

Игорь Борисов

JavaScript. Уровень 3. Knockout.js

<http://igor-borisov.ru>

Темы курса

- Введение в Knockout.js
- Наблюдения
- Привязки
- Использование шаблонов
- Использование компонентов

Модуль 1

Knockout.js

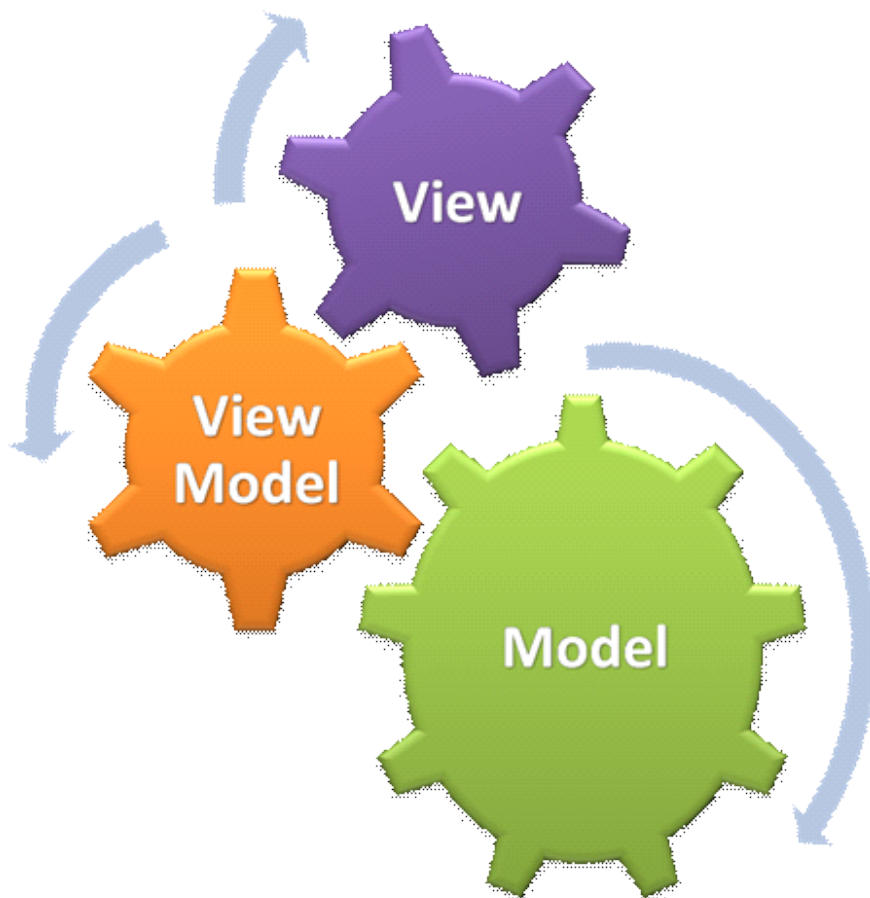
Наблюдения

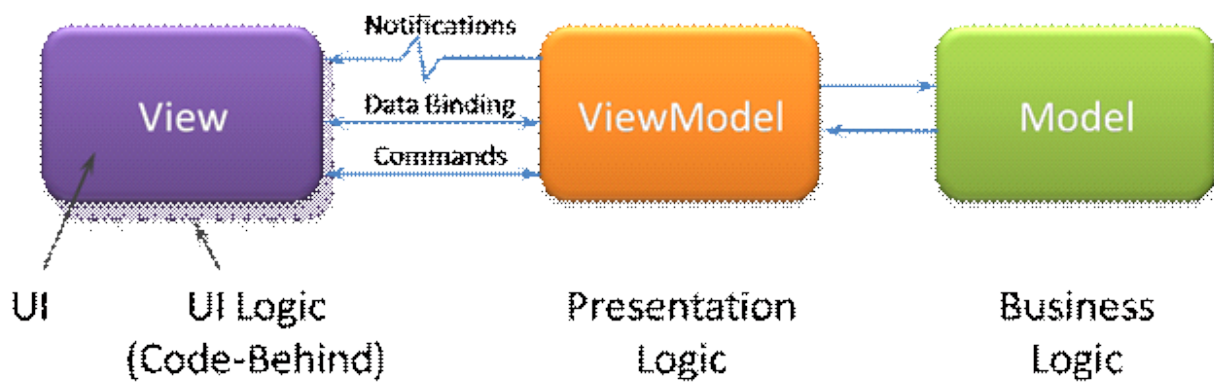
Темы модуля

- Общие сведения о Knockout.js
- Шаблон Model-ViewModel-View
- Установка наблюдения
- Вычисляемые наблюдения
- Наблюдения за массивами

Что такое Knockout.js

- Свободный JavaScript каркас (framework) веб-приложений, реализующий Model-View-ViewModel шаблон.
- Основными принципами являются:
 - чёткое разделение доменных данных, компонентов отображения и отображаемых данных
 - наличие чётко определённого слоя специализированного кода, задающего отношение компонентов отображения
 - простота использования
- Используется для проектирования масштабируемых, управляемых данными пользовательских интерфейсов





Где живёт Knockout.js?

<http://knockoutjs.com>


[Home](#) [Download / Install](#) [Tutorials](#) [Live examples](#) [Documentation](#) [Forum](#) [Source](#)

Knockout.

Simplify dynamic JavaScript UIs with the Model-View-View Model (MVVM) pattern


[Download](#)
v3.3.0 - 21kb min+gz
[release notes](#)

Key concepts




Declarative Bindings

Easily associate DOM elements with model data using a concise, readable syntax




Automatic UI Refresh

When your data model's state changes, your UI updates automatically



Dependency Tracking

Implicitly set up chains of relationships between model data, to transform and combine it



Templating

Quickly generate sophisticated, nested UIs as a function of your model data

More features

- ✓ Free, open source ([MIT license](#))
- ✓ Pure JavaScript — works with any web framework
- ✓ Small & lightweight — 54kb minified
... reduces to 20kb when using HTTP compression

New: Interactive tutorials

Get started with knockout.js quickly, learning to build *single-page applications*, *custom bindings* and more with [these interactive tutorials](#).

Модуль 1. Наблюдения Стр.7

Объявление ViewModel

```
<!-- Объявляем ViewModel -->
<script>
    var customer = {
        name: "Гость",
        location: "Москва"
    };
    // Активируем Knockout.js
    ko.applyBindings(customer);
</script>

<!-- Привязываем ViewModel к View -->
<h1>Привет, <span data-bind="text: name">
</span>
    из <span data-bind="text: location">
</span>!</h1>
```


Установка наблюдения

<!-- Устанавливаем наблюдение -->

<script>

```
var customer = {  
    name: ko.observable("Гость"),  
    location: ko.observable("Москва")  
};  
customer.name("Mike");
```

</script>

<!-- Создание объекта с помощью
конструктора -->

<script>

```
function Customer(){  
    this.name = ko.observable("Гость");  
    this.location = ko.observable("Москва");  
}  
ko.applyBindings(new Customer());
```

</script>

Вычисляемое наблюдение

```
<script>
  this.getLocation = function(){
    var location = prompt("Откуда вы?", this.location());
    this.location(location);
  };
  this.info = ko.computed(function() {
    return this.name() + " из " + this.location();
  }, this); // выделить болдом
</script>

<span data-bind='text: info'></span>
```

Лабораторная работа 1.1

Базовое использование наблюдений

Содержание лабораторной работы 1.1

Базовое использование наблюдений

Упражнение 1: Создание и активация ViewModel

- Откройте в текстовом редакторе файл **user.html**
- Создайте ViewModel с помощью функции-конструктора под именем **PersonViewModel**
- Опишите свойства: **firstName** и **lastName** с произвольными значениями
- Активируйте Knockout.js используя метод **applyBinding()**
- Свяжите элементы **span** со соответствующими свойствами модели с помощью атрибута **data-bind**
- Запустите код и убедитесь, что текущие данные модели отображаются

Упражнение 2: Использование вычисляемого наблюдения

- Добавьте в **div** с **id="person"** HTML-строку для вывода полного имени:
`<p>Полное имя: ...</p>`
- В модели **PersonViewModel** добавьте свойство **fullName**, которое будет осуществлять вычисляемое наблюдение за свойствами **firstName** и **lastName**
- Свяжите созданный элемент **strong** со свойством модели **fullName** с помощью атрибута **data-bind**
- Не забудьте сделать свойства **firstName** и **lastName** наблюдаемыми
- Запустите код и убедитесь, что текущие данные модели отображаются

Упражнение 3: Изменение значений свойств модели

- Добавьте в HTML-код кнопку:
`<button>Как вас зовут?</button>`
- В модели **PersonViewModel** добавьте метод **getName**, который запрашивает имя пользователя (с помощью метода `prompt` объекта `window`) и изменяет значения свойств модели **firstName** и **lastName**
- Свяжите созданную кнопку с методом модели **getName** с помощью атрибута **data-bind**
- Запустите код и убедитесь в его работоспособности

Наблюдение за массивами

```
<table>
  <thead>
    <tr>
      <th>Название</th>
      <th>Цена</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>title</td>
      <td>price</td>
    </tr>
  </tbody>
</table>
```

```
<script>
  function Course(title, price) {
    this.title = ko.observable(title);
    this.price = ko.observable(price);
  }

  // Добавляем список в объект Customer
  this.coursesList = ko.observableArray([
    new Course("HTML/CSS", 9150),
    new Course("WebMastering", 13450),
    new Course("JavaScript-ECMA", 8950)
  ]);
</script>
```

```
<tbody data-bind='foreach: coursesList'>
  ...
```

```
<td data-bind='text: title'></td>
```

...

Методы наблюдаемых массивов

```
deleteItem = function(item) {  
  observableArray.remove(item);  
  observableArray.destroy(item);  
};
```

- push(), pop()
- unshift(), shift()
- remove(), removeAll()
- destroy(), destroyAll()
- sort(), reversed()
- slice(), indexOf()

Лабораторная работа 1.2

Использование наблюдаемых массивов

Содержание лабораторной работы 1.2

Использование наблюдаемых массивов

Упражнение 1: Заполнение таблицы в цикле

- Откройте в текстовом редакторе файл **meal.html**
- Ознакомьтесь с уже имеющимся кодом
- Заполните таблицу, используя цикл `foreach` для элемента `tbody` по свойству `seats` модели **ReservationsViewModel**:
`<tbody data-bind="foreach: seats">`
- В цикле свяжите элементы **td** с помощью атрибута **data-bind** со свойствами модели **ReservationsViewModel**:
`<td data-bind="text: name"></td>`
...
- Обратите внимание на то, что свойство **self.meal** является массивом объектов, а доступ к свойству объекта можно получить как: **array().property**
- Запустите код и убедитесь в его работоспособности

Упражнение 2: Добавление записи в таблицу

- Добавьте в HTML-код элемент **button** с надписью **"Заказать место"**
- Свяжите созданную кнопку с методом модели **addSeat** с помощью атрибута **data-bind**:
`<button data-bind="click: addSeat">Заказать место</button>`
- В модели **ReservationsViewModel** добавьте метод **addSeat**, добавляет новый элемент в наблюдаемый массив **seats**. В качестве имени по-умолчанию укажите "Гость", в качестве блюда - первое блюдо:
`self.seats.push(new SeatReservation("Гость", self.availableMeals[0]));`
- Запустите код. Убедитесь, что в таблицу добавляется новая запись

Упражнение 3: Добавление записи в таблицу со значениями

- Добавьте в метод **addSeat** функционал, который запрашивает имя пользователя и номер блюда (с помощью метода `prompt` объекта `window`) в формате **"имя|номер"**, например "Гость|0"
- Измените код добавления нового элемента в массив **seats** используя полученные значения
- Запустите код. Убедитесь, что в таблицу добавляется новая запись со значениями переданными пользователем

Упражнение 4: Форматирование вывода цены

- Добавьте в конструктор **SeatReservation** метод **formattedPrice** как:

```
self.formattedPrice = ko.computed(function() {});
```

- Наблюдаемая функция должна вернуть строку **"цена руб"** (например "200 руб."), если таковая имеется, либо **"нет"**, в случае её отсутствия
- Свяжите элемент **th**, в котором указывается цена с методом **formattedPrice**
- Запустите код и убедитесь в его работоспособности

Упражнение 5: Удаление записи из таблицы

- В модели **ReservationsViewModel** добавьте метод **removeSeat**, который удаляет элемент из массива **seats**
- В таблице добавьте колонку для удаления текущего элемента:
`<td>Удалить</td>`
- Запустите код и убедитесь, что записи из таблицы удаляются

Упражнение 6: Вывод итоговой суммы заказов

- В модели **ReservationsViewModel** добавьте метод **totalPrice**, который возвращает сумму заказанных блюд
- В HTML-коде под таблицей добавьте подзаголовок для отображения суммы заказов:
`<h3>Всего заказов на:
 руб.</h3>`
- Запустите код и убедитесь в его работоспособности

Что мы изучили?

- Уяснили, что такое Knockout.js
- Получили представление о шаблоне MVVM
- Разобрались с понятием "наблюдение"

Модуль 2

Knockout.js

Привязки

Темы модуля

- Управляющие привязки
- Привязки, связанные с отображением данных
- Интерактивные привязки

Управляющие привязки

■ foreach

○ \$root

```
<ul data-bind='foreach: collection'>
  <li data-bind='text: $root.property'>
</li>
</ul>
```

○ \$data

```
<ul data-bind='foreach: collection'>
  <li data-bind='text: $data'></li>
</ul>
```

○ \$index (наблюдаемый)

```
<ul data-bind='foreach: collection'>
  <li data-bind='text: $index'></li>
</ul>
```

○ \$parent

■ if

```
<td data-bind='if: returnNumber() > 0'>
  <span>Отображаем при верном
условии</span>
</td>
```

■ ifnot

■ with

```
<p data-bind='with: course'>
  <span data-bind='text: title'></span>
<br />
```

```
<span data-bind='text: price'></span>  
рублей.  
</p>
```

Типы "видовых" привязок

- text: <value>
`<strong data-bind='text: methodName'>`
- html: <value>
`<strong data-bind='html: methodName'>`
- visible: <condition>
`<td data-bind='visible: returnNumber() > 0'>`
- css: <object>
` 0}">Paint it!`
- style: <object>
` 0 ? 'red' : 'black'}">Paint it!`
- attr: <object>
`<a data-bind='attr: {href: link}'>Some link`

Интерактивные привязки

- click: <method>
`<button data-bind='click: someMethod'>Нажать</button>`
- value: <property>
`<input type="text" data-bind='value: someValue' />`
- event: <object>
`<div data-bind='event:{mouseover: methodOne, mouseout: methodTwo}'>...</div>`
- submit: <method>
`<form data-bind="submit: doSomething">`
- enable: <property>
`<input type="text" data-bind='value: val1' />
<input type="text" data-bind='value: val2, enable: val1' />`
- disable: <property>
- checked: <property>
`<input type="checkbox" data-bind='checked: val' />`
- options: <array>
`<select data-bind='options: value, value: currentValue'>
</select>
<select data-bind='options:list, optionsText:"Text",
optionsValue:"Value"'></select>`
- selectedOptions: <array>
`<select data-bind="options: coursesList, selectedOptions:
currentCourse"></select>`
- hasfocus: <property>
`<input data-bind='value: val, hasfocus: true' />`

Лабораторная работа 2.1

Базовое использование интерактивных привязок

Содержание лабораторной работы 2.1

Базовое использование интерактивных привязок

Упражнение 1: Добавление полей для изменения значений

- Откройте в текстовом редакторе файл **user.html**
- Добавьте в HTML-код перед выводом полного имени пользователя два текстовых поля (используйте элемент веб-формы **input**).
- Сделайте текстовые поля редактируемыми, используя интерактивную привязку **value**. Первое поле связывает своё значение со значением **firstName**, а второе - со значением **lastName** модели **PersonViewModel**
- Запустите код и убедитесь, что при изменении значений в текстовых полях уже имеющиеся значения (в элементах **span** и **strong**) меняются.

Упражнение 2: Изменение видимости HTML-элемента

- В модели **PersonViewModel** добавьте метод **isFullName**, который возвращает булев тип **true** в случае, если оба свойства (**firstName** и **lastName**) имеют значения и **false**, если хотя бы одно из них значения не имеет
- Сделайте так, чтобы HTML-элемент **p**, который отвечает за показ полного имени, отображался только тогда, когда заполнены оба текстовых поля. Используйте для этого привязку **visible** и созданный метод **isFullName**
- Запустите код и убедитесь в работоспособности кода

Лабораторная работа 2.2

Использование привязок

Содержание лабораторной работы 2.2

Использование привязок

Упражнение 1: Добавление полей для ввода информации

- Откройте в текстовом редакторе файл **meal.html**
- Измените содержимое первой колонки таблицы на текстовое поле
- Свяжите это поле с помощью атрибута **data-bind** используя привязку **value** со свойством модели **ReservationsViewModel name**
- Удалите из метода **addSeat** модели **ReservationsViewModel** вызов метода **prompt** и назначьте новой записи произвольные значения по-умолчанию
- Запустите код и убедитесь, что теперь в первой колонке таблицы отображается редактируемое текстовое поле

Упражнение 2: Использование списка для выбора

- Измените содержимое второй колонки таблицы на HTML-элемент **select**
- Свяжите созданный элемент с помощью атрибута **data-bind** используя привязку **options** с массивом **availableMeals**. В списке должны показываться названия блюд
- Запустите код и убедитесь, что теперь во второй колонке таблицы отображается список блюд. При выборе того или иного пункта в списке значения в третьей колонке таблицы должны меняться

Упражнение 3: Экспорт данных в JSON

- Добавьте под таблицей HTML-элемент **textarea**
- Добавьте в HTML-код кнопку с надписью **"Экспорт"**
- Свяжите событие нажатия на созданную кнопку с методом **dataExport** модели **ReservationsViewModel**
- Опишите метод **dataExport**. Он должен преобразовывать текущие данные в строку в формате JSON и выводить их в HTML-элемент **textarea**
- Сделайте кнопку не работающей при отсутствии записей в таблице
- Запустите код и убедитесь в работоспособности кода

Что мы изучили?

- Разобрались с управляющими привязками
- Получили представление о разных типах привязок, связанных с отображением данных
- Изучили интерактивные привязки

Модуль 3

Knockout.js

Шаблоны и компоненты

Темы модуля

- Использование шаблонов в Knockout.js
- Использование компонентов

Использование шаблонов - 1

```
<script>
  function Courses() {
    this.kojs = { title: 'Knockout.js', desc: "Описание..." };
    // ...
  }
  ko.applyBindings(new Courses());
</script>

<script type="text/template" id="tpl">
  <h3 data-bind="text: title"></h3>
  <p data-bind="text: desc"></p>
</script>

<div data-bind="template: { name: 'tpl', data: kojs }"></div>
```

Использование шаблонов - 2

```
<script>
  function Courses() {
    this.courses = [
      { title: 'Knockout.js', desc: "Описание..." },
      //...
    ];
  }
  ko.applyBindings(new Courses());
</script>

<div data-bind="template: { name: 'tpl1', foreach: courses }"></div>
```

Использование шаблонов - 3

```
<script>
var Courses = {
  tracks: ko.observableArray([
    { name: 'JavaScript', length: 72, titles: [ 'ECMA-262',
    'BOM/DOM', 'Node.js' ] },
    //...
  ])
}
ko.applyBindings(Courses);
</script>

<!-- Шаблоны для отображения -->
<script type="text/html" id="tracks">
  <li>
    <strong data-bind="text: name"></strong>
    <ul data-bind="template: { name: 'titles', foreach:
titles, as: 'title' }"></ul>
  </li>
</script>

<script type="text/html" id="titles">
  <li data-bind="text: title"></li>
</script>

<!-- И само представление -->
<ul data-bind="template: { name: 'tracks', foreach: tracks,
as: 'track' }"></ul>
```

Лабораторная работа 3.1

Базовое использование шаблонов

Содержание лабораторной работы 3.1

Базовое использование шаблонов

Упражнение 1: Использование шаблона для вывода данных

- Откройте в текстовом редакторе файл **contacts.html**
- Создайте шаблон с помощью HTML-элемента **script** с **id="contacts"**
- Перенесите содержимое HTML-элемента **div** (не включая сам **div**) в созданный шаблон
- Измените привязку **div** так, чтобы в цикле использовался созданный шаблон:

```
template: { name: 'contacts', foreach: contacts }
```
- Запустите код и убедитесь в его работоспособности

Упражнение 2: Добавление возможности удаления данных

- В модели **ContactsViewModel** добавьте метод **removeContact**, который удаляет выбранный контакт:

```
this.removeContact = function () {  
    self.contacts.remove(this);  
}
```
- Добавьте в шаблон HTML-элемент **button**, который отвечает за удаление контакта:

```
<button>Удалить</button>
```
- Привяжите событие добавленной кнопки **click** к методу **removeContact**
- Запустите код и убедитесь в его работоспособности

Упражнение 3: Использование дополнительных событий

- В модели **ContactsViewModel** добавьте метод **afterAdd**, который изменяет цвет текста данных после их добавления:

```
this.afterAdd = function (elements, data) {  
    for (var i = 0; i < elements.length; ++i) {  
        if (elements[i].nodeType == 1 &&  
            elements[i].tagName != "BUTTON")  
            elements[i].style.color = "red";  
    }  
}
```
- Добавьте в HTML-элемент **div** привязку для вновь созданного метода:

```
{... afterRender: afterAdd}
```
- Запустите код и убедитесь в его работоспособности

Создание компонентов

```
<!-- Регистрация компонента: схема -->
<script>
    ko.components.register('component-name', {
        viewModel: {...},
        template: {...}
    });
</script>
```

```
<!-- Шаблон отображения компонента -->
<script type="text/mytmp" id="user-list-
template">
    <h3>Список пользователей</h3>
    <hr>
</script>
```

```
<!-- Регистрация компонента: вариант 1 -->
<script>
    function userViewModel(params){
        console.log("here");
    }
    ko.components.register("user-list", {
        viewModel: userViewModel,
        template: {element: "user-list-
template"}
    });
    ko.applyBindings();
</script>
```

```
<!-- Регистрация компонента: вариант 2 -->
<script>
    ko.components.register("user-list", {
        viewModel: {
            createViewModel: function(params,
componentInfo){
                console.log("here");
                return new userViewModel(params);
            }
        },
        template: {element: "user-list-
template"}
    });
    ko.applyBindings();
</script>
```

```
<!-- Варианты указания шаблона -->
<script>
    // шаблон по ID
    template: {element: "user-list-template"}

    // шаблон по выбранному объекту
    var cmp = document.getElementById('user-
list-template');
    template: { element: cmp }

    // шаблон как строка
    template: "<h3>Список пользователей</h3>
<hr>"
```

```
// шаблоны как список узлов
var list = [
    document.getElementById("x"),
    document.getElementById("y"),
]
template: list
</script>

<!-- Привязка компонентов -->
<div data-bind="component: 'text-field'">
</div>

<div data-bind="component: {name:'text-
field', params:{initialText:'!!!'}}"></div>

<text-field params="initialText:'Custom'">
</text-field>
```


Лабораторная работа 3.2

Базовое использование компонентов

Содержание лабораторной работы 3.2

Базовое использование компонентов

Упражнение 1: Создание шаблона компонента

- Откройте в текстовом редакторе файл **counter.html**
- В HTML-элементе **head** создайте шаблон компонента используя HTML-элемент **script** с произвольным значением атрибута **type** и со значением **counter** для атрибута **id**
- Внутри созданного шаблона создайте HTML-элемент **div** и задайте ему атрибут **data-bind** со значением **"text: inc"**
- Добавьте в шаблон два HTML-элемента **button** с текстом на кнопке **"Увеличить"** и **"Уменьшить"**
- Задайте первой кнопке атрибут **data-bind** со значением **"click: onClick(1)"**
- Задайте второй кнопке атрибут **data-bind** со значением **"click: onClick(-1)"**
- В HTML-элементе **body** создайте HTML-элемент **counter** и задайте ему атрибут **params** со значением **"inc: inc, dec: dec"**

Упражнение 2: Создание компонента и модели

- Зарегистрируйте компонент по имени **counter**
- Укажите для компонента свойство **viewModel** с содержанием:

```
self = this;  
this.inc = params.inc;  
this.dec = params.dec;
```
- Добавьте в свойство **viewModel** описание метода **onClick**, который принимает параметр **num**:

```
self.dec(parseInt(num));  
self.inc( self.inc() + parseInt(num) );
```
- Укажите для зарегистрированного компонента свойство **template** и укажите в качестве значения шаблон с **id="counter"**
- Создайте модель **ComponentViewModel** и опишите её как:

```
this.inc = ko.observable(1);  
this.dec = ko.observable(0);
```
- Осуществите привязку созданной модели с помощью метода **applyBindings**
- Запустите код и убедитесь, что при нажатии на кнопки числовое значение меняется в большую или меньшую сторону

Что мы изучили?

- Научились использовать шаблоны
- Научились использовать компоненты

Что почитать?

- Документация Knockout.js
- Примеры

Что дальше?

- Курсы по JavaScript