About the Tutorial

KnockoutJS is basically a library written in JavaScript, based on MVVM pattern that helps developers in building rich and responsive websites. KnockoutJS library provides an easy and clean way to handle complex data-driven interfaces. It is independent of any other framework.

This tutorial covers most of the topics required for a basic understanding of KnockoutJS and explains its various functionalities.

Audience

This tutorial is designed for software programmers who want to learn the basics of KnockoutJS and its programming concepts in a simple and easy way. This tutorial will give you enough understanding on the components of KnockoutJS with suitable examples.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of HTML, CSS, JavaScript, Document Object Model (DOM), and any text editor. As we are going to develop web-based application using KnockoutJS, it will be good if you have an understanding on how the Internet and web-based applications work.

Copyright & Disclaimer

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
# Table of Contents

About the Tutorial .............................................................................................................. i
Audience ............................................................................................................................... i
Prerequisites .......................................................................................................................... i
Copyright & Disclaimer ...................................................................................................... i
Table of Contents ................................................................................................................... ii

1. KNOCKOUTJS – OVERVIEW ......................................................................................... 1
2. KNOCKOUTJS – ENVIRONMENT SETUP ................................................................. 2
3. KNOCKOUTJS – APPLICATION .................................................................................... 6
   Architecture ...................................................................................................................... 6
4. KNOCKOUTJS – MVVM FRAMEWORK ........................................................................ 8
   Architecture ...................................................................................................................... 8
5. KNOCKOUTJS – OBSERVABLES .................................................................................. 10
   Reading and Writing Observables .................................................................................. 12
   Observable Arrays .......................................................................................................... 12
   ObservableArray Functions ........................................................................................... 13
   push() Method ............................................................................................................... 15
   pop() Method ................................................................................................................ 16
   unshift() Method .......................................................................................................... 18
   shift() Method ............................................................................................................ 20
   reverse() Method ........................................................................................................ 21
   sort() Method ............................................................................................................... 23
   splice() Method ........................................................................................................... 24
   indexOf() Method ....................................................................................................... 26
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice() Method</td>
<td>28</td>
</tr>
<tr>
<td>removeAll() Method</td>
<td>29</td>
</tr>
<tr>
<td>remove() Method</td>
<td>31</td>
</tr>
<tr>
<td>remove(function(item) { condition }) Method</td>
<td>32</td>
</tr>
<tr>
<td>remove([set of values]) Method</td>
<td>34</td>
</tr>
</tbody>
</table>

6. **KNOCKOUTJS – COMPUTED OBSERVABLES**

   - Pure Computed Observables | 36
   - Writable Computed Observables | 39

7. **KNOCKOUTJS – DECLARATIVE BINDINGS**

   - Binding Syntax | 42
   - Binding Context | 43
   - Visible Binding | 45
   - Text Binding | 46
   - Observations | 49
   - HTML Binding | 51
   - CSS Binding | 52
   - Style Binding | 54
   - Observations | 56
   - Attr Binding | 56
   - Foreach Binding | 59
   - Observations | 62
   - If Binding | 63
   - Observations | 65
   - Ifnot Binding | 66
   - with Binding | 68
   - Observations | 70
**Component Binding** .................................................................................................................. 70

**Component Processing Workflow** ............................................................................................ 71

**Observations** ............................................................................................................................... 75

**Click Binding** .................................................................................................................................. 77

**Observations** ..................................................................................................................................... 79

**Event Binding** .................................................................................................................................. 85

**Observations** ..................................................................................................................................... 87

**Submit Binding** .............................................................................................................................. 93

**Enable Binding** .............................................................................................................................. 95

**Disable Binding** ............................................................................................................................ 98

**Value Binding** ............................................................................................................................... 100

**Observations** ................................................................................................................................... 102

**textInput Binding** ......................................................................................................................... 105

**Observations** ................................................................................................................................... 106

**hasFocus Binding** ........................................................................................................................... 107

**checked Binding** ............................................................................................................................ 109

**options Binding** ............................................................................................................................. 113

**Observations** ................................................................................................................................... 115

**selectedOptions Binding** ............................................................................................................... 117

**uniqueName Binding** ....................................................................................................................... 119

8. **KNOCKOUTJS — DEPENDENCY TRACKING** .............................................................................. 122

**Controlling Dependencies Using Peek** ......................................................................................... 124

**Observations** ................................................................................................................................... 126

9. **KNOCKOUTJS — TEMPLATING** ................................................................................................. 127

**Observations** ................................................................................................................................... 128
10. KNOCKOUTJS – COMPONENTS

Component Registration ................................................................. 138
Component Binding ......................................................................... 143
Custom Element .............................................................................. 143
Component Loaders ........................................................................ 147
KnockoutJS is basically a library written in JavaScript, based on MVVM pattern that helps developers build rich and responsive websites. The model separates the application's Model (stored data), View (UI) and View Model (JavaScript Representation of model).

KnockoutJS was developed and is maintained as an open source project by Steve Sanderson, a Microsoft employee on July 5, 2010. KO is an abbreviation used for KnockoutJS. KO supports all mainstream browsers - IE 6+, Firefox 3.5+, Chrome, Opera, Safari (desktop/mobile).

**Features of KnockoutJS**

Here is a list of some of the most prominent features of KnockoutJS:

- **Declarative Binding**: HTML DOM elements are connected to the model through data-bind attribute using a very simple syntax. It is made easy to achieve responsiveness using this feature.

- **Automatic UI Refresh**: Any changes made to view the model data are reflected in the UI automatically and vice-versa. No need of writing extra code.

- **Dependency Tracking**: Relationship between KO attributes and KO library functions/components is transparent. Automatically tracks data changes in KO attribute and updates respective affected areas.

- ** Templating**: Templates are a simple and convenient way to build complex UI structures - with the possibility of repeating or nesting blocks - as a function of view model data.

- **Extensible**: Extends custom behavior very easily.

**Why Use KnockoutJS?**

- KnockoutJS library provides an easy and clean way to handle complex data-driven interfaces. One can create self-updating UIs for Javascript objects.

- It is pure JavaScript Library and works with any web framework. It's not a replacement of Jquery but can work as a supplement providing smart features.

- KnockoutJS library file is very small and lightweight.

- KnockoutJS is independent of any other framework. It is compatible with other client or server side technologies.

- Most important of all KnockoutJS is open source and hence free for use.
KnockoutJS is fully documented. The official site has full documentation including API docs, live examples, and interactive tutorials.
It is very easy to use KnockoutJS. Simply refer the JavaScript file using `<script>` tag in HTML pages.

Knockout.js can be accessed in the following ways:

- You can download production build of Knockout.js from its [official website](https://knockoutjs.com): A page as in the following image will be displayed. Click the download link and you will get the latest knockout.js file.

```
<script type='text/javascript' src='knockout-3.3.0.js'></script>
```

Now refer the file as shown in the following code.

```
<script type='text/javascript' src='knockout-3.3.0.js'></script>
```

Update the src attribute to match the location where the downloaded files are kept.

- You can refer to the KnockoutJS library from CDNs:
  - You can refer KnockoutJS library from Microsoft Ajax CDN in your code as follows:

```
<script src='https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js' type='text/javascript'></script>
```

  - Alternatively you can refer to a minified version of KnockoutJS library from CDNJS as follows:

```
<script src='https://cdnjs.cloudflare.com/ajax/libs/knockout/3.3.0/knockout-min.js' type='text/javascript'></script>
```

**Note:** In all the chapters for this tutorial, we have referred to CDN version of the KnockoutJS library.

**Example**
KnockoutJS is based on Model-View-ViewModel (MVVM) pattern. We will study this pattern in depth in chapter KnockoutJS - MVVM Framework. First let's take a look at a simple example of KnockoutJS.

```html
<!DOCTYPE html>
<head>
    <title>KnockoutJS Simple Example</title>
    <script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>

<body>
    <!-- This is called "view" of HTML markup that defines the appearance of UI -->

    <p>First String: <input data-bind="value: firstString" /></p>
    <p>Second String: <input data-bind="value: secondString" /></p>

    <p>First String: <strong data-bind="text: firstString">Hi</strong></p>
    <p>Second String: <strong data-bind="text: secondString">There</strong></p>
    <p>Derived String: <strong data-bind="text: thirdString"></strong></p>

    <script>
        <!-- This is called "viewmodel". This javascript section defines the data and behavior of UI -->

        function AppViewModel() {
            this.firstString = ko.observable("Enter First String");
            this.secondString = ko.observable("Enter Second String");

            this.thirdString = ko.computed(function() {
                return this.firstString() + " " + this.secondString();
            }, this);
        }

        // Activates knockout.js
        ko.applyBindings(new AppViewModel());
    </script>
```
The following line refers to KnockoutJS library.

```html
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"> </script>
```

We have two input boxes: **First String** and **Second String**. These 2 variables are initialized with values Enter First String and Enter Second String respectively in ViewModel.

```html
<p>First String: <input data-bind="value: firstString" /></p>
```

This is how we are binding values from ViewModel to HTML elements using 'data-bind' attribute in the body section.

Here, 'firstString' refers to ViewModel variable.

```javascript
this.firstString = ko.observable("Enter First String");
```

**ko.observable** is a concept which keeps an eye on the value changes so that it can update the underlying ViewModel data.

To understand this better, let's update the first input box to "Hello" and the second input box to "TutorialsPoint". You will see the values are updated simultaneously. We will study more about this concept in KnockoutJS - Observables chapter.

```javascript
this.thirdString = ko.computed(function() {
    return this.firstString() + " " + this.secondString();
}, this);
```

Next, we have computed function in viewmodel. This function derives the third string based on 2 strings mentioned earlier. Thus, any updates made to these strings automatically get reflected in this derived string. There is no need of writing an extra code to accomplish this. This is just a simple example. We will study about this concept in KnockoutJS - Computed Observables chapter.
Output
Save the above code as my_first_knockoutjs_program.html. Open this file in your browser and you will see an output as the following.

<table>
<thead>
<tr>
<th>First String:</th>
<th>Enter First String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second String:</td>
<td>Enter Second String</td>
</tr>
<tr>
<td>First String:</td>
<td>Enter First String</td>
</tr>
<tr>
<td>Second String:</td>
<td>Enter Second String</td>
</tr>
<tr>
<td>Derived String:</td>
<td>Enter First String Enter Second String</td>
</tr>
</tbody>
</table>

Modify strings to "Hello" and "TutorialsPoint" and the output changes as follows.

<table>
<thead>
<tr>
<th>First String:</th>
<th>Hello</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second String:</td>
<td>TutorialsPoint</td>
</tr>
<tr>
<td>First String:</td>
<td>Hello</td>
</tr>
<tr>
<td>Second String:</td>
<td>TutorialsPoint</td>
</tr>
<tr>
<td>Derived String:</td>
<td>Hello TutorialsPoint</td>
</tr>
</tbody>
</table>
KnockoutJS is widely used for Single Page Applications - A website created with the ability to retrieve all necessary data dynamically with a single page load reducing server round trips.

KnockoutJS is a client-side framework. This is a JavaScript library which makes it very easy to bind HTML to domain data. It implements a pattern called Model-View-ViewModel (MVVM). Observables is the magic ingredient of KnockoutJS. All data remains in sync because of Observable attribute.

**Architecture**

![Architecture Diagram](image)

**View**

View is nothing but user interface created using HTML elements and CSS styling.

You can bind HTML DOM elements to data model using KnockoutJS. It provides 2-way data binding between View and ViewModel using 'data-bind' concept, which means any updates done in the UI are reflected in the data model and any changes done in the data model are reflected in the UI. One can create self-updating UI with the help of knockoutJS.
**ViewModel**

ViewModel is a JavaScript object, which contains necessary properties and functions to represent data. View and ViewModel are connected together with declarative data-bind concept used in HTML. This makes it easy to change HTML without changing ViewModel. KnockoutJS takes care of automatic data refresh between them through the use of Observables.

Synchronization of data is achieved through binding DOM elements to Data Model, first using data-bind and then refreshing these 2 components through the use of Observables. Dependency tracking is done automatically due to this synchronization of data. No extra coding is required to achieve it. KnockoutJS allows to create direct connection between the display and underlying data.

You can create your own bindings called as custom bindings for application specific behaviors. This way Knockout gives direct control of how you want to transform your data into HTML.

**Model**

Model is the domain data on the server and it gets manipulated as and when the request is sent/received from ViewModel.

The data could be stored in database, cookie, or other form of persistent storage. KnockoutJS does not worry about how it is stored. It is up to the programmer to communicate between the stored data and KnockoutJS.

Most of the times, data is saved and loaded via an Ajax call.
Model-View-ViewModel (MVVM) is an architectural design pattern for developing software applications. MVVM was developed by Microsoft Architect John Gossman in 2005. This pattern is derived from Model-View-Controller (MVC) pattern. The advantage of MVVM is that it separates the application layer's graphical user interface from business logic. MVVM is responsible for handling data from the underlying model in such a way that it is represented and managed very easily. ViewModel in MVVM represents an abstract version of View's state and actions.

The view classes do not know that Model and ViewModel classes exists, also Model and ViewModel does not know that View exists. Model is also unaware that ViewModel and View exists.
Architecture

View

View is a Graphical User Interface created using markup language to represent data. View binds to properties of a ViewModel through data-bind concept, which indirectly connects to the model data. View need not be changed for any alteration done in ViewModel. Changes made to data in ViewModel is automatically propagated in View due to binding.

Model

Model is domain data or business object, which holds real-time data. Model does not carry behaviors. Behavior is mostly implemented in business logic.

ViewModel
ViewModel is the center place, where data from Model and View's display logic are bundled together. ViewModel holds the dynamic state of data. There is an implicit binder in between View and ViewModel to communicate with each other. This binding is inclusive of declarative data and command binding. Synchronization of View and ViewModel is achieved through this binding. Any change made in View is reflected in ViewModel, and similarly any change in ViewModel gets automatically reflected in View. Existence of this 2-way binding mechanism is a key aspect of this MVVM pattern.
KnockoutJS is build upon the following 3 important concepts.

- **Observables and dependency tracking between them** - DOM elements are connected to ViewModel via 'data-bind'. They exchange information through Observables. This automatically takes care of dependency tracking.

- **Declarative Bindings between UI and ViewModel** - DOM elements are connected to ViewModel via 'data-bind' concept.

- **Templatting to create re-usable components** - Templating provides a robust way to create complex web applications.

We will study Observables in this chapter.

As the name specifies, when you declare a ViewModel data/property as Observable, any data modification each time automatically gets reflected at all places the data is used. This also includes refreshing the related dependencies. KO takes care of these things and there is no need to write extra code to achieve this.

Using Observable, it becomes very easy to make UI and ViewModel communicate dynamically.

**Syntax**

You just need to declare ViewModel property with function `ko.observable()` to make it Observable.

```javascript
this.property = ko.observable('value');
```

**Example**

Let's take a look at the following example which demonstrates the use of Observable.

```html
<!DOCTYPE html>
<html>
<head>
<title>KnockoutJS Observable Example</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>

<!-- This is called "view" of HTML markup that defines the appearance of UI -->

<p>Enter your name: <input data-bind="value: yourName" /></p>
```

<p>Hi <strong data-bind="text: yourName"></strong> Good Morning!!!</p>

<script>
<!-- This is called "viewmodel". This javascript section defines the data and behavior of UI -->

function AppViewModel() {
    this.yourName = ko.observable('');
}

// Activates knockout.js
ko.applyBindings(new AppViewModel());
</script>
</body>
</html>

The following line is for the input box. As can be seen, we have used data-bind attribute to bind yourName value to ViewModel.

<p>Enter your name: <input data-bind="value: yourName" /></p>

The following line just prints the value of yourName. Note, that here data-bind type is the text as we are simply reading the value.

<p>Hi <strong data-bind="text: yourName"></strong> Good Morning!!!</p>

In the following line, ko.observable keeps an eye on yourName variable for any modification in data. Once there is a modification, the corresponding places also get updated with the modified value. When you run the following code, an input box will appear. As and when you update that input box, the new value will get reflected or refreshed in places wherever it is used.

this.yourName = ko.observable('');

**Output**

Let's carry out the following steps to see how the above code works:

- Save the above code in **first_observable_pgm.htm** file.
- Open this HTML file in a browser.
- Enter the name as Scott and observe that the name is reflected in the output.
Data modification can take place either from the UI or from ViewModel. Irrespective of from where the data is changed, the UI and ViewModel keeps synchronization among them. This makes it a two-way-binding mechanism. In the above example, when you change your name in the input box, ViewModel gets a new value. When you change yourName property from inside ViewModel, then the UI receives a new value.

## Reading and Writing Observables

Following table lists the read and write operations which can be performed on Observables.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Read/Write Operation &amp; Syntax</th>
</tr>
</thead>
</table>
| 1       | Read
To read value just call Observable property without parameters like:
AppViewModel.yourName(); |
| 2       | Write
To write/update value in Observable property, just pass the desired value in parameter like:
AppViewModel.yourName('Bob'); |
| 3       | Write multiple
Multiple ViewModel properties can be updated in a single row with the help of chaining-syntax like:
AppViewModel.yourName('Bob').yourAge(45); |

## Observable Arrays

Observable declaration takes care of data modifications of a single object. ObservableArray works with the collection of objects. This is a very useful feature when you are dealing with
complex applications containing multiple type of values and changing their status frequently based on the user actions.

**Syntax**

```
this.arrayName = ko.observableArray(); // It's an empty array
```

Observable array only tracks which objects in it are added or removed. It does not notify if the individual object’s properties are modified.

**Initialize It for the First Time**

You can initialize your array and at the same time you declare it as Observable by passing the initial values to the constructor as follows.

```
this.arrayName = ko.observableArray(['scott','jack']);
```

**Reading from Observable Array**

You can access Observable array elements as follows.

```
alert('The second element is ' + arrayName()[1]);
```

**ObservableArray Functions**

KnockoutJS has its own set of Observable array functions. They are convenient because:

- These functions work on all browsers.
- These functions will take care of dependency tracking automatically.
- Syntax is easy to use. For example, to insert an element into an array, you just need to use `arrayName.push('value')` instead of `arrayName().push('value')`.

Following is the list of various Observable Array methods.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Methods &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>push('value')</code></td>
</tr>
<tr>
<td></td>
<td>Inserts a new item at the end of array.</td>
</tr>
<tr>
<td>2</td>
<td><code>pop()</code></td>
</tr>
<tr>
<td></td>
<td>Removes the last item from the array and returns it.</td>
</tr>
<tr>
<td>3</td>
<td><code>unshift('value')</code></td>
</tr>
<tr>
<td></td>
<td>Inserts a new value at the beginning of the array.</td>
</tr>
<tr>
<td></td>
<td>Method</td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td>4</td>
<td><code>shift()</code></td>
</tr>
<tr>
<td>5</td>
<td><code>reverse()</code></td>
</tr>
<tr>
<td>6</td>
<td><code>sort()</code></td>
</tr>
<tr>
<td>7</td>
<td><code>splice(start-index,end-index)</code></td>
</tr>
<tr>
<td>8</td>
<td><code>indexOf('value')</code></td>
</tr>
<tr>
<td>9</td>
<td><code>slice(start-index,end-index)</code></td>
</tr>
<tr>
<td>10</td>
<td><code>removeAll()</code></td>
</tr>
<tr>
<td>11</td>
<td><code>remove('value')</code></td>
</tr>
<tr>
<td>12</td>
<td><code>remove(function(item) { condition })</code></td>
</tr>
<tr>
<td>13</td>
<td><code>remove([set of values])</code></td>
</tr>
<tr>
<td>14</td>
<td><code>destroyAll()</code></td>
</tr>
<tr>
<td>15</td>
<td><code>destroy('value')</code></td>
</tr>
<tr>
<td>16</td>
<td><code>destroy(function(item) { condition})</code></td>
</tr>
<tr>
<td>17</td>
<td><code>destroy([set of values])</code></td>
</tr>
</tbody>
</table>
**Note**: Destroy and DestroyAll Functions from ObservableArrays are mostly for 'Ruby on Rails' developers only.

When you use destroy method, the corresponding items are not really deleted from array at that moment but are made hidden by marking them with property `_destroy` with true value so that they can't be read by UI. Items marked as _destroy equal to true are deleted later while dealing with JSON object graph.

### push() Method

**Description**
The KnockoutJS Observable `push('value')` method inserts a new item at the end of an array.

**Syntax**
```
arrayName.push('value')
```

**Parameters**
Accepts only one parameter, that is the value to be inserted.

**Example**
```
<!DOCTYPE html>
<head>
<title>KnockoutJS Observable Array push() Method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>

<p>Example to demonstrate push() method.</p>

<p>Enter name: <input data-bind='value: empName' /></p>

<p><button data-bind="click: addEmp">Add Emp</button></p>

<p>Array of employees: <span data-bind='text: empArray()'></span></p>

<script>
function EmployeeModel(){
    this.empName = ko.observable('');
    this.chosenItem = ko.observableArray('');

    this.addEmp = function(){
        this.empName += this.chosenItem;
    }
}

var model = new EmployeeModel();
model.addEmp();
</script>
```
this.empArray = ko.observableArray(['Scott', 'James', 'Jordan', 'Lee', 'RoseMary', 'Kathie']);
//Initial Values

this.addEmp = function() {
    if (this.empName() != "") {
        this.empArray.push(this.empName()); //insert accepted value in array
        this.empName("");
    }
}.bind(this);

var emp = new EmployeeModel();
ko.applyBindings(emp);
</script>
</html>

Output

Let's carry out the following steps to see how the above code works:

- Save the above code in array-push.htm file.
- Open this HTML file in a browser.
- Type 'Tom' as an input and click the Add Emp button.

Example to demonstrate push() method.

Enter name: 

Add Emp

Array of employees: Scott, James, Jordan, Lee, RoseMary, Kathie

pop() Method

Description
The KnockoutJS Observable `pop()` method removes the last item from an array and returns it.

**Syntax**

```javascript
arrayName.pop()
```

**Parameters**

Does not accept any parameters.

**Example**

```html
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray pop method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>
  <p>Example to demonstrate pop() method.</p>
  <button data-bind="click: popEmp">Remove Emp</button>
  <p>Array of employees: <span data-bind="text: empArray()"></span></p>
  
  <script>
    function EmployeeModel()
    {
      this.empName = ko.observable("");
      this.chosenItem = ko.observableArray("");
      this.empArray = ko.observableArray(['Scott', 'James', 'Jordan', 'Lee', 'RoseMary', 'Kathie']);

      this.popEmp = function() {
        this.empArray.pop();
      }
    }
    var em = new EmployeeModel();
    ko.applyBindings(em);
  </script>
</body>
```
Output
Let's carry out the following steps to see how the above code works:

- Save the above code in `array-pop.htm` file.
- Open this HTML file in a browser.
- Click the Remove Emp button and observe that the last element is removed.

Example to demonstrate `pop()` method.

Click the Remove Emp button and observe that the last element is removed.

Array of employees: Scott, James, Jordan, Lee, Rose Mary, Kathie

unshift() Method

Description
The KnockoutJS Observable `unshift('value')` method inserts a new item at the beginning of the array.

Syntax

`arrayName.unshift('value')`

Parameters
Accepts one parameter, that is the value to be inserted.

Example

```html
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray unshift method</title>
</head>
```
Example to demonstrate unshift() method.
Enter name: <input data-bind='value: empName'/>
<button data-bind="click: unshiftEmp">Add Emp in Beginning</button><br><br>
Array of employees: <span data-bind='text: empArray()'></span>

```javascript
function EmployeeModel()
{
    this.empName = ko.observable();
    this.chosenItem = ko.observableArray();
    this.empArray = ko.observableArray(['Scott','James','Jordan','Lee','RoseMary','Kathie']);

    this.unshiftEmp = function() {
        if (this.empName() !== '') {
            this.empArray.unshift(this.empName()); // insert at the beginning
            this.empName('');
        }
    }.bind(this);

    var em = new EmployeeModel();
    ko.applyBindings(em);
}
</script>
```

**Output**

Let's carry out the following steps to see how the above code works:

- Save the above code in **array-unshift.htm** file.
- Open this HTML file in a browser.
- Enter name as Tom and click the Add Emp in Beginning button.
**shift() Method**

**Description**
The KnockoutJS Observable `shift()` method removes the first item from the array and returns it.

**Syntax**

```
arrayName.shift()
```

**Parameters**

Does not accept any parameter.

**Example**

```
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray shift method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>
<p>Example to demonstrate `shift()` method.</p>
<button data-bind="click: shiftEmp">Remove First Emp</button>
<p>Array of employees: <span data-bind="text: empArray()"></span></p>
<script>
```
```
function EmployeeModel(){
    this.empName = ko.observable("" unlawing); 
    this.chosenItem = ko.observableArray(""不得使用""); 
    this.empArray = ko.observableArray(['Scott','James','Jordan','Lee','RoseMary','Kathie']); 

    this.shiftEmp = function() { 
        this.empArray.shift(); // remove first item 
    } 
} 
var em = new EmployeeModel(); 
ko.applyBindings(em); 
}) </script> </body> </html>

Output
Let's carry out the following steps to see how the above code works:

- Save the above code in array-shift.htm file.
- Open this HTML file in a browser.
- Click the Remove First Emp button and observe that the first element is removed.

Example to demonstrate shift() method.

[Remove First Emp]

Array of employees: Scott, James, Jordan, Lee, RoseMary, Kathie

reverse() Method

Description
The KnockoutJS Observable reverse() method reverses the order of the array.
**Syntax**

`arrayName.reverse()`

**Parameters**

Does not accept any parameter.

**Example**

```html
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray reverse method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>

<p>Example to demonstrate reverse() method.</p>
<button data-bind="click: revEmp">Reverse Array</button>
<p>Array of employees: <span data-bind="text: empArray()"></span></p>

<script>
function EmployeeModel()
{
    this.empName = ko.observable("");
    this.chosenItem = ko.observableArray(" ");
    this.empArray = ko.observableArray(['Scott','James','Jordan','Lee','RoseMary','Kathie']);

    this.revEmp = function() {
        this.empArray.reverse(); // reverse order
    }
}

var em = new EmployeeModel();
ko.applyBindings(em);
</script>
</body>
</html>
```
Output
Let's carry out the following steps to see how the above code works:

- Save the above code in `array-reverse.htm` file.
- Open this HTML file in a browser.
- Click the Reverse Array button and observe that the array order is changed.

Example to demonstrate reverse() method.

```
Reverse Array
```

Array of employees: Scott, James, Jordan, Lee, RoseMary, Kathie

sort() Method

Description
The KnockoutJS Observable `sort()` method sorts all items in the array.

By default, items are sorted in an ascending order. For sorting an array in a descending order, use reverse() method on sorted array.

Syntax

```
arrayName.sort()
```

Parameters

Does not accept any parameter.

Example

```
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray sort method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
```
Example to demonstrate sort() method.

Let's carry out the following steps to see how the above code works:

- Save the above code in `array-sort.htm` file.
- Open this HTML file in a browser.
- Click the Sort Array button and see that the array is sorted.
splice() Method

Description
The KnockoutJS Observable splice() method takes 2 parameters specifying the start-index and the end-index. It removes items starting from start to end index and returns them as an array.

Syntax
arrayName.splice(start-index, end-index)

Parameters
Accepts 2 parameters, start-index is start index and end-index is end index.

Example
```html
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray splice method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>
<p>Example to demonstrate splice() method.</p>
<button data-bind="click: spliceEmp">Splice Emp</button>
<p>Array of employees: <span data-bind="text: empArray()"></span></p>
</body>
<script>
</script>
```
function EmployeeModel()
{
    this.empName = ko.observable('');
    this.chosenItem = ko.observableArray('');
    this.empArray = ko.observableArray(['Scott', 'James', 'Jordan', 'Lee', 'RoseMary', 'Kathie']);

    this.spliceEmp = function()
    {
        alert("Splice is removing items from index 1 to 3(If exists.).");
        this.empArray.splice(1,3);  // remove 2nd,3rd and 4th item, as array index starts with 0.
    }
}
var em = new EmployeeModel();
ko.applyBindings(em);
</script>
</body>
</html>

### Output

Let's carry out the following steps to see how the above code works:

- Save the above code in **array-splice.htm** file.
- Open this HTML file in a browser.
- Click the Splice Emp button and observe that items starting from index 1 to 3 are removed.
indexOf() Method

Description
The KnockoutJS Observable indexOf('value') method returns the index of the first occurrence of the parameter provided. This function will return -1, if no matching element is found.

Syntax

arrayName.indexOf('value')

Parameters
Accepts 1 parameter, whose index will be returned.

Example

```html
<!DOCTYPE html>
<head>
<title>KnockoutJS ObservableArray indexOf method</title>
<script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-3.1.0.js" type="text/javascript"></script>
</head>
<body>
<p>Example to demonstrate indexOf() method.</p>
<p>Index of Employee 'Jordan':<span data-bind="text: empArray().indexOf('Jordan')"></span></p>
<p>Array of employees: <span data-bind="text: empArray()"></span></p>
<script>
function EmployeeModel(){
 this.empName = ko.observable('');
 this.chosenItem = ko.observableArray('');
 this.empArray = ko.observableArray(['Scott','James','Jordan','Lee','RoseMary','Kathie']);
}
var em = new EmployeeModel();
ko.applyBindings(em);
</script>
```
Output
Let's carry out the following steps to see how the above code works:

- Save the above code in `array-indexof.htm` file.
- Open this HTML file in a browser.

Example to demonstrate `indexOf()` method.

Index of Employee 'Jordan': 2

Array of employees: Scott, James, Jordan, Lee, RoseMary, Kathie