Knockout.js Cheat Sheet

A quick reference guide to Knockout.js, covering observables, bindings, and core functionalities for building dynamic JavaScript UIs.

Core Concepts

Observables

`ko.observable( value )`

Creates an observable, a special JavaScript object with a `value` property that notifies subscribers when changed.

Example:

```
var myObservable = ko.observable('Initial Value');
myObservable(); // Returns 'Initial Value'
myObservable('New Value'); // Sets the value to 'New Value'
```

`ko.observableArray( array )`

Creates an observable array, tracking which objects are in the array and notifying listeners when items are added, moved, or deleted.

Example:

```
var myArray = ko.observableArray(['Item 1', 'Item 2']);
myArray.push('Item 3');
```

`subscribe( callback, target, event )`

Subscribes to changes in an observable. Executes the `callback` function whenever the observable's value changes. `target` and `event` are optional.

Example:

```
myObservable.subscribe(function(newValue) {
  console.log('The new value is ' + newValue);
});
```

Computed Observables

`ko.computed( function, target, options )`

Creates a computed observable whose value is dependent on other observables and automatically updates when dependencies change.

Example:

```
var firstName = ko.observable('John');
var lastName = ko.observable('Doe');

var fullName = ko.computed(function() {
  return firstName() + ' ' + lastName();
});

console.log(fullName()); // Outputs: John Doe
```

Options for `ko.computed`:

- `read`: Function to compute the value (default).
- `write`: Function to handle writes to the computed observable.
- `pure`: Indicates that the computed value is solely determined by its dependencies (default: false).
- `deferEvaluation`: Boolean indicating whether the computed observable should only be evaluated when accessed (default: false).

Binding Basics

`data-bind` Attribute

The cornerstone of Knockout.js. Links elements in your HTML to properties in your view model.

Example:

```
<span data-bind="text: fullName"></span>
```

Common Bindings

Text and Value Bindings

`text`	Displays the value of an observable as text within an element. **Example:** `<span data-bind="text: myObservable"></span>`
`value`	Binds the value of a form element (e.g., input, textarea) to an observable. Supports two-way binding. **Example:** `<input type="text" data-bind="value: myObservable" />`

Visibility and CSS Bindings

`visible`	Controls the visibility of an element based on an observable value (true/false). **Example:** `<div data-bind="visible: isVisible">This is visible</div>`
`css`	Applies CSS classes to an element based on an observable or an object of observable key/value pairs. **Example:** `<div data-bind="css: { highlighted: isHighlighted }"></div>`
`style`	Applies inline styles to an element based on an observable or an object of observable key/value pairs. **Example:** `<div data-bind="style: { color: textColor }"></div>`

Control Flow Bindings

`if`	Conditionally displays an element based on an observable value. Removes the element from the DOM if the value is false. **Example:** `<div data-bind="if: isLoggedIn">Welcome, user!</div>`
`ifnot`	The opposite of `if`. Displays an element only if the observable value is false. **Example:** `<div data-bind="ifnot: isLoggedIn">Please log in.</div>`

`foreach`	Repeats a section of markup for each item in an observable array.
	Example:
	`<ul data-bind="foreach: items">`
	`  <li data-bind="text: $data"></li>`
	`</ul>`

Event Handling & Advanced Bindings

Event Bindings

`click`

Binds a function to the click event of an element.

Example:

```
<button data-bind="click:
myClickHandler">Click Me</button>


var viewModel = {
  myClickHandler: function() {
    alert('Button clicked!');
  }
};
```

`submit`

Binds a function to the submit event of a form.

Example:

```
<form data-bind="submit: mySubmitHandler">
  ...
</form>
```

Template Binding

`template`

Renders a template with data from your view model. Templates can be inline or defined in separate script elements.

Example (Inline Template):

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

<script type="text/html" id="myTemplate">
  <span data-bind="text: name"></span>
</script>
```

Custom Bindings

`ko.bindingHandlers`

Allows you to define your own custom bindings to encapsulate reusable UI logic.

Example:

```
ko.bindingHandlers.fadeVisible = {
  init: function(element, valueAccessor) {
    // Initially set the element to be
instantly visible/hidden depending on the
value
    var shouldDisplay = valueAccessor();
    $(element).toggle(shouldDisplay);
  },
  update: function(element, valueAccessor) {
    // Whenever the value subsequently
changes, slowly fade the element in or out
    var shouldDisplay = valueAccessor();
    shouldDisplay ? $(element).fadeIn() :
$(element).fadeOut();
  }
};


<div data-bind="fadeVisible: isVisible"></div>
```

Utilities and Extensions

Utilities

`ko.utils.arrayForEach( array, callback )`

Iterates over an array, executing a callback for each item.

Example:

```
ko.utils.arrayForEach(['A', 'B', 'C'], function(item) {
  console.log(item);
});
```

`ko.utils.arrayMap( array, callback )`

Transforms an array by applying a callback function to each item and returning a new array with the results.

Example:

```
var doubled = ko.utils.arrayMap([1, 2, 3], function(item) {
  return item * 2;
});
console.log(doubled); // Outputs: [2, 4, 6]
```

Extenders

`extenders`

Allow you to add custom functionality to observables.

Example:

```
ko.extenders.numeric = function(target, precision) {
  var result = ko.computed({
    read: target,
    write: function(newValue) {
      var current = target(),
          roundingMultiplier = Math.pow(10, precision),
          newValueAsNum = isNaN(newValue) ? 0 : parseFloat(+newValue),
          valueToWrite = Math.round(newValueAsNum * roundingMultiplier) /
roundingMultiplier;

      if (valueToWrite !== current) {
        target(valueToWrite);
      } else {
        if (newValue !== current) {
          target.notifySubscribers(valueToWrite);
        }
      }
    }
  }).extend({ notify: 'always' });

  result(target());
  return result;
};

var price = ko.observable(123.456).extend({ numeric: 2 });
```