и `foo`. Для их получения нам понадобится метод `url.parse`. На выходе мы получаем объект `parts`, у которого будет два свойства `parts.query.page` и `parts.query.foo` со значениями параметров `page` и `query`

```
var url = require("url");
var parts = url.parse("http://test.com?page=25&foo=bar", true);
parts.query.page++; // увеличение значения GET-параметра page на единицу
```

Если же мы хотим получить GET-параметры из текущего `url`, то нам понадобится свойство `request.url` (`request` - это объект запроса в сервере)

Попробуем создать `echo`-сервер - то есть сервер, который будет возвращать значение GET-параметра, который мы передали

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
  var parts = url.parse(req.url, true);
  res.writeHead(200);
  res.end(parts.query.name); //выводим GET-параметр name
});

server.listen(8080);
```

Для проверки запускаем сервер и вводим в адресную строку браузера

```
localhost:8080/?name=petya
```

Должна вывестись страница с именем petya

Работа с pretty-urls

Иногда нам нужно считывать не GET-параметры, а извлекать путь после доменного имени

Например, у нас есть адрес

```
http://localhost:8080/check?login=petya
```

Нам нужно достать из этого адреса слово */check* . В этом нам поможет строчка

```
var pathname = url.parse(request.url).pathname;
```

Дополнительные материалы:

https://millermedeiros.github.io/mdoc/examples/node_api/doc/url.html

Практика:

1. Создать echo-сервер. Скрипт считывает значение какого-то GET-параметра и выводит его значение.
2. Сделать сервер, который получает на вход два параметра a и b и выводит их сумму.
3. Сделать сервер, который выводит форму с запросом логина и пароля и кнопкой отправки. После отправки данных проверяем смог пользователь залогиниться или нет - выводим соответствующие сообщения.
4. Реализовать игру угадай число. Есть форма, в которую пользователь вводит число. Есть три возможных ситуации: 1) он угадал число, тогда программа его поздравляет сообщением, 2) число пользователя меньше - выводим фразу "Меньше" и форму для ввода нового числа. 3) число пользователя больше - выводим фразу "Больше" и форму для ввода нового числа.

POST-параметры

Когда мы создаем сервер у нас создаются два потока, так называемые Stream. Request или req является Readable Stream. Когда нам приходит часть данных, у Readable Stream срабатывает событие *data*. Когда все части данных присланы происходит событие *end*

```
var http = require('http');
var fs   = require('fs');
var qs   = require('querystring');

http.createServer(function (req, res) {
  if (req.method == 'POST') {
    var body = '';

    req.on('data', function (data) {
      body += data;
    });

    req.on('end', function () {
      var obj = qs.parse(body)
      console.log(obj.test);
    });
  } else {
    fs.readFile('form.html', function (err, data){
      if (err) {
        res.end("File wasn't found");
      }
      res.writeHead(200, {"Content-Type": "text/html"});
      res.end(data);
    });
  }
}).listen(8080);
```

файл form.html

```
<form action="/" method="POST">
  <input type="text" name="test">
  <input type="submit" value="Отправить">
</form>
```

Иногда нам нужно избавиться от POST-параметров, чтобы при обновлении страницы пользователю не выводилось сообщение.

```
response.writeHead(302, {
  'Location': 'your/404/path.html'
  //add other headers here...
});
response.end();
```

Перенаправление на стороне сервера

Чтобы избавиться от POST-параметров делаем перенаправление на стороне сервера

```
res.writeHead(302, {'Location': '/ok'});
res.end();
```

Установка кодировки

```
res.writeHead(200, {"Content-Type": "text/html"});
```

Построение echo-сервера на NodeJS

<https://debugmode.net/2014/01/14/create-echo-server-in-node-js/>

Дополнительные материалы:

Как извлечь POST-данные в NodeJS <http://stackoverflow.com/questions/4295782/how-do-you-extract-post-data-in-node-js>

<https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/>

<http://ru.code-maven.com/http-client-request-in-nodejs>

Запрос к удаленному серверу

<https://docs.nodejitsu.com/articles/HTTP/cients/how-to-create-a-HTTP-request/>

Практика:

1. Есть форма для ввода логина, пароля. Если и логин и пароль совпали - вывести пользователю поздравление. Если нет, выводим сообщение о просьбе ввести данные повторно и форму для ввода.
2. Пользователь вводит число в форму. В ответ ему опять выводится форма. Так происходит до тех пор, пока пользователь не введет 0. После этого выводим сумму всех введенных чисел.

Загрузка файла на сервер

<http://stackoverflow.com/questions/15772394/how-to-upload-display-and-save-images-using-node-js-and-express>

Документация по Multer

<https://github.com/expressjs/multer>

Мульти-загрузка

<https://codeforgeek.com/2016/01/multiple-file-upload-node-js/>

<https://coligo.io/building-ajax-file-uploader-with-node/>

<https://habrahabr.ru/post/229743/>

еще <https://codeforgeek.com/2014/11/file-uploads-using-node-js/>

Angular + NodeJS

<http://code.ciphertrick.com/2015/12/07/file-upload-with-angularjs-and-nodejs/>

Cookies

Установка cookie

<http://stackoverflow.com/questions/3393854/get-and-set-a-single-cookie-with-node-js-http-server>

```
var http = require('http');

function parseCookies (request) {
  var list = {},
      rc = request.headers.cookie;

  rc && rc.split(';').forEach(function( cookie ) {
    var parts = cookie.split('=');
    list[parts.shift().trim()] = decodeURI(parts.join('='));
  });

  return list;
}

http.createServer(function (request, response) {

  // To Read a Cookie
  var cookies = parseCookies(request);

  // To Write a Cookie
  response.writeHead(200, {
    'Set-Cookie': 'mycookie=test',
    'Content-Type': 'text/plain'
  });
  response.end('Hello World\n');
}).listen(8124);

console.log('Server running at http://127.0.0.1:8124/');
```

Expired/Max-age

<http://stackoverflow.com/questions/7374042/how-can-i-set-a-cookies-expiration-in-nodejs>

<http://www.connecto.io/blog/nodejs-express-how-to-set-multiple-cookies-in-the-same-response-object/>

Генерация хеша


```
require('crypto').randomBytes(48, function(err, buffer) {  
  var token = buffer.toString('hex');  
});
```

<https://stackoverflow.com/questions/8855687/secure-random-token-in-node-js>

Практика:

1. Установить куку. Проверить ее установку в браузере
2. Сделать страницу, которая подсчитывает сколько раз пользователь ее посетил
3. Есть круг. Пользователь выбирает цвет круга
4. Пользователь выбирает цвет и радиус круга. Сохраняем настройки в куках.
5. Сделать программу, которая запоминает пользователя

#События

events EventEmitter

<https://www.sitepoint.com/nodejs-events-and-eventemitter/>

<http://www.hacksparrow.com/node-js-eventemitter-tutorial.html>

```
var events = require('events');
var EventEmitter = new events.EventEmitter();

eventEmitter.on('myCustomEvent', function(arg1,arg2)) {
  console.log(arg1+arg2);
}

setTimeout(function() {
  eventEmitter.emit('myCustomEvent', 'String1', 'and String2');
}, 2000);
```

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
myEmitter.emit('event');
```

Дополнительные материалы: https://www.youtube.com/watch?v=Fguk1wB9Ob8&index=12&list=PLYxzS__5yYQmHbpKMARP04F344zYRX91I

Практика:

1. Сделать, чтобы страница при каждом десятом запуске поздравляла пользователя.
Использовать свое кастомное событие
2. Есть три чекбокса. Когда нажаты 2 чекбокса из 3 - поздравляем пользователя.
3. Проверка логина-пароля

Модуль fs. Работа с файлами

<https://github.com/visionmedia/masteringnode/blob/master/chapters/fs.md>

<http://ru.code-maven.com/list-content-of-directory-with-nodejs>

Считываем данные с директории

```
fs.readdir(path, function(err, items) {
  for (var i=0; i<items.length; i++) {
    var file = path + '/' + items[i];

    console.log("Start: " + file);
    fs.stat(file, generate_callback(file));
  }
});

function generate_callback(file) {
  return function(err, stats) {
    console.log(file);
    console.log(stats["size"]);
  }
};
```

Считывание файла по строкам: <http://stackoverflow.com/questions/6156501/read-a-file-one-line-at-a-time-in-node-js>

```
var lineReader = require('readline').createInterface({
  input: require('fs').createReadStream('file.in')
});

lineReader.on('line', function (line) {
  console.log('Line from file:', line);
});
```

Получение подробной информации о файле <http://ru.code-maven.com/system-information-about-a-file-or-directory-in-nodejs>

Практика:

1. Считать из файла два числа. Найти их сумму
2. В файле в первой строчке заданы размеры карты m n. Далее, в последующих m строках указаны n символов, кодирующие ячейки карты. * - обычная ячейка, # - сундук. Требуется найти координаты сундука.

3. Считать данные с удаленного сайта и записать их в файл (например, файл погоды)

#Модули и require

<https://nodejs.org/docs/v0.4.4/api/modules.html>

Подключим к нашему файлу модуль circle.js . Он находится в той же директории.

```
var circle = require('./circle.js');
console.log( 'The area of a circle of radius 4 is '
    + circle.area(4));
```

Для создания модуля нам понадобится следующий код

```
var PI = Math.PI;

exports.area = function (r) {
    return PI * r * r;
};

exports.circumference = function (r) {
    return 2 * PI * r;
};
```

Если мы хотим сделать доступной переменную или метод - добавляем ее в exports.

Все локальные переменные в модули будут private-переменными.

О модулях https://www.youtube.com/watch?v=e1Ln1FrLv8&list=PLYxzS__5yYQmHbpKMARP04F344zYRX91I&index=3

Дополнительные материалы:

<https://darrenderidder.github.io/talks/ModulePatterns>

Практика:

1. Написать модуль, который возвращает блок, заданного размера с фоновым изображением.

NPM

Инициация package.json

package.json - файл, который хранит в себе всю необходимую информацию о Вашем модуле

```
{
  "name": "bingo",
  "version": "0.0.1",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "dmitry tinitilov",
  "license": "MIT"
}
```

Для того, чтобы сгенерировать package.json с помощью npm необходимо запустить

```
npm init
```

При этом нужно будет ответить на вопросы npm, либо нажать несколько раз Enter

Автоматическая генерация package.json выполняется с помощью

```
npm init --yes
```

Добавление зависимостей

Если для нашего модуля нужен jQuery, с добавлением в зависимости, то используем команду

```
npm install jquery --save
```

После чего в наш package.json добавиться свойство *dependencies*

```
{
  "name": "bingo",
  "version": "0.0.1",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "dmitry tinitilov",
  "license": "MIT",
  "dependencies": {
    "jquery": "^3.2.1"
  }
}
```

Для сборки проекта нам также нужны модули. Для их установки используем команду `install` с ключем `--save-dev`

```
npm install --save-dev babel-core
```

Запуск настраиваемых скриптов

В свойстве `scripts` `package.json` можно добавлять свои команды, например

```
"scripts": {
  "build": "webpack"
},
```

Предварительно нужно проинсталлировать `webpack`

```
npm install webpack
```

После его настройки можно будет запустить команду

```
npm run build
```

Публикация модуля

Регистрация или залогинивание

```
npm adduser
```

Публикация модуля из текущей директории

```
npm publish
```

Поиск модуля

```
npm search your_module
```

Установка модуля

```
npm install your_module
```

Удаление модуля из текущей папки

```
npm remove your_module
```

Удаление модуля из репозитория NPM

```
npm unpublish
```

Введение в NPM

<https://www.youtube.com/watch?v=fhwtUW9dXrA&list=PLDyvV36pndZFWfEQpNixIHVvp191Hb3Gg&index=7>

Структура пакета NPM

<https://www.youtube.com/watch?v=CrevZgTc7ow&list=PLDyvV36pndZFWfEQpNixIHVvp191Hb3Gg&index=8>

Создание собственного NPM-модуля

<https://medium.com/@jdaudier/how-to-create-and-publish-your-first-node-js-module-444e7585b738#.49e5dlo31>

```
npm init
```

<https://docs.npmjs.com/cli/init>

O package.json

<http://browsenpm.org/package.json>

<https://docs.npmjs.com/getting-started/using-a-package.json>

NPM для новичков

<https://habrahabr.ru/post/243335/>

Подготовка собственного пакета

<https://habrahabr.ru/post/206678/>

Более тонкая настройка NPM <http://prgssr.ru/development/vvedenie-v-paketnyj-menedzher-npm-dlya-nachinayushih.html>

Практика:

1. Сделать модуль, который по массиву чисел возвращает гистограмму

async/await

Полезное чтение:

1. <https://medium.com/@tmvvr/ecmascript-async-await-to-the-rescue-fc379ff89146>
2. <https://habrahabr.ru/company/ruvds/blog/326074/>
3. <http://thecodebarbarian.com/common-async-await-design-patterns-in-node.js.html>
4. <https://habrahabr.ru/post/282477/>

Process

Полезное чтение:

1. <https://nodejs.org/api/process.html>
2. Переменные среды USER_ID, USER_KEY
<https://stackoverflow.com/questions/22312671/node-js-setting-environment-variables>
3. Переменные среды <https://www.twilio.com/blog/2017/08/working-with-environment-variables-in-node-js.html>

Основы Express

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

Установка

```
$ npm install express --save
```

Вспомогательные модули

```
$ npm install body-parser --save  
$ npm install cookie-parser --save  
$ npm install multer --save
```

```
var express = require('express');  
var app = express();  
  
app.get('/', function (req, res) {  
  res.send('Hello World');  
})  
  
var server = app.listen(8081, function () {  
  
  var host = server.address().address  
  var port = server.address().port  
  
  console.log("Example app listening at http://%s:%s", host, port)  
  
})
```

res.send() - заканчивает отдачу данных

Подробнее о response object

https://www.tutorialspoint.com/nodejs/nodejs_response_object.htm

Подробнее о request object

https://www.tutorialspoint.com/nodejs/nodejs_request_object.htm

Строим файл-сервер на express

```
var express = require('express');
var app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

Если мы добавим картинку в директорию public, то сможем к ней обращаться через localhost:8081/picture.jpg

Ответ содержимым файла

Научимся отдавать файлы на запрос пользователя

```
res.sendFile(__dirname+'path/to/your/file.html')
```

Генерация файла построчно

```
app.get('/adduser', function (req, res){
  res.setHeader('content-type', 'text/html; charset=utf-8');
  res.write('<form action="adduser" method="GET">');
  res.write('<input type="text" name="username">');
  res.write('<input type="submit">');
  res.write('</form>');
  res.end();
})
```

Полезное чтение

1. Мануал по Express <http://jsman.ru/express/>
2. Подключение статических файлов <http://expressjs.com/ru/starter/static-files.html>
3. Генерация проекта-заготовки на Express https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website

Практика:

1. Сделать галерею картинок. При клике на картинку, мы переходим на нее

<http://expressjs.com/ru/guide/routing.html>

<http://expressjs.com/ru/guide/writing-middleware.html>

Написание собственного middleware

<https://codeforgeek.com/2015/12/how-to-write-custom-middleware-for-expressjs/>

<https://scotch.io/tutorials/use-expressjs-to-get-url-and-post-parameters>

<http://stackoverflow.com/questions/20089582/how-to-get-url-parameter-in-express-node-js>

?tagId=5 use req.query

```
app.get('/p', function(req, res) {  
  res.send("tagId is set to " + req.query.tagId);  
});
```

для ЧПУ

```
app.get('/p/:tagId', function(req, res) {  
  res.send("tagId is set to " + req.params.tagId);  
});
```

Практика:

1. Сделать скрипт, который находит сумму двух чисел

<http://stackoverflow.com/questions/24330014/bodyparser-is-deprecated-express-4>

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: true
}));
```

Установка header'a

<http://stackoverflow.com/questions/23751914/how-can-i-set-response-header-on-express-js-assets>

Перенаправление

Проблема POST-параметров в том, что при перезагрузке страницы, пользователю выведется "пугающее" сообщение, о том, что введенная им информация будет повторно использована.

Не стоит оставлять пользователя в таком положении и следует перенаправить его на новую страницу. По пути POST-параметры будут потеряны, что даст возможность пользователю нормально перезагружаться.

Для перенаправления используем `res.redirect`

```
res.redirect('/customers/');
```

<http://stackoverflow.com/questions/22677940/difference-between-location-and-redirect-in-node-js>

Практика:

1. Есть форма для ввода логина-пароля. Нужно отправить данные методом POST и проверить их правильность

Загрузка файла на NodeJS сервер, используя Express

<https://howtonode.org/really-simple-file-uploads>

<https://www.npmjs.com/package/multer>

Для реализации понадобится установка модуля multer

```
npm i multer
```

И создание папки uploads

```
var express = require('express');
var app = express();

var fs = require('fs');

var multer = require('multer')
var upload = multer({ dest: 'uploads/' })

app.get('/', function (req, res) {
  res.setHeader('content-type', 'text/html;charset=utf-8');
  res.write('<link rel="stylesheet" href="css/style.css">');
  res.write('<form action="/upload" method="POST" enctype="multipart/form-data">');
  res.write('<input type="file" name="avatar">');
  res.write('<input type="submit">');
  res.write('</form>');
  res.end();
})

app.post('/upload', upload.single('avatar'), function (req, res, next) {
  console.log(req.file);

  fs.rename(req.file.path, 'uploads/'+req.file.originalname, function (err) {
    if (err) throw err;
    console.log('renamed complete');
  });

  res.end('Gotcha');
  // req.file is the `avatar` file
  // req.body will hold the text fields, if there were any
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

Практика:

1. Есть страница, на которой выводятся все загруженные фотографии. Сделать, чтобы у пользователя была возможность удалить указанную им фотографию (рядом с каждой картинкой, есть кнопка удалить)

Установка Cookies

Хорошая статья по установке Cookies на Express

<https://www.codementor.io/nodejs/tutorial/cookie-management-in-express-js>

Нам понадобится модуль cookie-parser

Подробнее о модуле можно прочитать по ссылке

<https://www.npmjs.com/package/cookie-parser>

Подключение модуля cookie-parser

```
npm install cookie-parser
```

После установки подключим его в наш код

```
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();
app.use(cookieParser());
```

Установка куки

Далее в коде нашего приложения попробуем установить нашу куку

```
app.get('/cookie', function(req, res){
    res.cookie('cookie_name' , 'cookie_value').send('Cookie is set');
});
```

Считывание куки

```
app.get('/', function(req, res) {
    console.log("Cookies : ", req.cookies);

    console.log("Cookies : ", req.cookies.cookie_name);
});
```

Установка куки на длительный срок

вариант со сроком годности

```
res.cookie(name , 'value', {expire : new Date() + 9999});
```

вариант с временем жизни. Обратите внимание, что 9999 - это миллисекунды, то есть это всего лишь 10 секунд!

```
res.cookie(name, 'value', {maxAge : 9999});
```

Удаление cookie

```
app.get('/clearcookie', function(req,res){  
  res.clearCookie('cookie_name');  
  res.send('Cookie deleted');  
});
```

Практика:

1. Показывать пользователю, сколько раз он загрузил страницу
2. Есть форма для ввода логина, пароля с флажком "запомнить меня". Если пользователь ввел данные правильно, мы должны "залогинить" его, то есть установить куку с каким-то специальным значением. Должна быть отдельная страница, на которой мы можем проверить залогинен ли пользователь.

Unit-тесты

<http://metanit.com/web/nodejs/5.1.php>

```
var operations = require("./operations");

it("should multiply two numbers", function(){

    var expectedResult = 15;
    var result = operations.multiply(3, 5);
    if(result!==expectedResult){
        throw new Error(`Expected ${expectedResult}, but got ${result}`);
    }
});
```

Подключим какую-то assertion-library, например chai

```
npm i chai --save-dev
```

Подключим в наш test.js

```
var assert = require('chai').assert
```

```
describe("multitests", function() {

    function makeTest(x) {
        var expected = x * x * x;
        it("при возведении " + x + " в степень 3 результат: " + expected, function() {
            assert.equal(operations.pow(x, 3), expected);
        });
    }

    for (var x = 1; x <= 5; x++) {
        makeTest(x);
    }

});
```

<http://chaijs.com/api/bdd/>


```
expect(function () {}).to.not.throw();  
expect({a: 1}).to.not.have.property('b');  
expect([1, 2]).to.be.an('array').that.does.not.include(3);
```

Подробнее: <https://learn.javascript.ru/testing>

Pug

Установим Pug в наш проект

```
npm install pug
```

Создадим базовый шаблон в папке views

```
html
  head
    link(href='/css/style.css' rel='stylesheet')
  body Hello World!
```

Подключим pug к проекту

```
app.set('view engine', 'pug');

app.get('/test_render', function(req, res){
  res.render('test_render', {});
})
```

Передача переменной в шаблон

```
app.get('/test_render', function(req, res){
  res.render('test_render', {message: 'big start starts here'});
})
```

```
html
  head
    link(href='/css/style.css', rel='stylesheet')
  body Hello World!
  h1= message
```

Подключение CSS к шаблону

```
head
  title First Express App
  link(href='/css/style.css' rel='stylesheet')
```

Пример простой формы на Pug

```
form(method=GET action='/authorize')
  input(name='login' placeholder='login')
  input(name='password' placeholder='password')
  input(type='submit' value='зalogиниться')
```

или

```
input(name='caption' placeholder='caption')
textarea(name='description' placeholder='description')
button(class='submit_button') Добавить комментарий
```

Работа с массивами внутри Pug

Передаем переменную `listOfElements` и работаем с ней через `each`

```
ul
  each element in listOfElements
    li= element.name
```

Наследование шаблонов в Pug

<https://pugjs.org/language/inheritance.html>

<https://pugjs.org/language/includes.html>

Пример с официального сайта

```
// - index.pug
doctype html
html
  include includes/head.pug
  body
    h1 My Site
    p Welcome to my super lame site.
    include includes/foot.pug
```

```
// - includes/head.pug
head
  title My Site
  script(src='/javascripts/jquery.js')
  script(src='/javascripts/app.js')
```

```
//- includes/foot.pug
footer#footer
  p Copyright (c) foobar
```

Интересное чтение:

1. Полезная статья о pug, которая заставит Вас улыбнуться <https://codepen.io/mimoduo/post/learn-pug-js-with-pugs>
2. Использование шаблонизаторов в Express <http://expressjs.com/ru/guide/using-template-engines.html>
3. Введение в Jade <https://webapplog.com/jade/>
4. Handlebars <https://www.packtpub.com/books/content/using-handlebars-express>
5. Использование шаблонизаторов в express <https://webapplog.com/jade-handlebars-express/>

MongoDB

Учебник по Mongo <http://jsman.ru/mongo-book/>

MongoDB Старт

Установка MongoDB под Ubuntu <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>

Находим папку с файлом mongod.exe и пишем

```
mongod --dbpath D:\mongodb\data --port 27017 --storageEngine=mmapv1
```

или

```
mongod --storageEngine=mmapv1 --dbpath [your-path]
```

--port - задает порт, на котором будет работать mongodb. По умолчанию это 27017

--dbpath - задает путь к данным БД

--storageEngine - задает какой из движков БД будет запущен. Под Win32 без патчей идет только старый движок mmapv1

Если запускаться с каким-то конфигурационным файлом

```
"C:\mongodb\bin\mongod.exe" --config "C:\mongodb\mongod.cfg"
```

Инсталляция через npm

```
npm install mongodb
```

Тестируем подключение в nodejs на этой строчке

```
var mongo = require('mongodb');
```

Пробуем несколько расширенный вариант

```
var mongo = require('mongodb');
var host = 'localhost';
var port = mongo.Connection.DEFAULT_PORT;

var db = new mongo.Db('test', new mongo.Server(host, port, {}), {safe:false});
db.open(function(err, db) {
  console.log("Connected!");
  db.close();
});
```

Иногда возникают проблемы со конструкцией `mongo.Connection.DEFAULT_PORT`, тогда вместо нее ставим `27017`

Пробуем добавить что-то в базу данных, а потом считать

```
var mongo = require('mongodb');
var host = 'localhost';
var port = mongo.Connection.DEFAULT_PORT;

var db = new mongo.Db('test', new mongo.Server(host, port, {}), {safe:false});
db.open(function(err, db) {
  console.log("Connected!");

  var collection = db.collection("simple_collection");
  collection.insert({hello:'world'});

  collection.findOne({hello:'world'}, function(err, item) {
    console.log(item);
    db.close();
  })
});
```

Организация внешнего файла с подключением к БД

Создаем `db.js`

```
var MongoClient = require('mongodb').MongoClient

var state = {
  db: null,
}

exports.connect = function(url, done) {
  if (state.db) return done()

  MongoClient.connect(url, function(err, db) {
    if (err) return done(err)
    state.db = db
    done()
  })
}

exports.get = function() {
  return state.db
}

exports.close = function(done) {
  if (state.db) {
    state.db.close(function(err, result) {
      state.db = null
      state.mode = null
      done(err)
    })
  }
}
```

И в дальнейшем подключаем db.js к нужному нам файлу

```
var db = require('./db')
```

Полезное чтение:

1. Основы по MongoDB <http://jsman.ru/mongo-book/Glava-1-Osnovy.html>
2. Подключение MongoDB к NodeJS <http://stepansuvorov.com/blog/2012/10/node-js-%D0%B8-mongodb/>
3. Организация проекта на NodeJS, MongoDB <https://www.terlici.com/2015/04/03/mongodb-node-express.html>

Полезные ссылки:

////////////////////////////////

<http://nodeguide.ru/doc/dailyjs-nodepad/node-tutorial-2/#mongodb>

Минимануал по установке http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/?_ga=1.195041452.1278229924.1436390555

Мануал по Mongo <http://express-js.ru/mongo-book/>

////////////////////////////////////

Практика:

1. Есть форма для ввода слов. Под ней выводим ранее добавленные слова

Настройка Mongo GUI

Скачиваем RoboMongo <https://robomongo.org/>

MongoUI <https://webapplog.com/mongoui/>

Mongo Compas <https://www.mongodb.com/products/compass>

MongoDB CRUD

<https://mongodb.github.io/node-mongodb-native/api-generated/collection.html>

Пример Create запроса

```
collection.insert({hello:'world'});
```

Пример Read запроса

```
db.products.find( { qty: { $gt: 25 } }, { item: 1, qty: 1 } )
```

Если мы используем `find`, то нам могут вернуть несколько значений. Пройтись по ним можно через `cursor`

```
collection.find().toArray(function(err, items) {  
  console.log(items);  
  for (i=0;i<items.length;i++) {  
    res.write((items[i]._id).toString());  
  }  
  res.end();  
  db.close();  
})
```

Очень подробно про `find` запросы <https://habrahabr.ru/post/134590/>

Пример Update запроса

Первый параметр задает запрос, по которому выбираются записи. Второй параметр задает вариант новой записи, которая придет на замену старой.

Если нам не нужна замена старой записи, а просто ее модификация, то нужно использовать параметры `$inc` и `$set`. `$inc` - увеличивает указанные поля на указанные значения. `$set` - устанавливает указанным полям новые значения, неуказанные поля остаются без изменения.

```
db.books.update(
  { _id: 1 },
  {
    $inc: { stock: 5 },
    $set: {
      item: "ABC123",
      "info.publisher": "2222",
      tags: [ "software" ],
      "ratings.1": { by: "xyz", rating: 3 }
    }
  }
)
```

Пример Delete запроса

```
db.products.remove( { qty: { $gt: 20 } }, true )
```

true говорит о том, что мы удаляем одну запись

Удаление элемента по его id

```
var ObjectId = require('mongodb').ObjectId;

db.test_users.remove( {"_id": ObjectId("4d512b45cc9374271b02ec4f")} );
```

Получение информации по структуре БД

```
db.users.findOne({_id:ObjectId(...)})
```

Полезное чтение:

1. Что выбрать множественные коллекции или вложенные документы
<http://openmymind.net/Multiple-Collections-Versus-Embedded-Documents>

Практика:

1. Реализуйте гостевую книгу. То есть, есть форма для добавления комментариев. Под ней выводятся сами комментарии.

Limit,skip,count,sort

limit,skip

```
db.unicorns.find().sort({weight: -1}).limit(2).skip(1)
```

<http://stackoverflow.com/questions/22441482/order-and-limit-results-in-a-query-with-a-callback>

```
myModel.find(filter)
  .limit(pageSize)
  .skip(skip)
  .sort(sort)
  .toArray(callback);
```

или такой вариант

```
myModel.find(filter, {sort: {created_at: -1}, limit: 10}, function(err, items){
});
```

Обработка результата

```
var found_people_promise = people_collection.find().limit(-1).skip(rnd).next();

found_people_promise.then(function(people){
  db.close();
  console.log(people);
  res.end();
}, function(err) {
  console.log('Find random people error');
  res.end();
})
```

Подсчет количества элементов на странице

```
db.collection.count()
```

Вариант с callback'ом

```
Model.count({email: 'xyz@gmail.com'}, function (err, count) {  
  console.log(count);  
});
```

Сортировка

```
db.collection.find({ ... spec ... }).sort({ key: 1 })
```

где 1 - сортировка по возрастанию -1 сортировка по убыванию.

вариант через только через find

```
db.collection.find( { $query: {}, $orderby: { column : -1 } } )
```

Практика:

1. Сделать пагинацию на сайте
2. Вывести самый дорогой товар

<https://habrahabr.ru/post/210760/>

NPM-модуль для bcrypt

<https://www.npmjs.com/package/bcrypt-nodejs>

Fulltext search

Полнотекстовый поиск на MongoDB

<https://code.tutsplus.com/ru/tutorials/full-text-search-in-mongodb--cms-24835>

```
db.articles.find( { $text: { $search: "\"coffee shop\"" } } )
```


Геолокация

<https://medium.com/@roberto.b/geolocation-csv-mongodb-and-compass-3918889c1474>

<https://habrahabr.ru/company/xakep/blog/143909/>

COFFEE making grabber

<http://frontender.info/web-scraping-with-nodejs/>

Создаем чат на NodeJS

<https://habrahabr.ru/post/200866/>

Чат на Socket.io <https://habrahabr.ru/post/127525/>

```
// создаем сервер
var WebSocketServer = require('ws').Server,
    wss = new WebSocketServer({port: 9000});

// соединение с БД
var MongoClient = require('mongodb').MongoClient,
    format = require('util').format;

var userListDB, chatDB;

// подключаемся к БД
MongoClient.connect('mongodb://127.0.0.1:27017', function (err, db) {
  if (err) {throw err}

  // записываем ссылки на таблицы (коллекции) в глобальные переменные
  userListDB = db.collection('users');
  chatDB = db.collection('chat');
});
```

MEAN

<https://habrahabr.ru/company/piter/blog/279237/>

Паттерны

<https://habrahabr.ru/company/piter/blog/278017/>

WebSockets

<http://hungtran.co/long-polling-and-websockets-on-nodejs/>

<http://stackabuse.com/node-js-websocket-examples-with-socket-io/>

<http://ahoj.io/nodejs-and-websocket-simple-chat-tutorial>

```
var WebSocketServer = require('websocket').server;
var http = require('http');

var server = http.createServer(function(request, response) {
  // process HTTP request. Since we're writing just WebSockets server
  // we don't have to implement anything.
});
server.listen(1337, function() { });

// create the server
wsServer = new WebSocketServer({
  httpServer: server
});

// WebSocket server
wsServer.on('request', function(request) {
  var connection = request.accept(null, request.origin);

  // This is the most important callback for us, we'll handle
  // all messages from users here.
  connection.on('message', function(message) {
    if (message.type === 'utf8') {
      // process WebSocket message
    }
  });

  connection.on('close', function(connection) {
    // close user connection
  });
});
```

Материалы для чтения по NodeJS

Современный JavaScript для доисторических Web-разработчиков

<https://trackchanges.postlight.com/modern-javascript-for-ancient-web-developers-58e7cae050f9#.q6wo0jasc>

Создание бота на NodeJS <http://www.girliemac.com/blog/2016/10/24/slack-command-bot-nodejs/>

Глава XI - Angular

Неплохие уроки по Angular <http://monsterlessons.com/project/categories/angularjs>

Angular Старт

Скачиваем angular.js (посмотрите обязательно исходный код этой библиотеки) с <https://angularjs.org/> и пробуем запустить данное приложение

```
<!DOCTYPE html>
<html>
<script src="angular.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

ng-init

Директива ng-init позволяет устанавливать начальные значения переменным. Например:

```
ng-init="firstName='John'"
```

```
<div ng-app="" ng-init="quantity=1;cost=5">
  <p>Total in dollar: {{ quantity * cost }}</p>
</div>
```

Директива ng-bind позволяет "привязать" переменную к html-коду

```
<div ng-app="" ng-init="quantity=1;cost=5">
  <p>Total in dollar: <span ng-bind="quantity * cost"></span>    </p>
</div>
```

Значение переменной может быть объектом

```
<div ng-app="" ng-init="person= {firstName:'John',lastName:'Doe'}">
  <p>The name is {{ person.lastName }}</p>
</div>
```

Практика:

1) Вывести прямоугольник по вводимым координатам

ng-repeat

Директива ng-repeat позволяет проходиться по элементам массива в переменной

```
<div ng-app="" ng-init="names=['Jani','Hege','Kai']">
  <ul>
    <li ng-repeat="x in names">
      {{ x }}
    </li>
  </ul>
</div>
```

В случае прохода по элементам объекта как по ассоциативному массиву, синтаксис становится следующим:

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

```
<tr ng-repeat="(key, value) in data">
  <td>{{key}}<input type="text" ng-model="data[key]"></td>
</tr>
```

Пример для случая многомерных массивов

```
<ul>
  <li ng-repeat="category in categories">
    {{ category.title }}
    <ul ng-if="category.categories">
      <li ng-repeat="category in category.categories">
        {{ category.title }}
      </li>
    </ul>
  </li>
</ul>
```

Практика:

1. Вывести несколько div'ов в цикле

Модули

http://www.w3schools.com/angular/angular_modules.asp

```
<div ng-app="myApp">...</div>

<script>

var app = angular.module("myApp", []);

</script>
```

Контроллеры

http://www.w3schools.com/angular/angular_controllers.asp

Пример простого контроллера

```
<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
});
</script>
```

Еще один пример

```
<div ng-app="" ng-controller="myCtrl">

<button ng-click="count = count + 1">Click me!</button>

<p>{{ count }}</p>

</div>
```

```
<body ng-controller="MainCtrl">
  <p>Hello {{name}}!</p>
  {{msg}}
  <a ng-href='#here' ng-click='go()' >click me</a>
  <div style='height:1000px'>

    <a id='here'></a>

  </div>
  <h1>here</h1>
</body>
```

```
var app = angular.module('plunker', []);

app.controller('MainCtrl', function($scope) {
  $scope.name = 'World';

  $scope.go = function() {

    $scope.msg = 'clicked';
  }
});
```

ng-click вместе ng-hide

```
<div ng-app="myApp" ng-controller="personCtrl">

  <button ng-click="toggle()">Toggle</button>

  <p ng-hide="myVar">
    First Name: <input type="text" ng-model="firstName"><br>
    Last Name: <input type="text" ng-model="lastName"><br>
    <br>
    Full Name: {{firstName + " " + lastName}}
  </p>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
  $scope.firstName = "John",
  $scope.lastName = "Doe"
  $scope.myVar = false;
  $scope.toggle = function() {
    $scope.myVar = !$scope.myVar;
  };
});
</script>
```

Примеры:

5-ть практических примеров использования Angular

<http://tutorialzine.com/2013/08/learn-angularjs-5-examples/>

Практика:

1. Сделать кликер
2. Сделать блок-мигалку (при клике переключает цвет с первого на второй, со второго на первый)
3. Сделать проверку данных из формы для ввода логина-пароля
4. Сделать приложение - список покупок

Сервисы

http://www.w3schools.com/angular/angular_services.asp

Список встроенных сервисов Angular <http://www.techstrikers.com/AngularJS/angularjs-built-in-services.php>

\$http сервис

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
    $scope.myWelcome = response.data;
  });
});
```

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>Today's welcome message is:</p>

<h1>{{myWelcome}}</h1>

</div>

<p>The $http service requests a page on the server, and the response is set as the value of the "myWelcome" variable.</p>

</body>
</html>
```

Еще один пример с \$http-сервисом

```
<div ng-app="myApp" ng-controller="customersCtrl">

<ul>
<li ng-repeat="x in names">
  {{ x.Name + ', ' + x.Country }}
</li>
</ul>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
  $http.get("http://www.w3schools.com/angular/customers.php")
    .success(function(response) {$scope.names = response.records;});
});
</script>
```

Создание своего собственного сервиса

```
app.service('hexafy', function() {
  this.myFunc = function (x) {
    return x.toString(16);
  }
});
```

Вызов своего собственного сервиса

```
app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
```

Практика:

1. Делаем трехстраничный сайт, используя \$http
2. Сделать подгрузку товаров при прокрутке в интернет магазине

Директивы

Директива расширяет HTML с помощью атрибутов и элементов

Создание собственной директивы

```
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
    return {
        template : "<h1>Made by a directive!</h1>"
    };
});
</script>

</body>
```

Данная директива выведет на месте < w3-test-directive> < /w3-test-directive> код < h1> Made by a directive!< /h1>

Подробнее о вариантах http://www.w3schools.com/angular/angular_directives.asp

закрытые элементы

```
<div ng-app="">

<p>
<button ng-disabled="mySwitch">Click Me!</button>
</p>

<p>
<input type="checkbox" ng-model="mySwitch">Button
</p>

</div>
```

Полезное чтение:

Директивы в Angularjs для начинающих. Часть 1

<https://habrahabr.ru/post/179755/>

Директивы в Angularjs для начинающих. Часть 2

<https://habrahabr.ru/post/180365/>

Разбор структуры директив по кусочкам

<https://habrahabr.ru/post/164493/>

Хранилище примеров от Андреева Артема

https://github.com/andreev-artem/angular_experiments

Практика:

1. Сделать DIV с параметром value, который превращается в progress bar

<http://tutorials.jenkov.com/angularjs/custom-directives.html>

<https://www.sitepoint.com/practical-guide-angularjs-directives/>

<http://www.jvandemo.com/the-nitty-gritty-of-compile-and-link-functions-inside-angularjs-directives/>

Работа со scope директивы

https://www.tutorialspoint.com/angularjs/angularjs_custom_directives.htm

Scope директив

scope:false - директива использует переменные внешнего контроллера. При изменении переменной внутри директивы, мы меняем переменную в контроллере

scope:true - директива получает переменные внешнего контроллера, но работает с ними "локально". То есть изменяя значение внутри директивы, мы не меняем значение внутри внешнего контроллера.

Изолированный scope scope:{}

При изолированном scope мы используем атрибут в директиве для связи переменных контроллера и директивы

@ односторонне связывание, только текст

В директиву передается значение из атрибута(см пример ниже атрибут connection) = *двустороннее связывание*

Изменения в контроллера передаются в директиву, и наоборот

```
var app = angular.module('scopeApp', []);

app.controller("AppCtrl", function ($scope) {
    $scope.ctrlName = "Vasya";
})

app.directive("changeName", function () {
    return {
        scope: {
            dirName: "=connection"
        },
        template: '<div>{{ dirName }}</div>',
    };
});
```

```
<div ng-app="scopeApp">
  <div ng-controller="AppCtrl">
    <input type="text" ng-model="ctrlName"> <!-- Ctrl -->
    <div drink connection="ctrlName"></div> <!-- Dir -->
  </div>
</div>
```

& использование метода контроллера

О различных scope'ах в Angular

Очень подробно о scope в Angular <http://jonathancreamer.com/working-with-all-the-different-kinds-of-scopes-in-angular/>

Fiddle с различными скоупами <https://jsfiddle.net/biondifabio/g3brja67/>

<http://monsterlessons.com/project/lessons/scope-true-v-diriektivakh-v-angularjs>

[http://artemdemo.me/blog/scope-%D0%B2-](http://artemdemo.me/blog/scope-%D0%B2-%D0%B4%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%85-%D0%B0%D0%BD%D0%B3%D1%83%D0%BB%D0%B0%D1%80%D0%B0-angularjs-directives/)

[%D0%B4%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%85-%D0%B0%D0%BD%D0%B3%D1%83%D0%BB%D0%B0%D1%80%D0%B0-angularjs-directives/](http://artemdemo.me/blog/scope-%D0%B2-%D0%B4%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%85-%D0%B0%D0%BD%D0%B3%D1%83%D0%BB%D0%B0%D1%80%D0%B0-angularjs-directives/)

<https://www.undefinednull.com/2014/02/11/mastering-the-scope-of-a-directive-in-angularjs/>

Об изолированном scope в Angular

<http://onehungrymind.com/angularjs-sticky-notes-pt-2-isolated-scope/>

\$routeProvider, \$locationProvider

Базовый роутинг https://www.w3schools.com/angular/angular_routing.asp

Построение одностраничного приложения на Angular <https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templating>

Убираем хештег из адресов <https://scotch.io/tutorials/pretty-urls-in-angularjs-removing-the-hashtag>

Динамический роутинг

<http://stackoverflow.com/questions/20637999/angularjs-use-routeprovider-when-variables-to-construct-templateurl-name>

```
.when('/pages/:pageName', {
  templateUrl: function(params) {
    return 'views/pages/' + params.pageName + '.html';
  },
  controller: 'MainController'
});
```

Примеры:

Список студентов <http://www.journaldev.com/6225/angularjs-routing-example-ngroute-routeprovider>

Фабрики

Обмен данными между двумя контроллерами через фабрики

<http://monsterlessons.com/project/lessons/obmien-dannymi-miezhdu-kontrollierami-v-angularjs>

Сервисы против Фабрик

<https://blog.thoughttram.io/angular/2015/07/07/service-vs-factory-once-and-for-all.html>

Фильтры

Фильтры позволяют модифицировать выдачу выражений

```
<div ng-app="myApp" ng-controller="personCtrl">

  <p>The name is {{ lastName | uppercase }}</p>

</div>
```

```
<div ng-app="myApp" ng-controller="namesCtrl">

  <ul>
    <li ng-repeat="x in names | orderBy:'country'">
      {{ x.name + ', ' + x.country }}
    </li>
  </ul>

</div>
```

Unit-тесты в Angular

Быстрый старт

Скачиваем standalone версию Jasmine <https://github.com/jasmine/jasmine/releases>

Нам нужен файл SpecRunner.html, который лежит в корне.

Создаем свой hello.js

```
function helloWorld() {  
  return "Hello world!";  
}
```

Подключаем его к SpecRunner.html . В нем же дописываем код

```
describe("Hello world", function() {  
  it("says hello", function() {  
    expect(helloWorld()).toEqual("Hello world!");  
  });  
});
```

Запускаем и ищем строчку "Hello World says hello"

Давайте разберем, что происходит

В describe мы указываем описание самого теста. it - дает описание того, что должно происходить в тесте expect - запускает функцию, которую мы тестируем и метод для сравнения (в нашем примере toEqual)

Создание собственной функции сравнения

```
function gimmeANumber() {
    var x = 4;
    return x;
}

describe('Hello world', function () {

    beforeEach(function () {
        jasmine.addMatchers({
            toBeDivisibleByTwo: function () {
                return {
                    compare: function (actual, expected) {
                        return {
                            pass: (actual % 2) === 0
                        };
                    }
                };
            }
        });
    });

    it('is divisible by 2', function () {
        expect(gimmeANumber()).toBeDivisibleByTwo();
        expect(5).not.toBeDivisibleByTwo();
    });

});
```

Пример взят с <http://stackoverflow.com/questions/23986665/jasmine-2-0-test-with-a-custom-matcher-fails-undefined-is-not-a-function>

Дополнительные материалы:

Пошаговое руководство для быстрого старта

<https://evanhahn.com/how-do-i-jasmine/>

Фикс примера с добавлением собственного matcher'a

<http://stackoverflow.com/questions/23986665/jasmine-2-0-test-with-a-custom-matcher-fails-undefined-is-not-a-function>

Введение в Jasmine

<https://habrahabr.ru/post/167173/>

Jasmine для NodeJS

<https://github.com/jasmine/jasmine-npm> <https://jasmine.github.io/2.5/node.html>

<http://stepansuvorov.com/blog/2012/10/jasmine-%D0%B8-%D1%8E%D0%BD%D0%B8%D1%82-%D1%82%D0%B5%D1%81%D1%82%D1%8B/>

Unit-тесты с помощью Jasmine <https://docs.angularjs.org/guide/unit-testing>

Jasmine, Karma <https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1>

Практика:

1. Сделать функцию, которая находит максимум из трех чисел. Написать как минимум три теста для тестирования.

\$watch \$digest \$apply

Предподготовка <https://docs.angularjs.org/guide/scope>

Хорошее объяснение использования \$watch и \$apply

<http://stackoverflow.com/questions/15112584/how-do-i-use-scope-watch-and-scope-apply-in-angularjs>

<http://tutorials.jenkov.com/angularjs/watch-digest-apply.html>

<https://www.sitepoint.com/understanding-angulars-apply-digest/>

<https://www.sitepoint.com/mastering-watch-angularjs/>

Практика:

1. Есть три круга и один квадрат. При клике на круг, он меняет цвет. Когда три круга поменяли цвет, меняет цвет квадрат.

\$broadcast, \$emit, \$on

<http://stackoverflow.com/questions/37717493/usage-of-broadcast-emit-and-on-in-angularjs>

Пример к ответу <http://plnkr.co/edit/am6IDw?p=preview>

<http://stackoverflow.com/questions/19446755/on-and-broadcast-in-angular>

\$index, \$event, \$log

<https://thinkster.io/egghead/index-event-log>

Компоненты

Об отличиях между компонентами и директивами

<https://scotch.io/tutorials/how-to-use-angular-1-5s-component-method>

Сравнение на примерах директив и компонент

<https://toddmotto.com/exploring-the-angular-1-5-component-method/>

<https://www.sitepoint.com/building-angular-1-5-components/>

<http://stepansuvorov.com/blog/2016/02/angularjs-%D0%BE%D1%82-directive-%D0%BA-component/>

<https://habrahabr.ru/post/277087/>

<https://gist.github.com/toddmotto/5b4de6c777d3e446e6410fdadb824522>

require, \$onInit

<https://toddmotto.com/angular-1-5-lifecycle-hooks>

history, \$location, \$route

<http://stackoverflow.com/questions/14070285/how-to-implement-history-back-in-angular-js>

Конфигурация route-провайдера <https://thinkster.io/a-better-way-to-learn-angularjs/config-function>

Подробное пошаговое изложение материала по Webpack

<https://learn.javascript.ru/screencast/webpack>

Webpack Start

Создадим файл home.js

```
'use strict';

let welcome = require('./welcome');

welcome("home");
```

Создадим файл welcome.js

```
'use strict';

module.exports = function(message) {
  alert(`Welcome ${message}`);
}
```

Установим webpack

```
sudo npm i -g webpack
```

Создадим файл конфигурации webpack.config.js

```
'use strict';

module.exports = {
  entry: "./home.js",
  output: {
    filename: "build.js"
  }
}
```

Далее вбиваем в командной строке

```
webpack
```

В результате должны получить build.js, который мы можем добавить к нашему index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script type="text/javascript" src="build.js"></script>
</head>
<body>

</body>
</html>
```

Полученный код может испытывать проблемы в Firefox. Вместо alert'a будем получать ошибку `let is a reserved identifier on firefox`

Нужно залезть в `build.js` и поменять `let` на `var`

Чуть позже мы воспользуемся для преобразования кода `babel`'ем

Разделение исходников и результата

Перенесем наши исходники в папку `app` и добьемся, чтобы файлы сборки помещались в папку `dist`. `index.html` останется на своем старом месте

Нам придется изменить `webpack.config.js`

```
module.exports = {
  entry: './app/home.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  }
}
```

и изменить путь и название подключаемого файла в `index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script type="text/javascript" src="dist/bundle.js"></script>
</head>
<body>

</body>
</html>
```

HtmlWebpackPlugin

HtmlWebpackPlugin занимается тем, что генерирует для html-файл по шаблону при сборке

Для использования плагина нам необходимо добавить его в dev-зависимости

```
npm install --save-dev html-webpack-plugin
```

После чего наш webpack.config.js нужно изменить на следующий код

```
'use strict';

var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: './app/home.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  plugins: [new HtmlWebpackPlugin({
    template: 'app/index.html'
  })]
}
```

То есть мы добавляем сверху подключение модуля нашего плагина

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
```

и добавляем его в настройки

```
plugins: [new HtmlWebpackPlugin({
  template: 'app/index.html'
})]
```

В качестве шаблона используем наш index.html, который мы перенесли в папку app

CSS в Webpack'e

Нам понадобятся style-loader, css-loader

настройки подключения

```
module.exports = {  
  module: {  
    rules: [  
      {  
        test: /\.css$/,  
        use: [ 'style-loader', 'css-loader' ]  
      }  
    ]  
  }  
}
```

в файл, который является точкой входа добавляем import('./styles.css');

```
'use strict';  
  
require('./styles.css');  
let welcome = require('./welcome');  
  
welcome("home");
```

и в результате получаем вот так вот webpack.config.js

```
'use strict';

var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: './app/home.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {test: /\.css$/, use: [ 'style-loader', 'css-loader' ]}
    ]
  },
  plugins: [new HtmlWebpackPlugin({
    template: 'app/index.html'
  })]
}
```

Полезное чтение:

1. Разбираемся с CSS <https://monsterlessons.com/project/lessons/sborshchik-moduliei-webpack-razbiraemsia-s-css-chast-3>
2. Получаем внешний css-файл с помощью webpack
<https://webpack.js.org/plugins/extract-text-webpack-plugin/#usage-example-with-css>
3. Работа с CSS <https://webpack.github.io/docs/stylesheets.html>

Babel

```
npm install --save-dev babel-loader babel-core babel-preset-env
```

В наш webpack.config.js нужно будет добавить строчку с правилом на сборку babel-loader'a

```
module: {
  rules: [
    {
      test: /\.js$/,
      exclude: /(node_modules|bower_components)/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['env']
        }
      }
    }
  ]
}
```

Если всё настроено правильно, то после сборки наши файлы

home.js

```
'use strict';

let welcome = require('./welcome');

welcome("home");
```

и welcome.js

```
'use strict';

module.exports = function(message) {
  alert(`Welcome ${message}`);
}
```

должны выдать фразу Welcome home

Подробнее: <https://github.com/babel/babel-loader>

Webpack start

Запуск из командной строки

```
npm install webpack -g
```

Запустим

```
webpack src/index.js assets/bundle.js
```

Создаем build.js

```
var path = require('path');
var webpack = require('webpack');
var config = {
  context: path.join(__dirname, 'src'), // исходная директория
  entry: './index', // файл для сборки, если несколько - указываем hash (entry name
=> filename)
  output: {
    path: path.join(__dirname, 'assets') // выходная директория
  }
};
var compiler = webpack(config);
compiler.run(function (err, stats) {
  console.log(stats.toJson()); // по завершению, выводим всю статистику
});
```

Запуск сборки

```
node build
```

watch:true

Webpack dev server

```
npm install webpack-dev-server -g
```

Глобально вызываемый модуль

```
module.exports = {  
  output: {  
    library: 'myClassName',  
  }  
};
```

Загрузка нескольких файлов

```
module.exports = {  
  entry: ["/global.js", "/app.js"],  
  output: {  
    filename: "bundle.js"  
  }  
}
```

Loader'ы

Минификация

```
webpack -p
```

```
var webpack = require("webpack");  
  
module.exports = {  
  
  entry: "./entry.js",  
  devtool: "source-map",  
  output: {  
    path: "./dist",  
    filename: "bundle.min.js"  
  },  
  plugins: [  
    new webpack.optimize.UglifyJsPlugin({minimize: true})  
  ]  
};
```

Дополнительные материалы:

Стартовый гайд по Webpack'у

<https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460#.vlijhe21f>

Getting started with Webpack2

<https://blog.madewithenvy.com/getting-started-with-webpack-2-ed2b86c68783#.wizy7qh3p>

<https://survivejs.com/webpack/developing-with-webpack/getting-started/>

О книге Detailed introduction to webpack, [Joseph Zimmerman](#)

www.smashingmagazine.com/2017/02/a-detailed-introduction-to-webpack/

7 бед - один ответ

<https://habrahabr.ru/post/245991/>

Скринкаст по Webpack

<https://www.youtube.com/playlist?list=PLDyvV36pndZHfBThhg4Z0822EEG9VGenn>

Минификация

Для минификации нам понадобится плагин `webpack.optimize.UglifyJsPlugin`

Конфигурация для минификации

```
var webpack = require("webpack");
var path = require('path');

module.exports = {

  entry: "./home.js",
  devtool: "source-map",
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: "bundle.min.js"
  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin({minimize: true})
  ]
};
```

Вызов минификации из командной строки

```
webpack -p
```

React

Дополнительные материалы:

<https://reacttraining.com/online/react-fundamentals>

React Старт

Создаем папку проекта big_start и запускаем в ней

```
npm init
```

Это создает package.json нашего проекта

```
npm install --save react react-dom
```

Эта команда заставит npm загрузить модули react и react_dom, а также все нужные им модули, в папку node_modules

Так же в package.json появится раздел dependencies

```
dependencies: {  
  "react": "^15.4.2",  
  "react-dom": "^15.4.2"  
}
```

Создадим файл .gitignore , для того, чтобы не загружать лишние файлы на Github.
Добавим в игнор-список папки

```
node_modules  
dist
```

Создадим папку app, в ней файлы index.css и index.js

index.css будет кратким

```
body {  
  background:darkolivegreen;  
}
```

index.js

```
var React = require('react');
var ReactDOM = require('react-dom');

require('./index.css');

class App extends React.Component {

  render(){
    return (
      <div>
        Hello World!
      </div>
    )
  }
}
```

Поменяем этот код на следующий

```
var React = require('react');
var ReactDOM = require('react-dom');

require('./index.css');

class App extends React.Component {

  render(){
    return React.createElement (
      "div",
      null,
      "Hello World!"
    );
  }
}
```

Попробуем воспользоваться командой render у ReactDOM

```
var React = require('react');
var ReactDOM = require('react-dom');

require('./index.css');

class App extends React.Component {

  render(){
    return (
      <div>
        Hello World!
      </div>
    )
  }
}

ReactDOM.render(
  <App />,
  document.getElementById('app')
)
```

Добавим модули к нашему проекту в раздел `devDependencies`, то есть они будут нужны нам для сборки проекта, но не в самом готовом проекте

```
npm install --save-dev babel-core babel-loader babel-preset-env babel-preset-react css-loader style-loader html-webpack-plugin webpack webpack-dev-server
```

После этого настроим webpack для нашего проекта. Для этого создадим в корне нашего проекта файл `webpack.config.js` со следующим наполнением

```
var path = require('path');

module.exports = {
  entry: './app/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'index_bundle.js'
  }
}
```

Раздел `entry` говорит Webpack'у, какой файл является ключевым. В разделе `output` мы задаем директорию, куда будем загружать готовые исходники и имя файла, под которым будет сохраняться результат(наш `index_bundle.js`)

Добавим модули для обработки наших файлов. Так все файлы с расширением .js будут обрабатываться babel-loader'ом, а .css обрабатываются style-loader'ом и css-loader'ом.

```
var path = require('path');

module.exports = {
  entry: './app/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'index_bundle.js'
  },
  module: {
    rules: [
      {test: /\.js$/, use: 'babel-loader'},
      {test: /\.css$/, use: [ 'style-loader', 'css-loader' ]}
    ]
  }
}
```

Добавим в наш package.json поднастройки babel

```
"babel": {
  "presets": [
    "env",
    "react"
  ]
}
```

babel-preset-env мы установили ранее, он позволяет динамически добавлять модули необходимые babel для запуска.

babel-react нужен нам для преобразования JSX

В результате получим package.json следующего вида

```
{
  "name": "big-start",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "babel": {
    "presets": [
      "env",
      "react"
    ]
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^15.5.4",
    "react-dom": "^15.5.4"
  },
  "devDependencies": {
    "babel-core": "^6.24.1",
    "babel-loader": "^7.0.0",
    "babel-preset-env": "^1.4.0",
    "babel-preset-react": "^6.24.1",
    "css-loader": "^0.28.0",
    "html-webpack-plugin": "^2.28.0",
    "style-loader": "^0.16.1",
    "webpack": "^2.4.1",
    "webpack-dev-server": "^2.4.2"
  }
}
```

Добавим html-webpack-plugin

```
var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: './app/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'index_bundle.js'
  },
  module: {
    rules: [
      {test: /\.js$/, use: 'babel-loader'},
      {test: /\.css$/, use: [ 'style-loader', 'css-loader' ]}
    ]
  },
  plugins: [new HtmlWebpackPlugin({
    template: 'app/index.html'
  })]
}
```

И создадим в папке app index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Big Start</title>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

Добавим раздел scripts в наш package.json

```
"scripts": {
  "create": webpack
}
```

Далее запускаем npm run create

У нас должна создаться папка dist, и в ней два файла index.html и index_bundle.js

index.html имеет следующий вид:


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Github Battle</title>
</head>
<body>
  <div id="app"></div>
<script type="text/javascript" src="index_bundle.js"></script></body>
</html>
```

Мы видим, что внутри кода происходит подключение файла `index_bundle.js`, который образовался после сборки

Если мы запустим `index.html` из папки `dist`, то наконец-то увидим Hello World!

Давайте попробуем автоматизировать обновлени страницы при обновлении кода

```
npm install --save-dev webpack-dev-server
```

Заменяем раздел `scripts` в нашем `package.json`

```
"scripts": {
  "start": "webpack-dev-server --open"
}
```

Результирующий `package.json` будет следующего вида

```
{
  "name": "big-start",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "webpack-dev-server --open"
  },
  "babel": {
    "presets": [
      "env",
      "react"
    ]
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^15.5.4",
    "react-dom": "^15.5.4"
  },
  "devDependencies": {
    "babel-core": "^6.24.1",
    "babel-loader": "^7.0.0",
    "babel-preset-env": "^1.4.0",
    "babel-preset-react": "^6.24.1",
    "css-loader": "^0.28.0",
    "html-webpack-plugin": "^2.28.0",
    "style-loader": "^0.16.1",
    "webpack": "^2.4.1",
    "webpack-dev-server": "^2.4.2"
  }
}
```

После запуска команды

```
npm run start
```

У нас откроется вкладка с адресом localhost:8080 и при дальнейших изменениях кода, она будет автоматически обновляться

Поздравляю! Вы наконец-то запустили свое первое React-приложение)

Дополнительные материалы:

<https://maxfarseer.gitbooks.io/react-course-ru>

Компоненты и свойства

Внутри функции Fullname, мы сможем получать доступ к свойствам компонента Fullname через props

```
function Fullname(props) {  
  return <h1>Hello, {props.name} {props.surname}</h1>;  
}  
  
const element = <Fullname name="Ostap" surname="Bender"/>;  
  
ReactDOM.render(element,  
  document.getElementById('mount-point')  
);
```

Посмотрите этот пример на Codepen <https://codepen.io/dmitrytinitilov/pen/eRzdWX>

Подробнее: <https://facebook.github.io/react/docs/components-and-props.html>

Работа с DOM-элементом

<https://facebook.github.io/react/docs/dom-elements.html>

State

<https://facebook.github.io/react/docs/state-and-lifecycle.html>

Практика:

1. Есть блок, внутри него счетчик. Каждую секунду счетчик увеличивается.
2. Счетчик обратного отсчета. Когда счетчик доходит до нуля, меняем цвет блока.

События

<https://facebook.github.io/react/docs/handling-events.html>

Счетчик кликов <https://codepen.io/html5andblog/pen/ENPWme>

Ответы на вопрос, почему мы делаем `.bind(this)` в обработчиках

<https://medium.com/@rjun07a/binding-callbacks-in-react-components-9133c0b396c6>

<http://reactkungfu.com/2015/07/why-and-how-to-bind-methods-in-your-react-component-classes/>

Рендеринг на сервере

<https://camjackson.net/post/server-side-rendering-with-react>

Redux

<http://redux.js.org/docs/introduction/Motivation.html>

https://github.com/tayiorbeii/egghead.io_redux_course_notes

<https://habrahabr.ru/post/318148/>

Redux Старт

<https://github.com/StephenGrider/ReduxSimpleStarter>

TODO list на Redux <https://github.com/lucasbento/react-redux-todo>

<https://github.com/reactjs/redux/tree/master/examples/todomvc>

Примеры:

Счетчик на Redux <https://codepen.io/jeremiahbiard/pen/bVyWEP>

Книга рецептов <https://codepen.io/alanbuchanan/pen/dGRqyR>

Змейка <https://codepen.io/CrocoDillon/pen/pgARwb>

Полезное чтение:

<https://medium.com/@MKulinski/react-redux-basics-a36914c0035d>

<https://medium.com/@firasd/quick-start-tutorial-using-redux-in-react-apps-89b142d6c5c1>

<https://github.com/happypoulp/redux-tutorial>

Много ссылок на другие ресурсы

<http://www.youhavetolearncomputers.com/blog/2015/9/15/a-conceptual-overview-of-redux-or-how-i-fell-in-love-with-a-javascript-state-container>

Vue JS

<https://github.com/vuejs/vue/blob/dev/README.md>

Полезное чтение:

Почему мы выбрали VueJS <https://geektimes.ru/company/banderolka/blog/284094/>

Vue JS Быстрый старт

```
<!DOCTYPE html>
<html lang="en">
  <meta>
    <meta charset="UTF-8">
    <title>Hello World in Vue.js</title>
  </meta>

  <body>

    <div id="hello-world-app">
      <h1>{{ msg }}</h1>
    </div>

    <script src="https://unpkg.com/vue@2.3.4"></script>

    <script>
      new Vue({
        el: "#hello-world-app",
        data() {
          return {
            msg: "Hello World!"
          }
        }
      });
    </script>

  </body>
</html>
```

Базовые директивы

v-for

```
<!DOCTYPE html>
<html lang="en">
  <meta>
    <meta charset="UTF-8">
    <title>Hello World in Vue.js</title>
  </meta>

  <body>

    <div id="hello-world-app">
      <h1>{{ msg }}</h1>
    </div>

    <script src="https://unpkg.com/vue@2.3.4"></script>

    <script>
      new Vue({
        el: "#hello-world-app",
        data() {
          return {
            msg: "Hello World!"
          }
        }
      });
    </script>

  </body>
</html>
```

Обработка событий

```
<script src="https://unpkg.com/vue@2.3.4"></script>

<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>

<script type="text/javascript">
  var app5 = new Vue({
    el: '#app-5',
    data: {
      message: 'Hello Vue.js!'
    },
    methods: {
      reverseMessage: function () {
        this.message = this.message.split('').reverse().join('')
      }
    }
  })
</script>
```

Мы можем генерировать кастомные события с помощью \$emit

```
<script src="https://unpkg.com/vue"></script>

<div id="todo-list-example">
  <input
    v-model="newTodoText"
    v-on:keyup.enter="addNewTodo"
    placeholder="Add a todo"
  >
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo"
      v-bind:title="todo"
      v-on:remove="todos.splice(index, 1)"
    ></li>
  </ul>
</div>

<script type="text/javascript">
  Vue.component('todo-item', {
    template: `
      <li>
        {{ title }}
        <button v-on:click="$emit('remove')">X</button>
      </li>
    `,
    props: ['title']
  })
  new Vue({
    el: '#todo-list-example',
    data: {
      newTodoText: '',
      todos: [
        'Do the dishes',
        'Take out the trash',
        'Mow the lawn'
      ]
    },
    methods: {
      addNewTodo: function () {
        this.todos.push(this.newTodoText)
        this.newTodoText = ''
      }
    }
  })
</script>
```


Экземпляр Vue

```
var data = { a: 1 }
var vm = new Vue({
  el: '#example',
  data: data
})

vm.a === data.a

vm.a = 2
data.a // -> 2

vm.$data === data // -> true
vm.$el === document.getElementById('example') // -> true
// $watch is an instance method
vm.$watch('a', function (newVal, oldVal) {
  // этот callback сработает, когда `vm.a` изменяется
})
```

Шаблоны

Базовые шаблоны

Значение из data.msg подставится в такой шаблон

```
<span>Message: {{ msg }}</span>
```

v-html позволяет устанавливать innerHTML div'a

```
<div v-html="innerHTML of that div"></div>
```

Привязка аргументов

```
<a v-bind:href="url"></a>
```

Привязка событий

```
<a v-on:click="doSomething">
```

Запуск event.preventDefault() на событии

Фильтры

Мы можем применить к сообщению фильтры

```
{{ message | capitalize }}
```

```
new Vue({
  // ...
  filters: {
    capitalize: function (value) {
      if (!value) return ''
      value = value.toString()
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
  }
})
```

Можно добавить несколько фильтров

```
{{ message | filterA | filterB }}
```

Сокращенные записи

Для v-bind

```
<!-- full syntax -->  
<a v-bind:href="url"></a>  
<!-- shorthand -->  
<a :href="url"></a>
```

Для v-on

```
<!-- full syntax -->  
<a v-on:click="doSomething"></a>  
<!-- shorthand -->  
<a @click="doSomething"></a>
```

Полезное чтение:

1. 7 способов определить шаблон во VueJS

<https://medium.com/js-dojo/7-ways-to-define-a-component-template-in-vuejs-c04e0c72900d>

Компоненты

Напишем наш компонент

```
// register
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// create a root instance
new Vue({
  el: '#example'
})
```

Такой компонент будет менять вот такой html-код

```
<div id="example">
  <my-component></my-component>
</div>
```

В ВОТ ТАКОЙ

```
<div id="example">
  <div>A custom component!</div>
</div>
```

Дополнения

Коллекция концепций из книги «Чистый код» с наглядными примерами их применения на JavaScript

<https://github.com/ryanmcdermott/clean-code-javascript>

#Инструменты JavaScript разработчика

Более 800 ресурсов для Front-End разработчиков <https://habrahabr.ru/company/it-grad/blog/275673/>

<https://habrahabr.ru/company/nixsolutions/blog/277869/>

Плагины к Sublime <https://habrahabr.ru/post/235901/>

Tern для Sublime - автодополнение и упрощенная навигация по документу
https://github.com/ternjs/tern_for_sublime

Настройка автоподсказок для JavaScript
<https://stackoverflow.com/questions/13661462/getting-full-js-autocompletion-under-sublime-text>

JavaScript Completion <https://packagecontrol.io/packages/JavaScript%20Completions>

All AutoComplete <https://packagecontrol.io/packages/All%20Autocomplete>

ESLint

Базовая установка

<https://packagecontrol.io/packages/ESLint>

Предполагается, что у нас на компьютере уже установлен nodejs и npm

1. Устанавливаем eslint через npm в наш проект

```
npm install eslint
```

Или глобально, если хотим, чтобы оно сработало для всех проектов

```
npm install -g eslint
```

1. Настраиваем или скачиваем .eslintrc

```
{
  "globals": {
    // Put things like jQuery, etc
    "jQuery": true,
    "$": true
  },
  "env": {
    // I write for browser
    "browser": true,
    // in CommonJS
    "node": true
  },
  // To give you an idea how to override rule options:
  "rules": {
    // Tons of rules you can use, for example...
    "quotes": [1, "double"]
  }
}
```

1. В Sublime устанавливаем через Package Control, пакет ESLint

Далее по клику правой клавишей в открывшемся меню выбираем ESLint, либо нажимаем Ctrl+Alt+e

F4 - перейти на следующую ошибку Shift + F4 - вернуться на предыдущую ошибку

Настраиваем правила

0 - Отключить правило 1 - Warn - правило предупреждение 2 - Error - правило бросает ошибку

Полный список правил можно найти вот здесь <http://eslint.org/docs/rules/>

Настраиваем ESLint под настройки Airbnb

```
npm install --save-dev eslint-config-airbnb
```

в .eslintrc добавляем

```
"extends": "airbnb"
```

Дополнительное чтение:

1. Установка ESLint на Sublime Text <http://jonathancreamer.com/setup-eslint-with-es6-in-sublime-text/>
2. Настраиваем ESLint под настройки Airbnb <https://medium.freecodecamp.org/adding-some-air-to-the-airbnb-style-guide-3df40e31c57a>
3. Настройка Linter'ов на Sublime <https://habrahabr.ru/post/278747/>

Вдохновение

Подборка сайтов для вдохновения <https://vc.ru/p/inspiration-design>

Сайт студии <https://www.madeinhaus.com/>

Необычная навигация <http://www.welovenoise.com/>

Эффекты при скроллинге <https://trends2015.hautehorlogerie.org>

Примеры для медитации

<http://codepen.io/tmrDevelops/pen/ygBgVL>

<http://codepen.io/thykka/pen/evzWZW>

Эффект кипящей воды <http://jsfiddle.net/blakedietz/R5cRK/1/light/>

Часы в виде кривых <http://codepen.io/gbnikolov/pen/OWYGma>

Подборка интересных Codepen'ов <https://freebiesupply.com/blog/19-mind-blowing-codepen-experiments/>

Ресурсы

<http://frontender.info/>

<http://heyjavascript.com/>

Эксперименты

Устраиваем гонки на пяти девайсах <https://activetheory.net/work/racer>

Codepens

Графический редактор с музыкой <http://codepen.io/mattjuggins/pen/JRjEwB>

Шарики и гравитация <http://codepen.io/plasm/pen/vxMYEz>

Пример с параллаксом <http://codepen.io/nikitagupp/pen/ZewmPj>

Вращающийся куб <http://codepen.io/FlawlessDog/pen/ryQMvM>

Кот, который двигает глазами <http://codepen.io/tsukulognet/pen/VpYOyp>

Микровзаимодействия

Изменение градиента, при движении мыши <http://codepen.io/codedsignal/pen/vxPaVo>

Интерфейсы

Нотификации http://codepen.io/sean_codes/pen/XMwYqr

Slide-button из iPhone <http://codepen.io/kkanyingi/pen/mWYMzV>

Эффекты

Нерезкие буквы <http://codepen.io/jlnljn/pen/XMQrXW>

Насыпающиеся частицы <http://codepen.io/plasm/pen/WpBpRR>

Примеры для медитации

Mandala Outline <http://codepen.io/TWAIN/pen/vxzzvM>

Rasengan <http://codepen.io/gotoandplaynowtoo/pen/BLAyjQ>

Chidori(не спрашивайте, что это?) <http://codepen.io/gotoandplaynowtoo/pen/RGrVok>

Рост <http://codepen.io/hoqqanen/pen/WpWgYd>

Новый источник энергии <http://codepen.io/Mertl/pen/YZbZMy>

Материалы для чтения

1. Написание собственного js-фреймворка <https://blog.risingstack.com/writing-a-javascript-framework-project-structuring/>
2. От нуля до героя фронтенда <https://medium.com/russian/%D0%BE%D1%82-%D0%BD%D1%83%D0%BB%D1%8F-%D0%B4%D0%BE-%D0%B3%D0%B5%D1%80%D0%BE%D1%8F-%D1%84%D1%80%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D0%B4%D0%B0-%D1%87%D0%B0%D1%81%D1%82%D1%8C-2-25f19e56eb29>

Книги

1. JavaScript: The Definitive Guide by David Flanagan
2. Eloquent JavaScript by Marijn Haverbeke Выразительный JavaScript
<http://habrahabr.ru/post/240225/>
3. Дуглас Крокфорд JavaScript, Сильные стороны JavaScript: The Good Parts by Douglas Crockford
4. Writing Maintainable JavaScript, Nicholas Zakas
5. JavaScript Patterns by Stoyan Stefanov
6. Эдди Османи, Паттерны для масштабируемых JavaScript-приложений
<http://largescalejs.ru/>
7. Функциональный JavaScript <http://habrahabr.ru/post/229893/> Leland Richardson, Functional Javascript
8. Michael Fogus - Functional JavaScript - 2013
9. JavaScript Allongé, Reginald Braithwaite - рассмотрение тех сторон JavaScript, благодаря которым он отличается от других языков
10. Human JavaScript - книга о веб-разработке в команде
<http://read.humanjavascript.com/>
11. Сила JavaScript. 68 способов эффективного использования JS. Дэвид Херман. Идеальная книга перед собеседованиями по JavaScript

Подборка книг по JavaScript http://vk.com/page-54530371_48792013

Подборка от habrahabr

<http://habrahabr.ru/post/149082/>

Frontend

Фундаментальная книга по фронтенду <http://www.frontendhandbook.com/>

Основы для фронтенда <http://frontender.info/a-baseline-for-front-end-developers/>

Построение SPA <http://singlepageappbook.com>

Angular

Recipes with angular js <https://leanpub.com/recipes-with-angular-js/read>

О фреймворках <http://habrahabr.ru/company/piter/blog/257923/>

Неплохой tutorial <https://thinkster.io/a-better-way-to-learn-angularjs/>

Сервисы AngularJS: Factory, Service, Provider - создание и расширение
<http://onedev.net/post/334>

Сервисы <http://metanit.com/web/angular/3.1.php>

Понимание типов сервисов <http://habrahabr.ru/post/190342/>

Ошибки в AngularJS <http://stepansuvorov.com/blog/2014/12/angularjs-mistakes/>

Как правильно готовить Angular

<http://stepansuvorov.com/blog/2014/07/%D0%BA%D0%B0%D0%BA-%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE-%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%B8%D1%82%D1%8C-angular/>

AngularJS для привыкших к jQuery <http://habrahabr.ru/post/172975/>

ЗАЧЕМ НУЖЕН ANGULAR.JS И ПОЧЕМУ ИМЕННО ОН <https://mkdev.me/posts/zachem-nuzhen-angular-js-i-pochemu-imenno-on>

Магия AngularJS: никогда не вешайте binding на примитивы
<http://habrahabr.ru/post/223529/>

AngularJS: нестандартное поведение ng-if <http://habrahabr.ru/post/225243/>

Рекурсивные шаблоны в AngularJS <http://devacademy.ru/posts/rekursivnyie-shablonyi-v-angularjs/>

Вопросы к собеседованию

<https://habrahabr.ru/post/239065/>

<https://www.sitepoint.com/5-typical-javascript-interview-exercises/>

<http://www.quizful.net/interview/js>

Разноплановые вопросы

<https://habrahabr.ru/post/231071/>

Вопросы к собеседованию по Frontend'у

<https://github.com/h5bp/Front-end-Developer-Interview-Questions/tree/master/Translations/Russian>

Знаете ли Вы все по фронтенду?

<https://know-it-all.io/>

Работа с Git

```
git init
```

git init - инициализирует репозиторий

```
git clone https://github.com/shorten/js.git
```

```
git status
```

покажет нам состояние системы

```
git add <filename>
```

добавит один файл

```
git add *
```

Добавим адрес удаленного хранилища

```
git remote add origin <server>
```

Отправим данные на удаленное хранилище из ветки master

```
git push origin master
```

добавит все файлы из директории

Краткий стартовый мануал по Git <http://rogerdudler.github.io/git-guide/>

Cheatsheet по различным ситуациям <https://gist.github.com/jedmao/5053440>

Работа с ветками

О feature branch <https://bocoup.com/blog/git-workflow-walkthrough-feature-branches>

Workflow <https://confluence.atlassian.com/bitbucket/workflow-for-git-feature-branching-814201830.html>

<https://www.atlassian.com/git/tutorials/comparing-workflows>

Очень емкий мануал по всяким проблемным ситуациям с git <http://ohshitgit.com>

Заливка проекта на github

<http://gotit.com.ua/administration-systems-i-services/kak-zalit-proekt-na-github-com.html>

Создание ssh-ключа

<https://docs.joyent.com/public-cloud/getting-started/ssh-keys/generating-an-ssh-key-manually/manually-generating-your-ssh-key-in-windows>

Создание ветки на git

<https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches>

Советы по Git'у от Mail.ru

<https://habrahabr.ru/company/mailru/blog/267595/>

Полезное чтение:

1. Ежедневная работа с Git <https://habrahabr.ru/post/174467/>
2. Интерактивный тур по Git'у <https://githowto.com/ru>

Ветки на Git

Клонирование с репозитория нужно ветки `git clone http://whatever.git -b branch-name`

1. Работа в команде на Github <https://code.tutsplus.com/articles/team-collaboration-with-github--net-29876>
2. Удачная модель ветвления для Git <https://habrahabr.ru/post/106912/>
3. Правила хорошего тона при работе с Git в многопользовательском окружении <https://evasive.ru/9c07c3cd906aa3a02cba77b18c1a573b.html>

Интеграция Slack с GitHub

Полезное чтение:

<https://get.slack.help/hc/en-us/articles/232289568-Use-GitHub-with-Slack>

Задания для практики

Здесь Вы можете потренировать свои умения по JavaScript

<https://vc.ru/p/moberg-dots>