



มหาวิทยาลัยขอนแก่น

วิทยา ชริยา มัญญา



KHON KAEN UNIVERSITY

# Introduction to Vue.js

Assoc. Prof. Dr. Kanda Runapongsa Saikaew

(krunapon@kku.ac.th)

Department of Computer Engineering

Khon Kaen University



คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น  
FACULTY OF ENGINEERING KHON KAEN UNIVERSITY

# What is Vue.js?

- Vue (pronounced /vju:/, like **view**) is a **progressive framework** for building user interfaces
- The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects
- On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with **modern tooling** and **supporting libraries**.



# Hello Vue.js!

- <https://jsfiddle.net/chrisvfritz/50wL7mdz/>

HTML ▾

```
1 <script src="https://unpkg.com/vue"></script>
2
3 <div id="app">
4   <p>{{ message }}</p>
5 </div>
6
```

CSS ▾

```
1
```

JavaScript + No-Library (pure JS) ▾

```
1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Hello Vue.js!'
5   }
6 })
```

Hello Vue.js!



คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น  
FACULTY OF ENGINEERING KHON KAEN UNIVERSITY

# Declarative Rendering

- At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax

```
<div id="app">  
  {{ message }}  
</div>
```

HTML

```
var app = new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello Vue!'  
  }  
})
```

JS

Hello Vue!



# Reactive

- The data and the DOM are now linked, and everything is now **reactive**
- How do we know? Open your browser's JavaScript console and set app.message to a different value
- You should see the rendered example above update accordingly.



# Ex1: helloworld.html

A screenshot of a web browser with developer tools open. The browser window shows the URL "localhost:8000/helloworld.html" and the text "I have changed the data!". Below the browser is the developer tools interface. The "Console" tab is selected. The console output includes a message about Vue Devtools, a note about development mode, and a transcript of the JavaScript console interactions:

```
Download the Vue Devtools extension for a vue.js:8542
better development experience:
https://github.com/vuejs/vue-devtools
You are running Vue in development mode. vue.js:8553
Make sure to turn on production mode when deploying
for production.
See more tips at https://vuejs.org/guide/deployment.html
> app.message = "I have changed the data!"
< "I have changed the data!"
>
```



# Binding element attributes

HTML

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

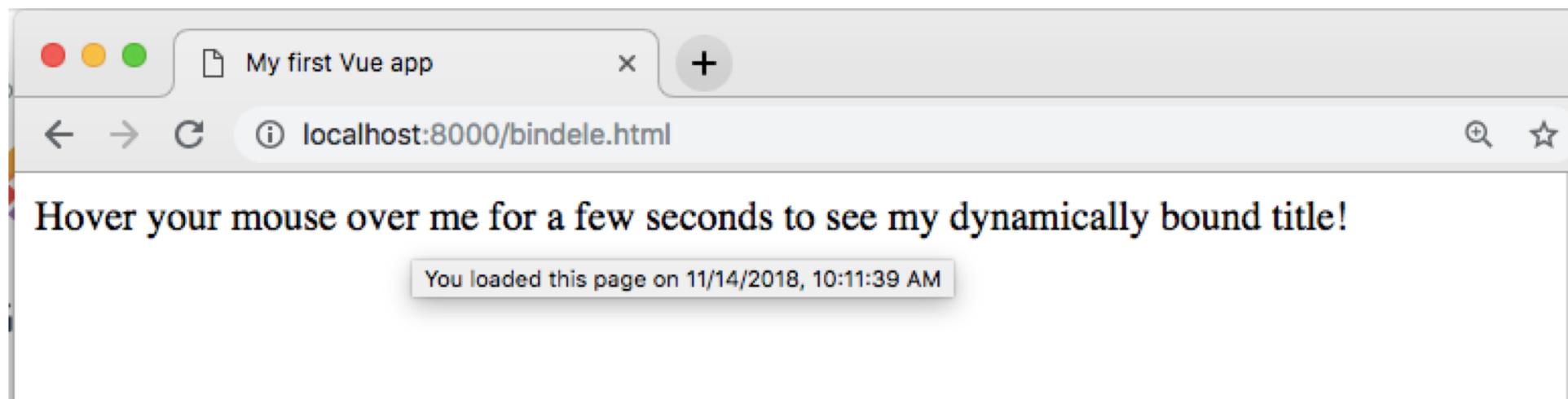
JS

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date().toLocaleString()
  }
})
```

Hover your mouse over me for a few seconds to see my  
dynamically bound title!



# The result of binding element attributes



# Directive

- The v-bind attribute you are seeing is called a **directive**
- Directives are prefixed with v- to indicate that they are special attributes provided by Vue
  - They apply special reactive behavior to the rendered DOM
  - Here, it is basically saying “keep this element’s title attribute up-to-date with the message property on the Vue instance



# Conditionals

HTML

```
<div id="app-3">  
  <span v-if="seen">Now you see me</span>  
</div>
```

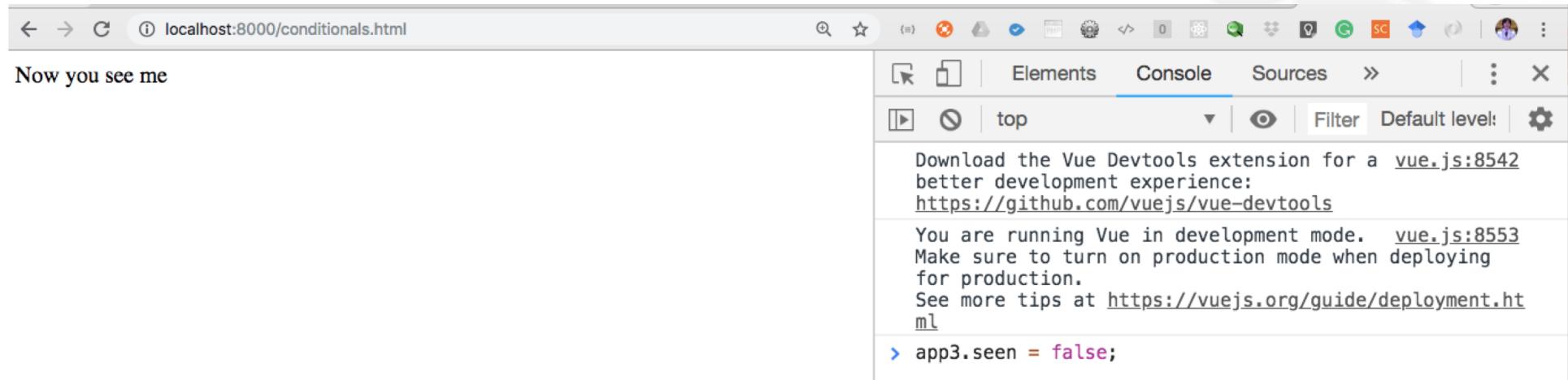
JS

```
var app3 = new Vue({  
  el: '#app-3',  
  data: {  
    seen: true  
  }  
})
```

Now you see me

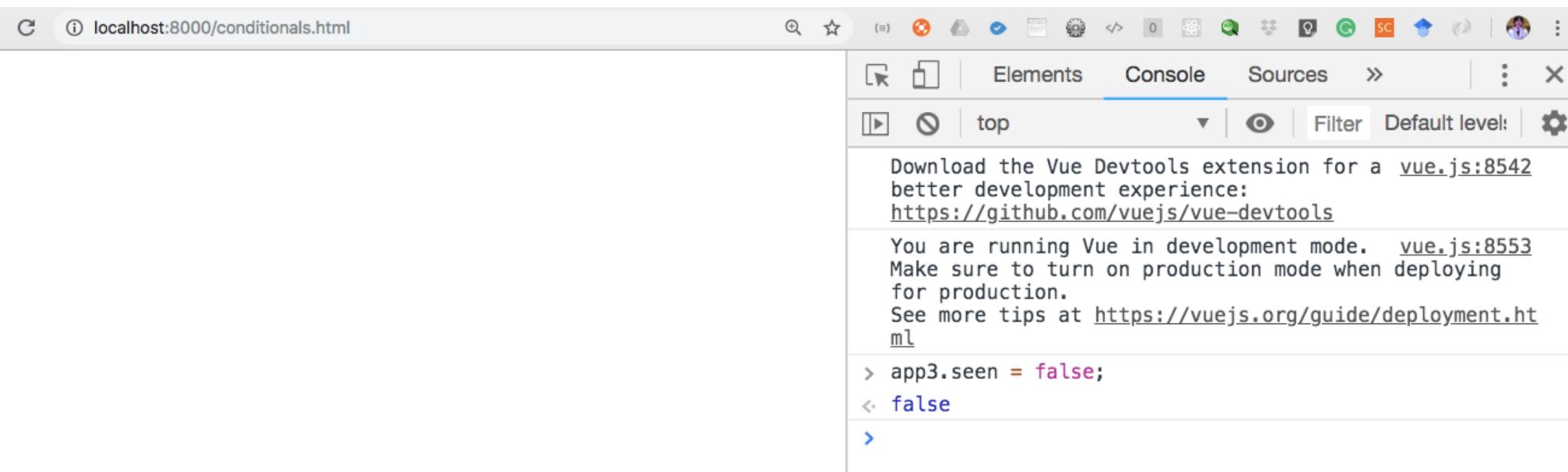


# Using conditionals



A screenshot of a web browser window displaying a local file at `localhost:8000/conditionals.html`. The page content is "Now you see me". The browser's developer tools are open, specifically the "Console" tab. The console output shows the following:

```
Download the Vue Devtools extension for a vue.js:8542 better development experience:  
https://github.com/vuejs/vue-devtools  
You are running Vue in development mode. vue.js:8553 Make sure to turn on production mode when deploying for production.  
See more tips at https://vuejs.org/guide/deployment.html  
> app3.seen = false;
```



A screenshot of a web browser window displaying a local file at `localhost:8000/conditionals.html`. The page content is "Now you see me". The browser's developer tools are open, specifically the "Console" tab. The console output shows the following:

```
Download the Vue Devtools extension for a vue.js:8542 better development experience:  
https://github.com/vuejs/vue-devtools  
You are running Vue in development mode. vue.js:8553 Make sure to turn on production mode when deploying for production.  
See more tips at https://vuejs.org/guide/deployment.html  
> app3.seen = false;  
< false  
>
```



# Loops

- The v-for directive can be used for displaying a list of items using the data from an Array

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```
JS
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})
```

1. Learn JavaScript
2. Learn Vue
3. Build something awesome



# Ex 2: Update an element in loops dynamically

A screenshot of a web browser window titled "localhost:8000/loops.html". The page content displays a list of three items:

- 1. Learn JavaScript
- 2. Learn Vue
- 3. Build something awesome

The browser's developer tools are open, specifically the "Console" tab. It shows the following message:

```
You are running Vue in development mode. vue.js:8553
Make sure to turn on production mode when deploying
for production.
See more tips at https://vuejs.org/guide/deployment.ht
ml
```

A screenshot of a web browser window titled "localhost:8000/loops.html". The page content displays a list of four items:

- 1. Learn JavaScript
- 2. Learn Vue
- 3. Build something awesome
- 4. Build something easy

The browser's developer tools are open, specifically the "Console" tab. It shows the following message:

```
You are running Vue in development mode. vue.js:8553
Make sure to turn on production mode when deploying
for production.
See more tips at https://vuejs.org/guide/deployment.ht
ml
```



# Handling user input

- To let users interact with your app, we can use the v-on directive to attach event listeners that invoke methods on our Vue instances:

```
HTML  
<div id="app-5">  
  <p>{{ message }}</p>  
  <button v-on:click="reverseMessage">Reverse Message</button>  
</div>
```

```
JS  
var app5 = new Vue({  
  el: '#app-5',  
  data: {  
    message: 'Hello Vue.js!'  
  },  
  methods: {  
    reverseMessage: function () {  
      this.message = this.message.split('').reverse().join('')  
    }  
  }  
)
```

Hello Vue.js!

Reverse Message

# The result of handling user input



Hello Vue.js!

Reverse Message



!sj.euV olleH

Reverse Message

Note that in this method we update the state of our app without touching the DOM - all DOM manipulations are handled by Vue, and the code you write is focused on the underlying logic.



# Two-way binding between form input and app

- Vue also provides the `-model` directive that makes two-way binding between form input and app state a breeze

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

HTML

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

JS

Good bye

Good bye

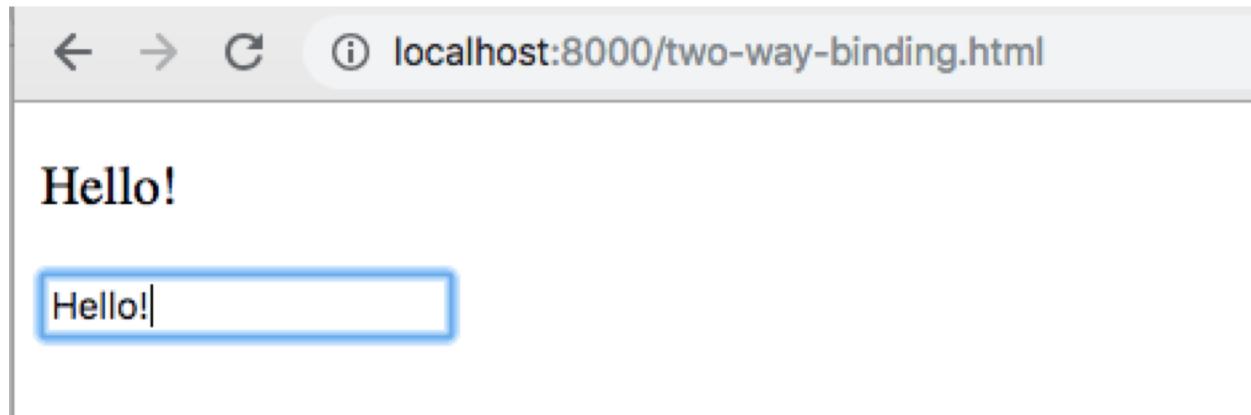


# The result of two-way binding



Hello Vue!

Hello Vue!



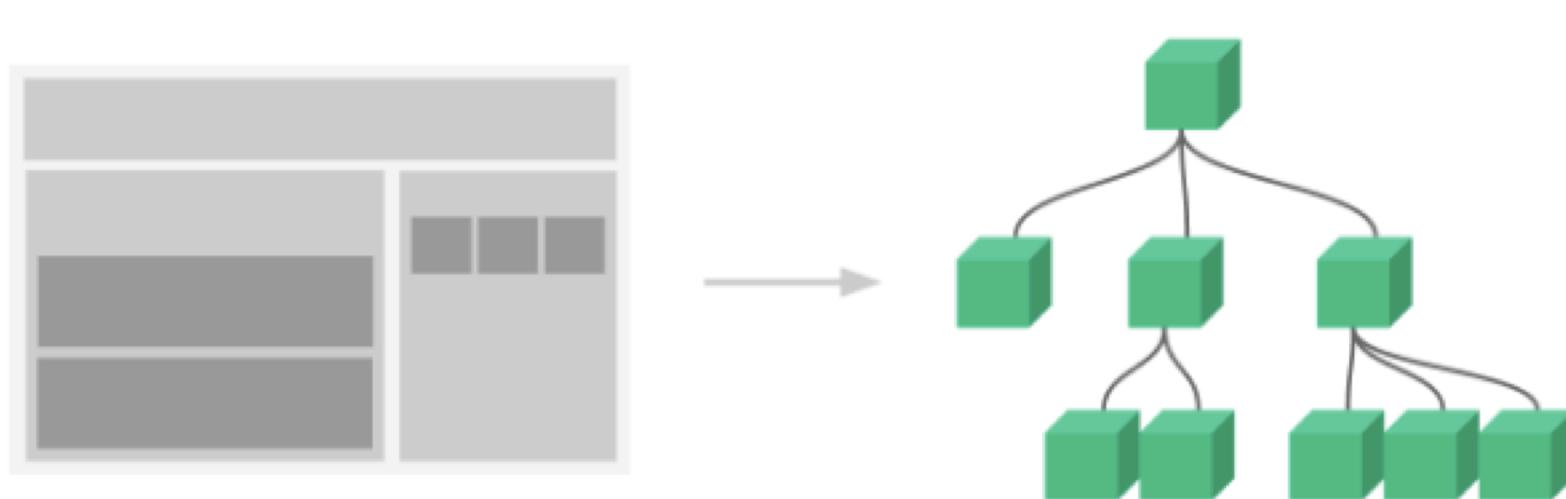
Hello!

Hello!



# Composing with components

- The component system is another important concept in Vue, because it's an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components



```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>My first Vue app</title>
5      <script src="https://unpkg.com/vue"></script>
6  </head>
7  <body>
8  <div id="app">
9  <ol>
10     <todo-item></todo-item>
11  </ol>
12 </div>
13 <script>
14  Vue.component('todo-item', {
15      template: '<li>This is a todo</li>'
16  })
17  var app = new Vue({
18      el: '#app'
19  })
20 </script>
21 </body>
22 </html>
```

← → C ⓘ localhost:8000/components1.html

## 1. This is a todo



# Components with props

- We should be able to pass data from the parent scope into child components
- Let's modify the component definition to make it accept a prop:

```
JS
Vue.component('todo-item', {
  // The todo-item component now accepts a
  // "prop", which is like a custom attribute.
  // This prop is called todo.
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Now we can pass the todo into each repeated component using `v-bind` :

```
HTML
<div id="app-7">
  <ol>
    <!--
      Now we provide each todo-item with the todo object
      it's representing, so that its content can be dynamic.
      We also need to provide each component with a "key",
      which will be explained later.
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id">
    </todo-item>
  </ol>
</div>
```

# Results of components with props

```
<div id="app">
  <ol>
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id">
    </todo-item>
  </ol>
</div>
<script>
  Vue.component('todo-item', {
    props: ['todo'],
    template: '<li>{{ todo.text }}</li>'
  })
  var app = new Vue({
    el: '#app',
    data: {
      groceryList: [
        {id: 0, text: 'Vegetables'},
        {id: 1, text: 'Cheese'},
        {id: 2, text: 'Whatever else humans are supposed to eat'}
      ]
    }
  })
</script>
```

← → ⌂ ⓘ localhost:8000/components-props.html

1. Vegetables
2. Cheese
3. Whatever else humans are supposed to eat



# Creating a vue instance

- Every Vue application starts by a new Vue instance with the Vue function

```
JS  
var vm = new Vue({  
    // options  
})
```

- A Vue application consists of a **root Vue instance** created with new Vue, optionally organized into a tree of nested, reusable components.



# Data and methods

- When a Vue instance is created, it adds all the properties found in its data object to Vue's reactivity system
- When the values of those properties change, the view will “react”, updating to match the new values

JS

```
// Our data object
var data = { a: 1 }

// The object is added to a Vue instance
var vm = new Vue({
  data: data
})

// Getting the property on the instance
// returns the one from the original data
vm.a == data.a // => true
```



# Template syntax

- Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data
- All Vue.js templates are valid HTML that can be parsed by spec-compliant browsers and HTML parsers
- Vue is able to intelligently figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes



# #text

- The most basic form of data binding is text interpolation using the “Mustache” syntax (double curly braces)

HTML

```
<span>Message: {{ msg }}</span>
```

- The mustache tag will be replaced with the value of msg property on the corresponding data object
- It will also be updated whenever the data object's msg property changes



# v-once directive

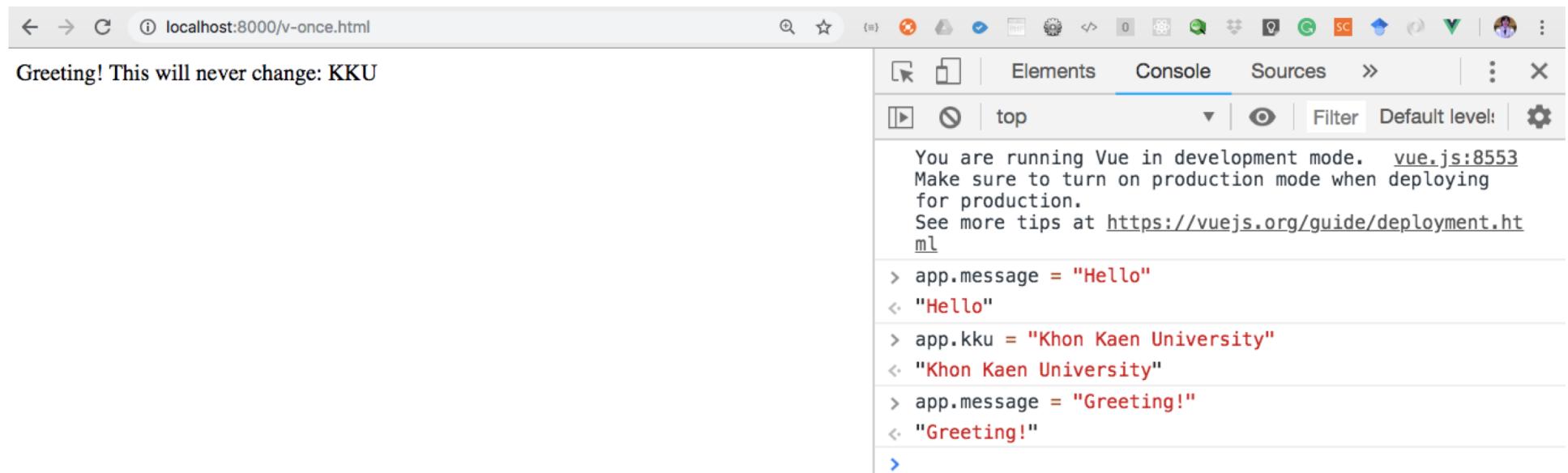
- You can also perform one-time interpolations that do not update on data change by using the **v-once directive**, but keep in mind this will also affect any other bindings on the same node:

HTML

```
<span v-once>This will never change: {{ msg }}</span>
```



# Ex 3: Using v-once directive



The screenshot shows a browser window with the URL `localhost:8000/v-once.html`. The page content is "Greeting! This will never change: KKU". To the right is the developer tools console tab, which displays the following output:

```
You are running Vue in development mode. vue.js:8553
Make sure to turn on production mode when deploying
for production.
See more tips at https://vuejs.org/guide/deployment.html
> app.message = "Hello"
< "Hello"
> app.kku = "Khon Kaen University"
< "Khon Kaen University"
> app.message = "Greeting!"
< "Greeting!"
```



# References

- <https://vuejs.org/v2/guide/#>

