KnockoutJS Quick Reference

1. Knockout is a JavaScript Library (as opposed to Backbone.js which is a framework) that helps you improve the User Experience of your web application.

2. Knockout provides two way data-binding using a ViewModel and provides DOM Templating, it doesn't deal with sending data over to server or routing.

3. You use Knockout as a drop-in enhancement library to improve usability and user experience, whereas Framework like Backbone is used to group up in new applications (Single Page Apps is a good example).

4. You can create a KO ViewModel as a JavaScript object or as a Function (known as prototype). It's outside the jQuery document ready.

5. ViewModel as a function

```
var myViewModel = function() {
this.Email = "myemail@email.com";
this.Name = "Sumit";
this.LastName = "Maitra";
this.WebSite = "http://www.devcurry.com";
}
```

6. ViewModel as an object: you can follow the JavaScript Object Notation –

```
var myViewModel = {
Email: "myemail@email.com",
Name: "Sumit",
LastName: "Maitra"
}
```

7. Applying a ViewModel

```
// In case the View Model is defined as a function
ko.applyBindings(new myViewModel());
// In case the View Model is defined as a JSON Object
ko.applyBindings(myViewModel);
```

Here ko is the global reference to Knockout that you get once you add Knockout Script reference in your page.

8. Observable Properties: KO has the concept of Observable properties. If you define a property as an observable, then DOM elements bound to it will be updated as soon as the property changes.

```
e.g.
var myViewMode = {
Email: ko.observable("myemail@gmail.com")
}
```

9. Observable Arrays: When you define an array of Json objects as Observable, KO refreshes DOM elements bound to it, when items are added or removed from the Array.

```
var orderViewModels = function() {
…
// Binds to an empty array
this.Address = ko.observableArray([]);
}
```

Note: When properties of an Item in the array changes, then KO does not raise 'modified' events.

10. Simple Data Binding to properties

```
<input data-bind="value : Email" />
```

Value of input element to the Email in the ViewModel.

11. Binding to Computed Values: Binding KO to a function gives you added flexibility of defining methods that do computation and return a computed value. KO can actually bind DOM elements to these methods as well. For example, if we wanted to bind First Name and Last Name and show it in a single DOM element, we could do something like this:

In the View Model

```
var myViewModel = function() {
this.Email = "myemail@email.com";
this.Name = "Sumit";
this.LastName = "Maitra";
```

```
this.FullName = ko.computed(function () {
  return this.LastName() + “, ” + this.Name();
}, this);
}
```

Binding to DOM Element

```
<label data-bind= “text: FullName” />
```

12. Observable Properties are functions: When dealing with observable values, you have to end the property with parenthesis like a function call, because computed values are actually functions that need to be evaluated on bind.

13. Assign value to observables: We saw how to declare observable Properties above. But when we have to assign a value to an observable in JavaScript, we assign it as if we are calling a function, for example:

myViewModel.Email(“myNewEmail@email.com”);

14. Binding to an array of objects: KO can bind the DOM to an array in the view Model. We use KO’s foreach syntax as follows

```
<table>
& lt;tbody data-bind= “foreach: Address”>
  …
</tbody>
& lt;/table>
```

You can do the same for an Ordered <ol> List or an Unordered List <ul> too. Once you have done the foreach binding, KO treats anything DOM element inside as a part of a template that’s repeated as many times as the number of elements items in the list to which it I bound.

15. Binding elements of an array: Once you have bound an Array to a DOM element, KO gives you each element in the array to bind against. So an Address object may be bound as follows:

```
<table>
& lt;tbody data-bind= “foreach: Address”>
  <tr>
   <td>
```

```
     Street: <label data-bind: “text: Street” />
     #: <label data-bind: “text: Number” />
     City: <label data-bind: “text: City” />
     State: <label data-bind: “text: State” />
     Country: <label data-bind: “text: Country” />
   </td>
  </tr>
& lt;/tbody>
& lt;/table>
```

16. **Binding to properties other than text and value:** Now let’s say we want to bind the WebSite element of the ViewModel an anchor tag

```
<a data-bind=“attr : {href : WebSite}”>
& lt;span data-bind= “text: Name”</a>
```

What we are doing here is using Knockout’s attribute binding technique to bind the ‘href’ attribute to the URL in the WebSite property of the View Model. Using attribute binding we can bind to any HTML attribute we want to.

17. **Scoping inside a foreach binding:** Let’s say we have a utility method in the view model to concatenate the Street, Number and City properties of each address.

```
var myViewModel = function() {
this.Email = “myemail@email.com”;
this.Name = “Sumit”;
this.LastName = “Maitra”;
this.Address = ko.observableArray([]);
this.FullName = ko.computed(function ()
  return this.LastName() + “, ” + this.Name();  }, this);
this.AddressString = function (street. Number, city)
{
  return street + ‘,’ + number + ‘,’ + city;
}
}
```

Now in the table we can update the binding as follows:

```
<table>
& lt;tbody data-bind= “foreach: Address”>
```

```
  <tr>
   <td>
     Address: <label data-bind: “text: $parent.AddressString(Street, Number,
City)” />
     State: <label data-bind: “text: State” />
     Country: <label data-bind: “text: Country” />
   </td>
  </tr>
& lt;/tbody>
& lt;/table>
```

Note the $parent prefix. This is done because inside the foreach, KO has reduced the scope to the Address object and not the entire ViewModel. So to call the AddressString method, we need to prefix the $parent telling KO to look at the view model where the AddressString method actually is.

18. Getting KO Context in jQuery: Now let’s say we want to have a Delete button for each address. The following markup will add a button

```
<table>
& lt;tbody class=“addressList” data-bind= “foreach: Address”>
  <tr>
   <td>
     Address: <label data-bind: “text: $parent.AddressString(Street, Number,
City)” />
     State: <label data-bind: “text: State” />
     Country: <label data-bind: “text: Country” />
   </td>
   <td>
    <button class= “addressDeleter” ></button>
   </td>
  </tr>
& lt;/tbody>
& lt;/table>
```

Now to handle the click event using jQuery. Since the button is generated on KO binding, we cannot use the normal click handler assignment of jQuery. Instead we use the parent container and assign a delegate as follows

```
$(“#addressList”).delegate(“.noteDeleter”, “click”, function() {
var address = ko.dataFor(this);
```

```
// send the address to the server and delete it
});
```

As we can see above the ko.dataFor(this) helper method in KO returns the object that was bound to that particular row of data. So it returns an Address object. If you need the entire ViewModel you can use ko.contextFor(this).