Vue.js

Getting Started

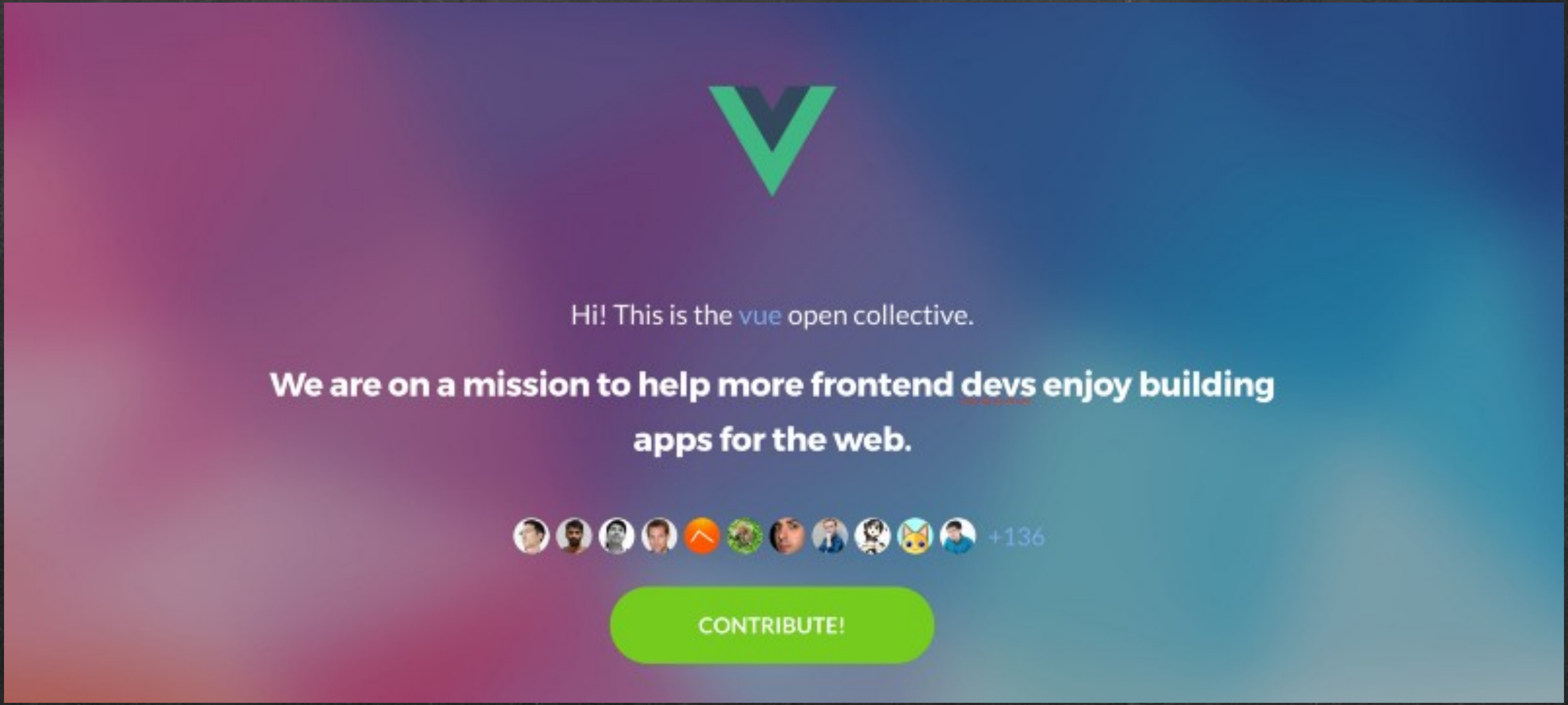
Murat Dogan – Sr. Front End Developer @ Hurriyet

- What is Vue.js ?
- Comparison with other libraries/frameworks
- SFC Concept
- Getting Started with Vue.js
- Basic Vue.js Features
- Creating a development environment in 1 min
- Okur Temsilcisi Vue.js Edition / Vue Router

How to choose the right framework?

- How **mature are the frameworks / libraries**?
- Are the frameworks likely to **be around for a while**?
- How **extensive and helpful are their corresponding communities**?
- How **easy is it to find developers** for each of the frameworks?
- What are the **basic programming concepts** of the frameworks?
- How easy is it to use the frameworks **for small or large applications**?
- What does the **learning curve** look like for each framework?
- What kind of **performance** can you expect from the frameworks?
- Where can you have a **closer look under the hood**?
- How **can you start developing** with the chosen framework?

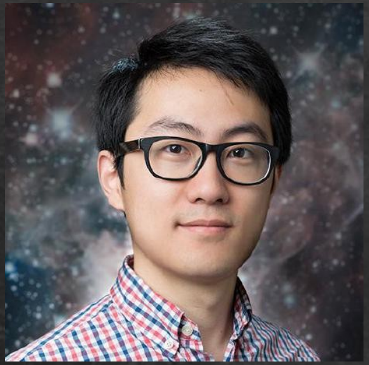
Latest News : 11th September



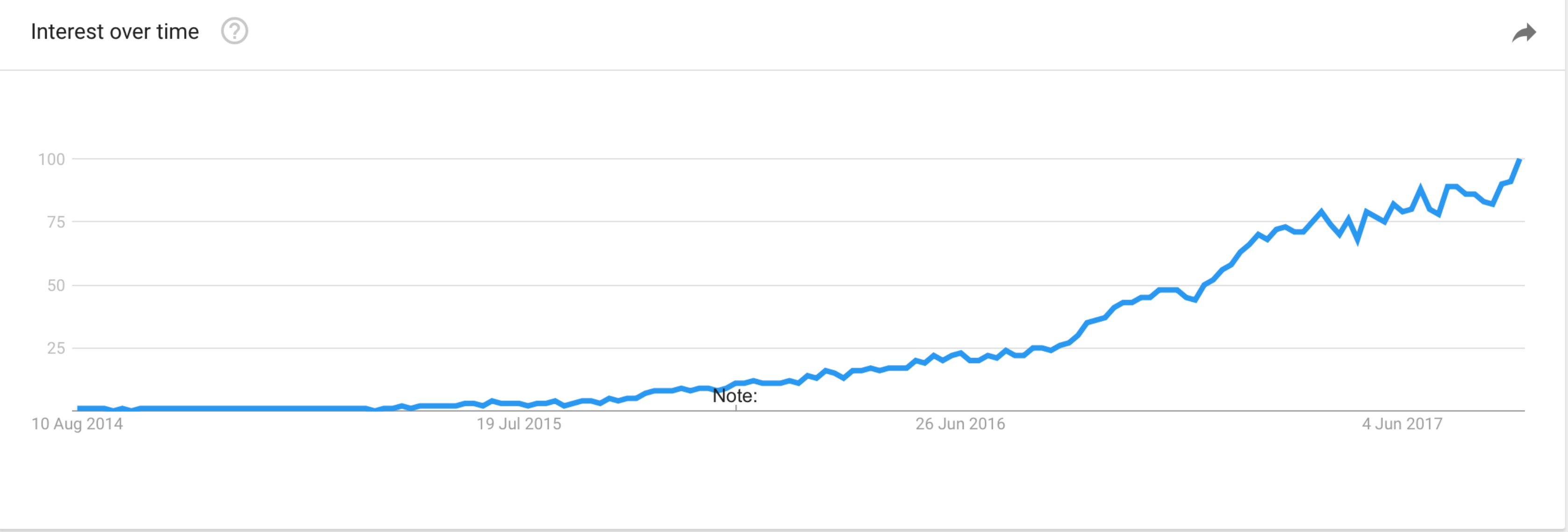
Hi! This is the vue open collective.

We are on a mission to help more frontend devs enjoy building apps for the web.

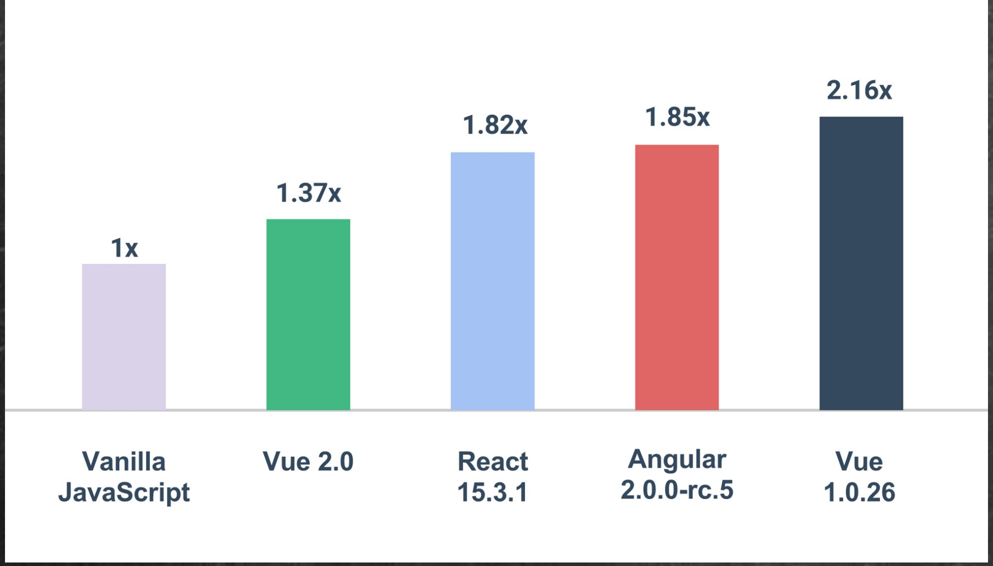
What is Vue.js?



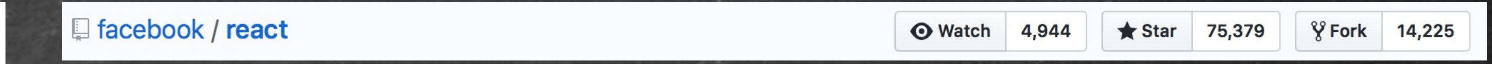
- Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces.[1]
- It was first released in February 2014 by ex-Google-employee Evan You
- Only thing you need to know is HTML, CSS and JavaScript
- It’s been popular since its last version 2.0
- Vue is used by Alibaba, Baidu, Expedia, Nintendo, GitLab — a list of smaller projects can be found on madewithvuejs.com.



Comparison with other libraries/frameworks

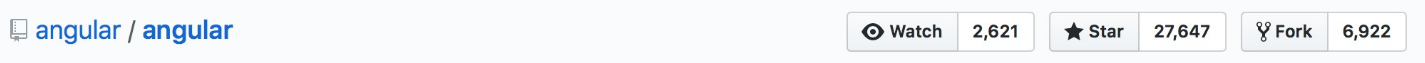


Based on 3rd party benchmark, lower is better



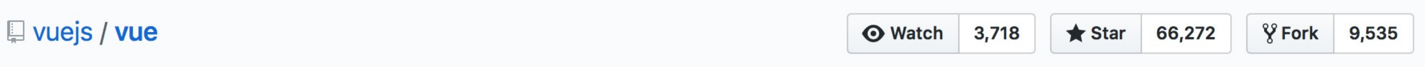
facebook / react

Watch 4,944 | Star 75,379 | Fork 14,225



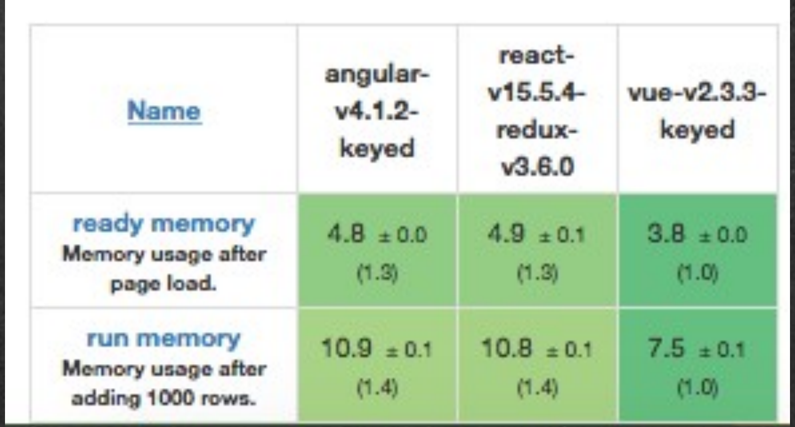
angular / angular

Watch 2,621 | Star 27,647 | Fork 6,922



vuejs / vue

Watch 3,718 | Star 66,272 | Fork 9,535



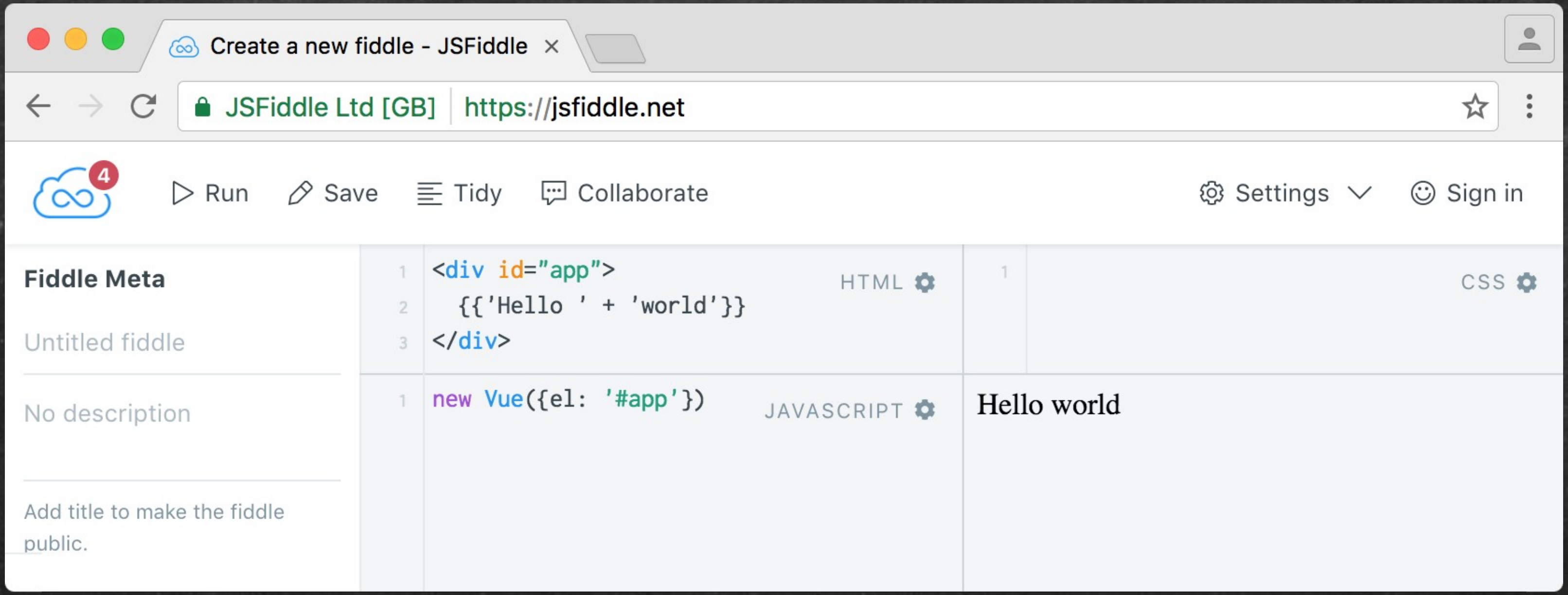
Name	angular-v4.1.2-keyed	react-v15.5.4-redux-v3.6.0	vue-v2.3.3-keyed
ready memory Memory usage after page load.	4.8 ± 0.0 (1.3)	4.9 ± 0.1 (1.3)	3.8 ± 0.0 (1.0)
run memory Memory usage after adding 1000 rows.	10.9 ± 0.1 (1.4)	10.8 ± 0.1 (1.4)	7.5 ± 0.1 (1.0)

Installing Vue.js

- You can install core library via CDN (Content Delivery Network)
- If you want to always follow up the latest version of Vue, use unpkg:
  - https://unpkg.com/vue - Always redirect to the latest version of Vue!
  - https://unpkg.com/vue@[version]/dist/vue.min.js - to specific version.
- You can install core library using Bower or npm

Getting Started – Hello World

- Open https://jsfiddle.net, choose the additional JS Framework; Vue.js
- JavaScript section :
  - new Vue({el:'#app'})
- HTML section:
  - <div id="app"> {{'Hello ' + 'world'}} </div>
- Result:



Fiddle Meta

Untitled fiddle

No description

Add title to make the fiddle public.

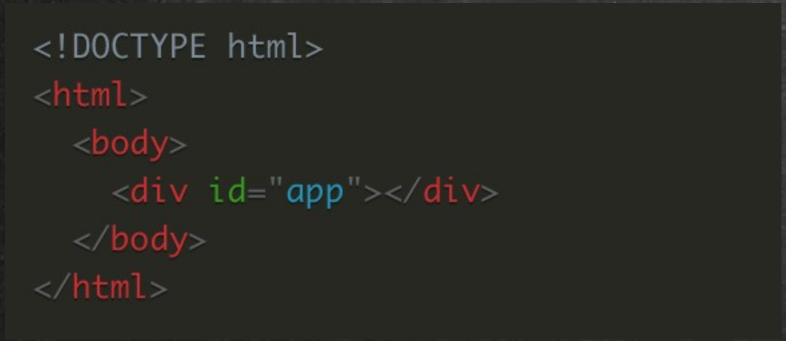
```
<div id="app">
  {{'Hello ' + 'world'}}
</div>
```

```
new Vue({el: '#app'})
```

Hello world

How it works...

new Vue({el:'#app'}) will instantiate a new Vue instance. It accepts an options object as a parameter. This object is central in Vue, and defines and controls data and behavior. It contains all the information needed to create Vue instances and components. In our case, we only specified the el option which accepts a selector or an element as an argument.
The #app parameter is a selector that will return the element in the page with app as the identifier. For example, in a page like this:

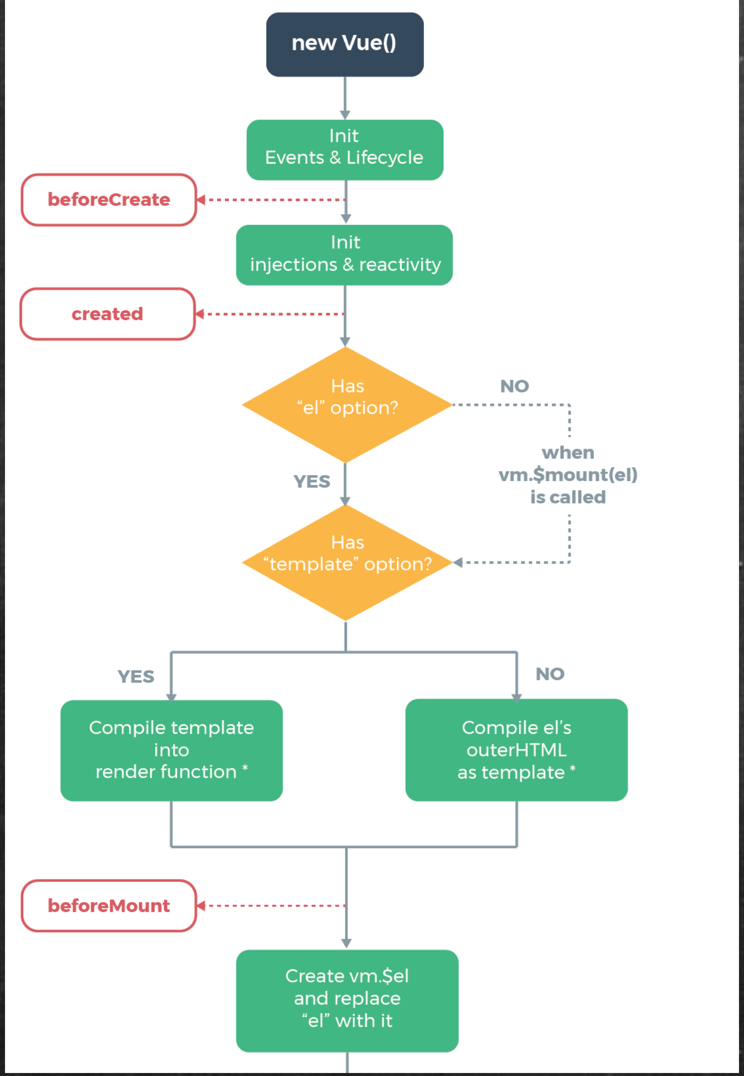


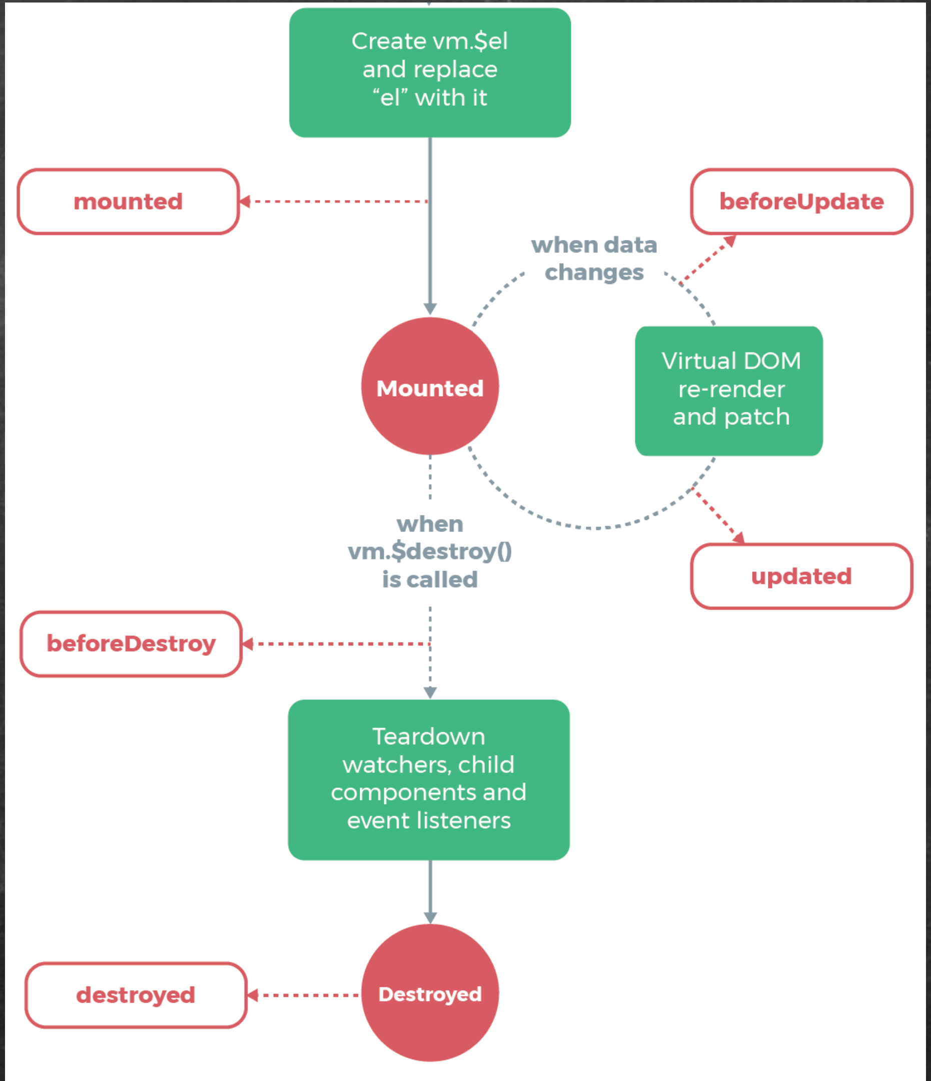
```
<!DOCTYPE html>
<html>
  <body>
    <div id="app"></div>
  </body>
</html>
```

Everything that we will write inside the <div> with the ID as app will be under the scope of Vue.
Now, JSFiddle takes everything we write in the HTML quadrant and wraps it in body tags. This means that if we just need to write the <div> in the HTML quadrant, JSFiddle will take care of wrapping it in the body tags.

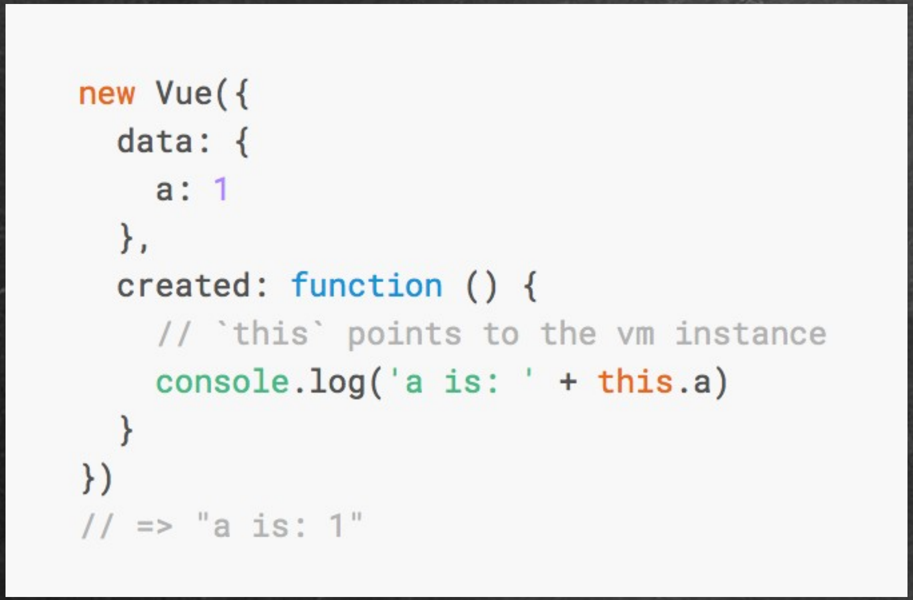
It's also important to note that placing the #app on the *body* or *html* tag will throw an error, as Vue advises us to mount our apps on normal elements, and its the same thing goes for selecting the body in the el option.

Lifecycle Diagram





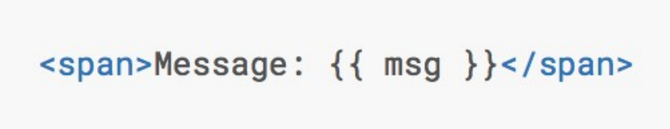
Lifecycle Diagram Example



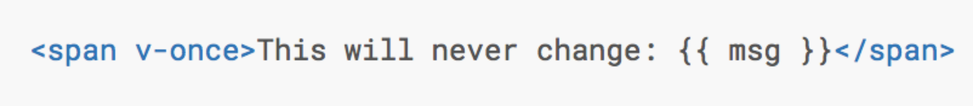
```
new Vue({
  data: {
    a: 1
  },
  created: function () {
    // `this` points to the vm instance
    console.log('a is: ' + this.a)
  }
})
// => "a is: 1"
```

Interpolations

Text



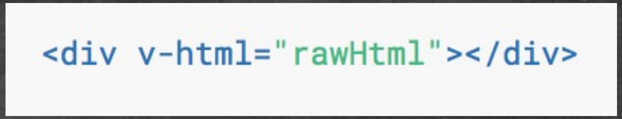
```
<span>Message: {{ msg }}</span>
```



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

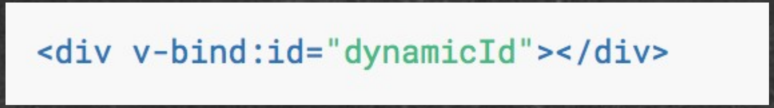
this will also affect any binding on the same node

Raw Html

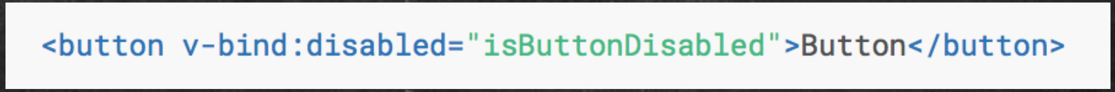


```
<div v-html="rawHtml"></div>
```

Attributes



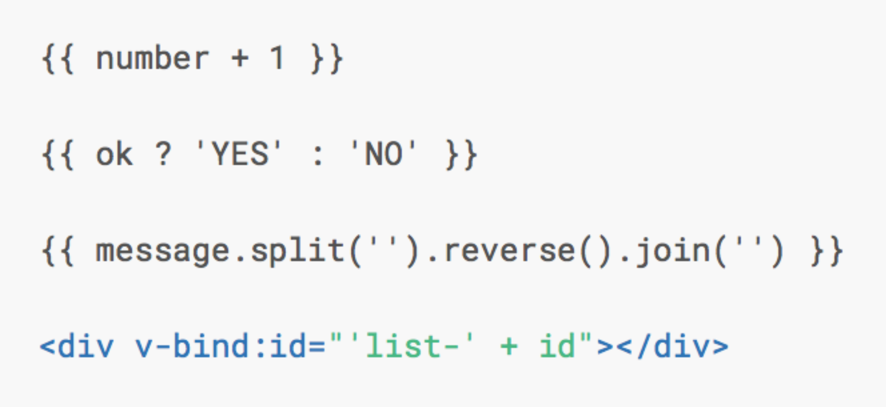
```
<div v-bind:id="dynamicId"></div>
```



```
<button v-bind:disabled="isButtonDisabled">Button</button>
```

Interpolations - JavaScript Expressions

So far we’ve only been binding to simple property keys in our templates. But Vue.js actually supports the full power of JavaScript expressions inside all data bindings

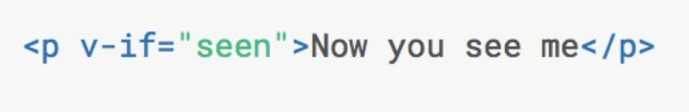


```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

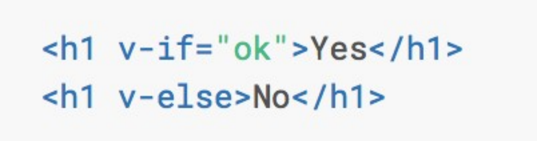
Directives

Directives are special attributes with the v- prefix. Directive attribute values are expected to be **a single JavaScript expression** (with the exception for v-for, which will be discussed later). A directive’s job is to reactively apply side effects to the DOM when the value of its expression changes. Let’s review the example we saw in the introduction:

v-if

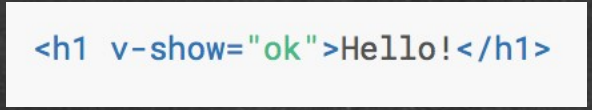


```
<p v-if="seen">Now you see me</p>
```



```
<h1 v-if="ok">Yes</h1>
<h1 v-else>No</h1>
```

v-show



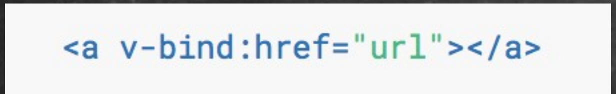
```
<h1 v-show="ok">Hello!</h1>
```

v-if vs v-show

- v-if is “real” conditional rendering because it ensures that event listeners and child components inside the conditional block are properly destroyed and re-created during toggles.
- v-if is also **lazy**: if the condition is false on initial render, it will not do anything - the conditional block won’t be rendered until the condition becomes true for the first time.
- In comparison, v-show is much simpler - the element is always rendered regardless of initial condition, with just simple CSS-based toggling.
- Generally speaking, v-if has higher toggle costs while v-show has higher initial render costs. So prefer v-show if you need to toggle something very often, and prefer v-if if the condition is unlikely to change at runtime.

Arguments

Some directives can take an “argument”, denoted by a colon after the directive name. For example, the v-bind directive is used to reactively update an HTML attribute:

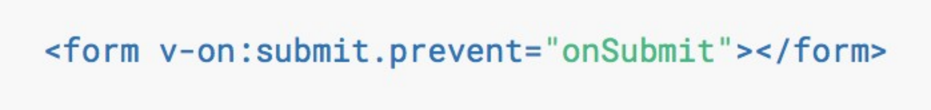


```
<a v-bind:href="url"></a>
```

Here ***href*** is the argument, which tells the ***v-bind*** directive to bind the element’s ***href attribute*** to the value of the expression ***url***.

Modifiers

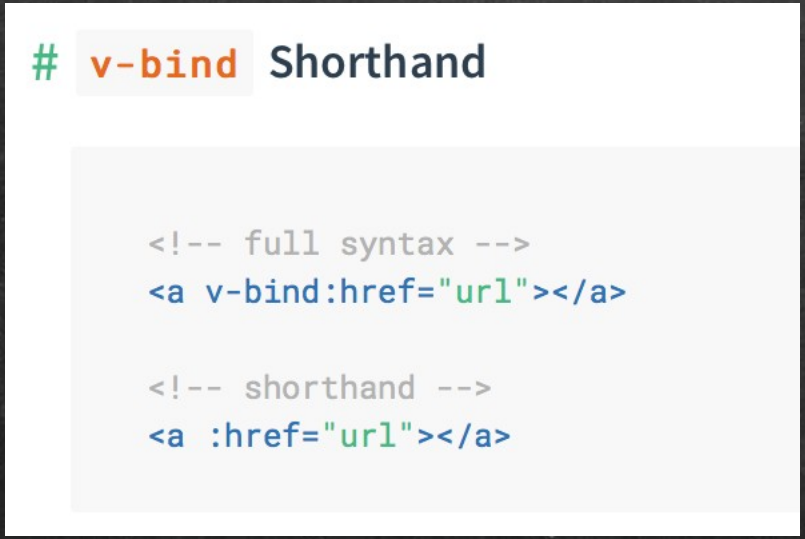
Modifiers are special postfixes denoted by a dot, which indicate that a directive should be bound in some special way. For example, the .prevent modifier tells the v-on directive to call event.preventDefault() on the triggered event:



```
<form v-on:submit.prevent="onSubmit"></form>
```

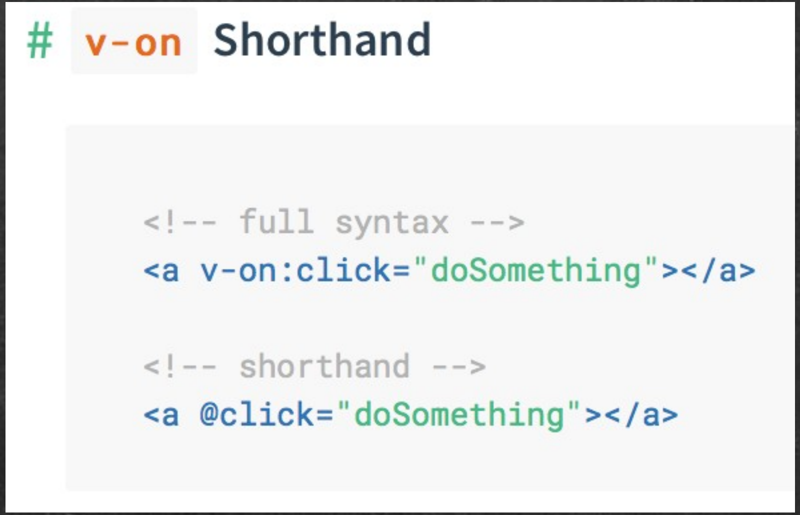
Shorthands

The v- prefix serves as a visual cue for identifying Vue-specific attributes in your templates. This is useful when you are using Vue.js to apply dynamic behavior to some existing markup, but can feel verbose for some frequently used directives. At the same time, the need for the v- prefix becomes less important when you are building an SPA where Vue.js manages every template. Therefore, Vue.js provides special shorthands for two of the most often used directives, v-bind and v-on:



v-bind Shorthand

```
<!-- full syntax -->
<a v-bind:href="url"></a>

<!-- shorthand -->
<a :href="url"></a>
```

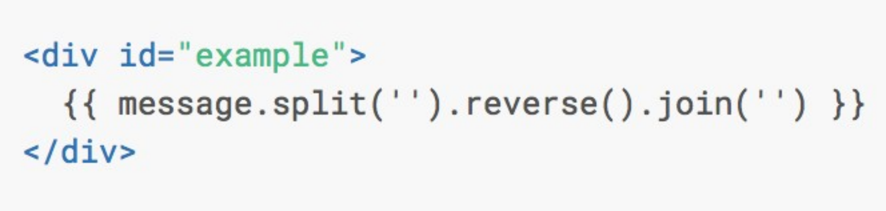


v-on Shorthand

```
<!-- full syntax -->
<a v-on:click="doSomething"></a>

<!-- shorthand -->
<a @click="doSomething"></a>
```

Computed properties

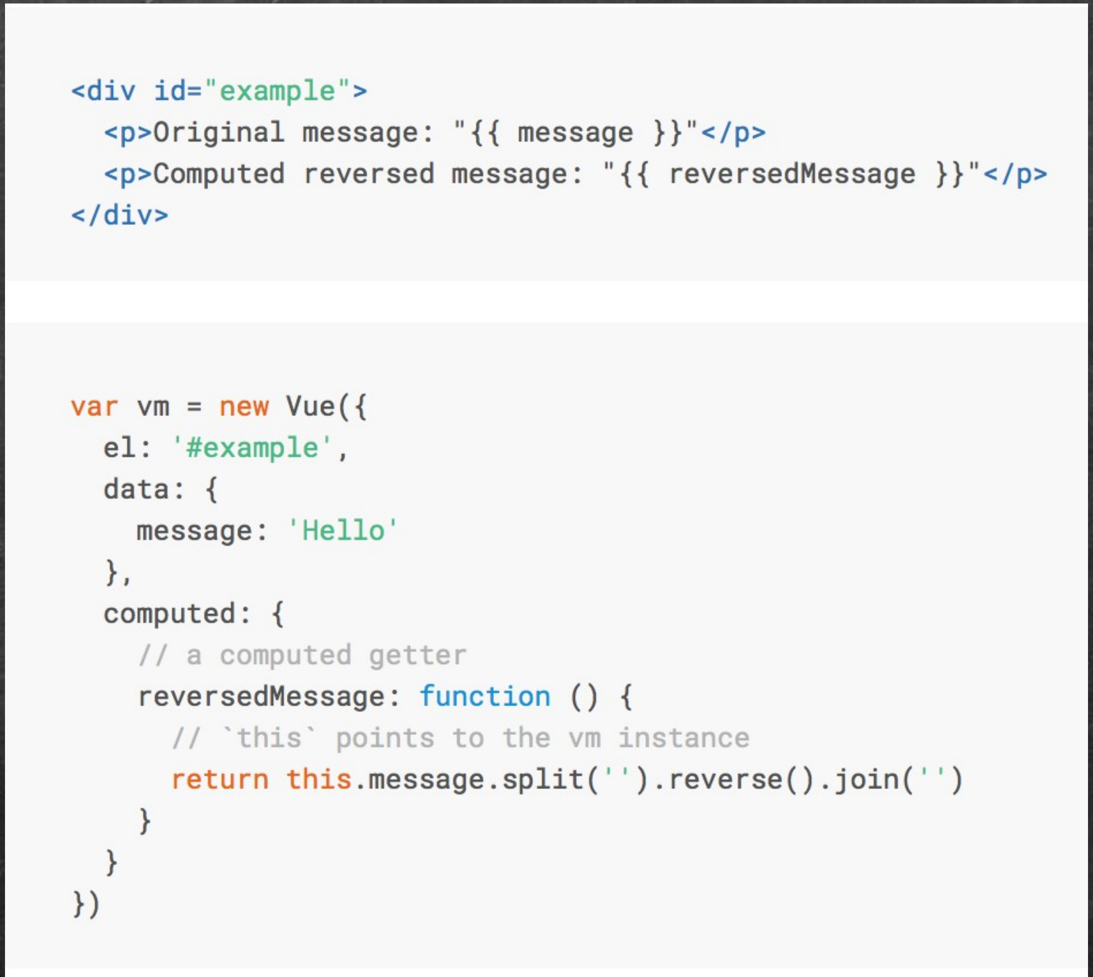
In-template expressions are very convenient, but they are really only meant for simple operations. Putting too much logic into your templates can make them bloated and hard to maintain. For example:



```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

At this point, the template is no longer simple and declarative. That’s why for any complex logic, you should use a **computed property**.

Computed properties

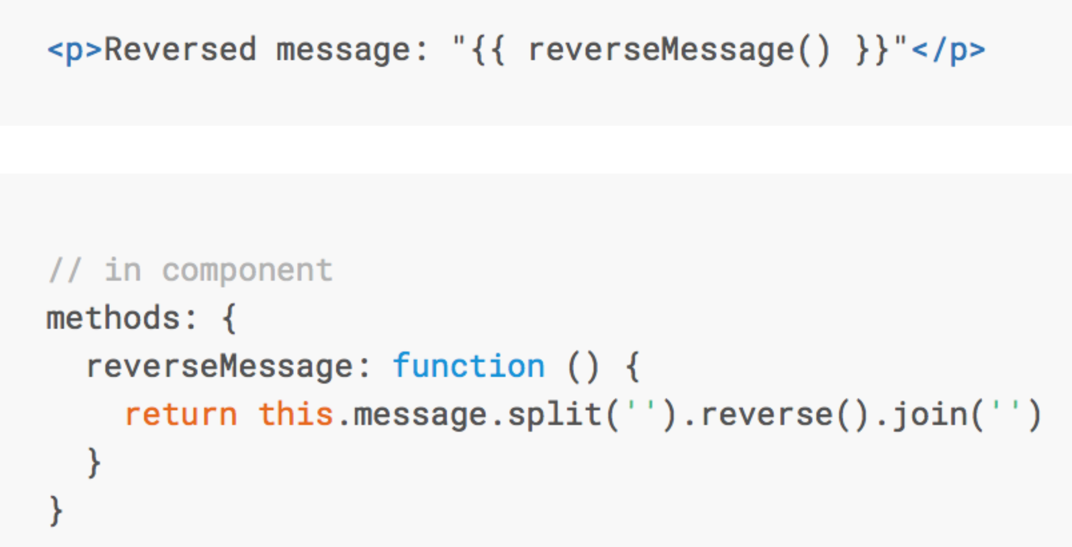


```html
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```javascript
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

Computed caching vs Methods

You may have noticed we can achieve the same result by invoking a method in the expression:



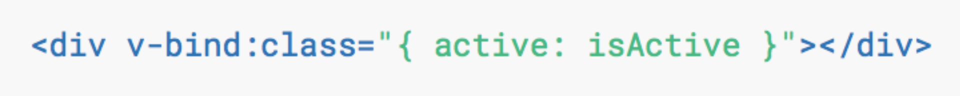
```
<p>Reversed message: "{{ reverseMessage() }}"</p>
```

```
// in component
methods: {
  reverseMessage: function () {
    return this.message.split('').reverse().join('')
  }
}
```

However, the difference is that **computed properties are cached based on their dependencies.** A computed property will only re-evaluate when some of its dependencies have changed. This means as long as messagehas not changed, multiple access to the reversedMessage computed property will immediately return the previously computed result without having to run the function again.

Binding HTML Classes

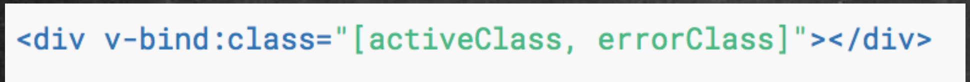
Object Syntax



```
<div v-bind:class="{ active: isActive }"></div>
```

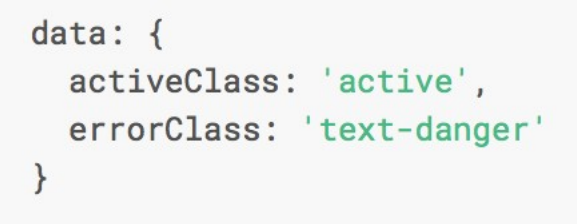
The above syntax means the presence of the active class will be determined by the **truthiness** of the data property isActive.

Array Syntax



```
<div v-bind:class="[activeClass, errorClass]"></div>
```

The above syntax means the presence of the active class will be determined by the **truthiness** of the data property isActive.

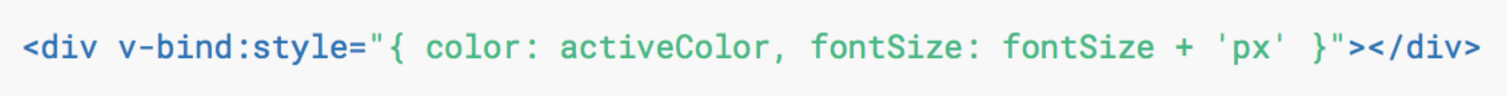


```
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

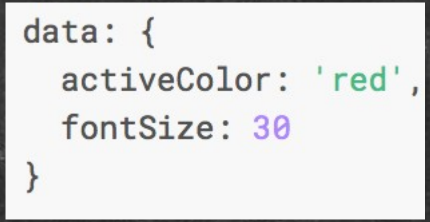
Binding Inline Styles

Object Syntax

The object syntax for v-bind:style is pretty straightforward - it looks almost like CSS, except it’s a JavaScript object. You can use either camelCase or kebab-case (use quotes with kebab-case) for the CSS property names:



```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

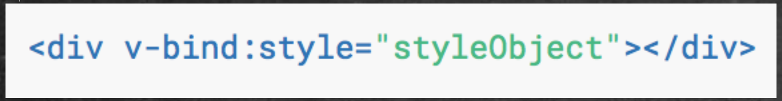


```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

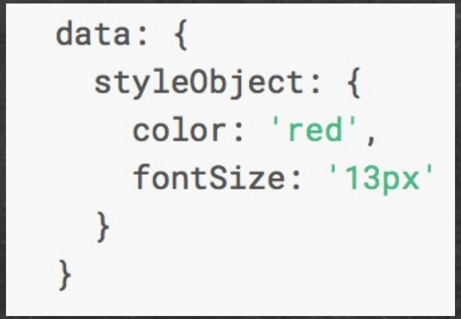
Binding Inline Styles

Object Syntax

It is often a good idea to bind to a style object directly so that the template is cleaner:



```
<div v-bind:style="styleObject"></div>
```



```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

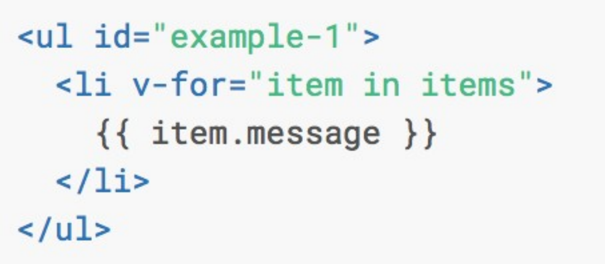
Binding Inline Styles

Auto-prefixing

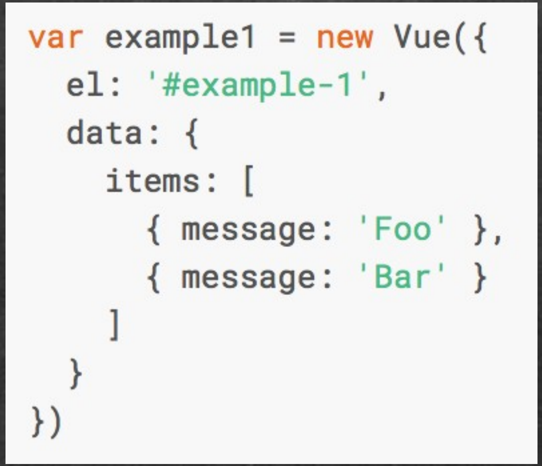
When you use a CSS property that requires vendor prefixes in v-bind:style, for example transform, Vue will automatically detect and add appropriate prefixes to the applied styles.

List Rendering

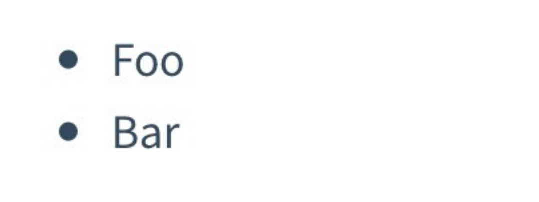
We can use the v-for directive to render a list of items based on an array. The v-fordirective requires a special syntax in the form of item in items, where items is the source data array and item is an **alias** for the array element being iterated on:



```html
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```



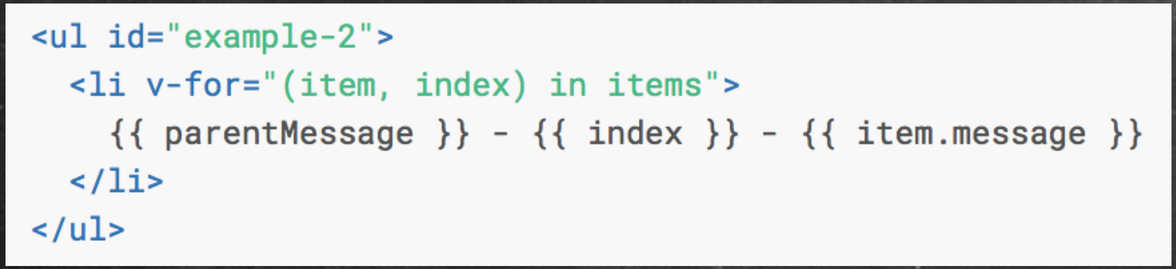
```javascript
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```



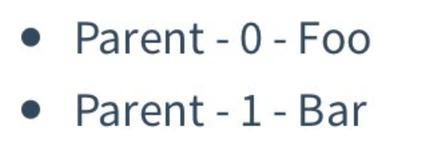
- Foo
- Bar

List Rendering

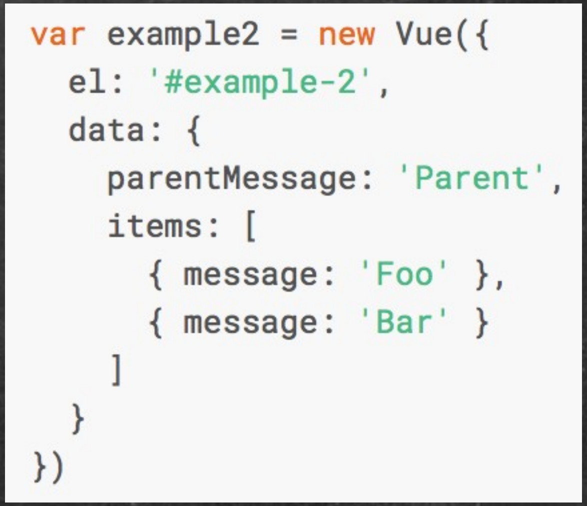
Inside v-for blocks we have full access to parent scope properties. v-for also supports an optional second argument for the index of the current item.



```html
<ul id="example-2">
  <li v-for="(item, index) in items">
    {{ parentMessage }} - {{ index }} - {{ item.message }}
  </li>
</ul>
```



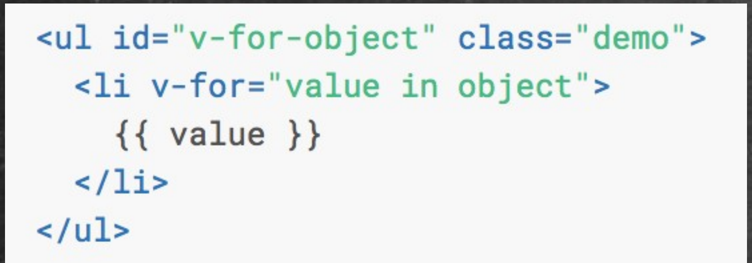
- Parent - 0 - Foo
- Parent - 1 - Bar



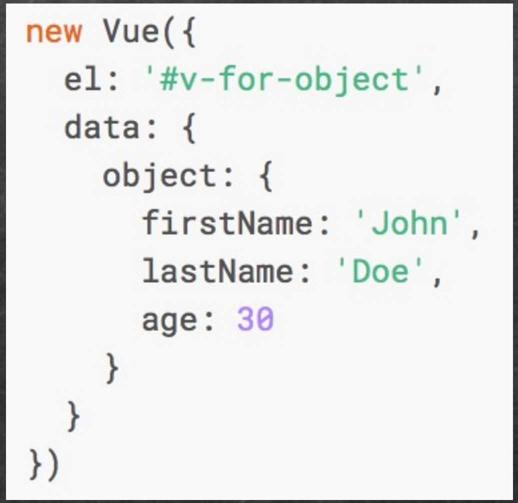
```js
var example2 = new Vue({
  el: '#example-2',
  data: {
    parentMessage: 'Parent',
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

v-for with an Object

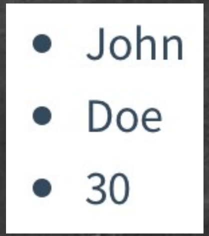
You can also use v-for to iterate through the properties of an object.



```
<ul id="v-for-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```



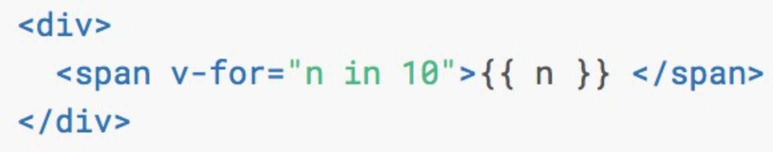
```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      firstName: 'John',
      lastName: 'Doe',
      age: 30
    }
  }
})
```



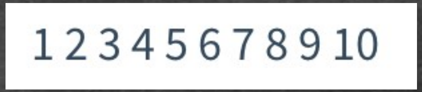
- John
- Doe
- 30

v-for with a Range

v-for can also take an integer. In this case it will repeat the template that many times.



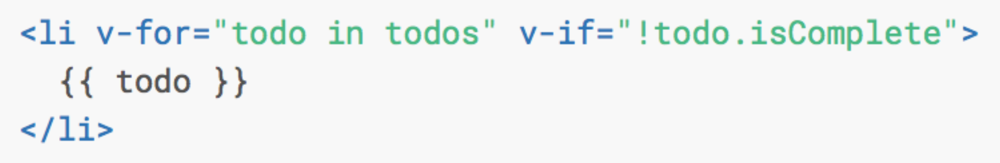
```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```



```
1 2 3 4 5 6 7 8 9 10
```

v-for with v-if

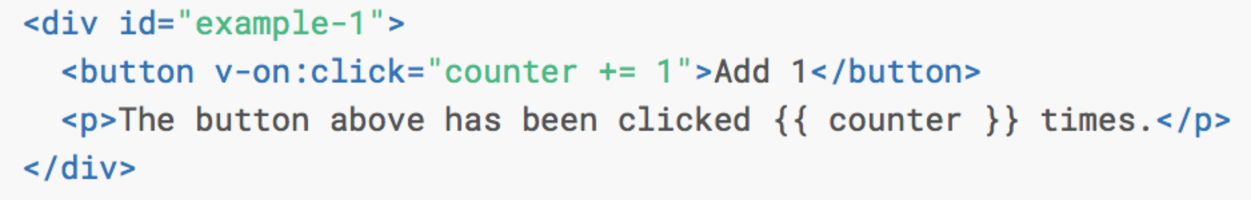
When they exist on the same node, v-for has a higher priority than v-if. That means the v-if will be run on each iteration of the loop separately. This can be useful when you want to render nodes for only *some* items, like below:



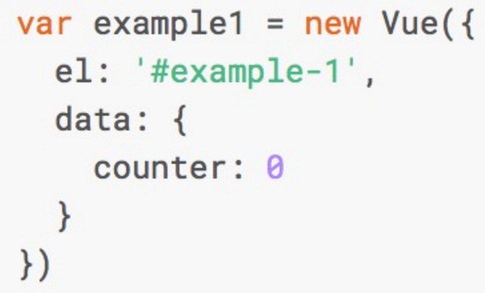
```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

Event Handling

We can use the v-on directive to listen to DOM events and run some JavaScript when they’re triggered.
For example:



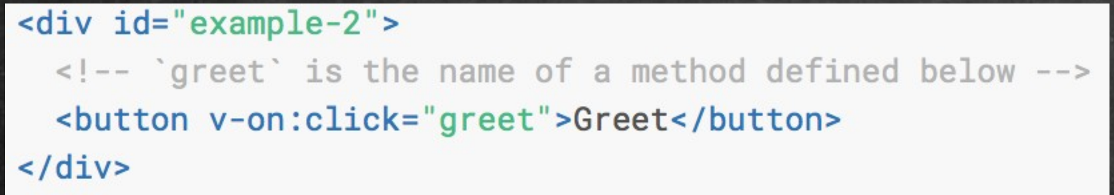
```html
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```



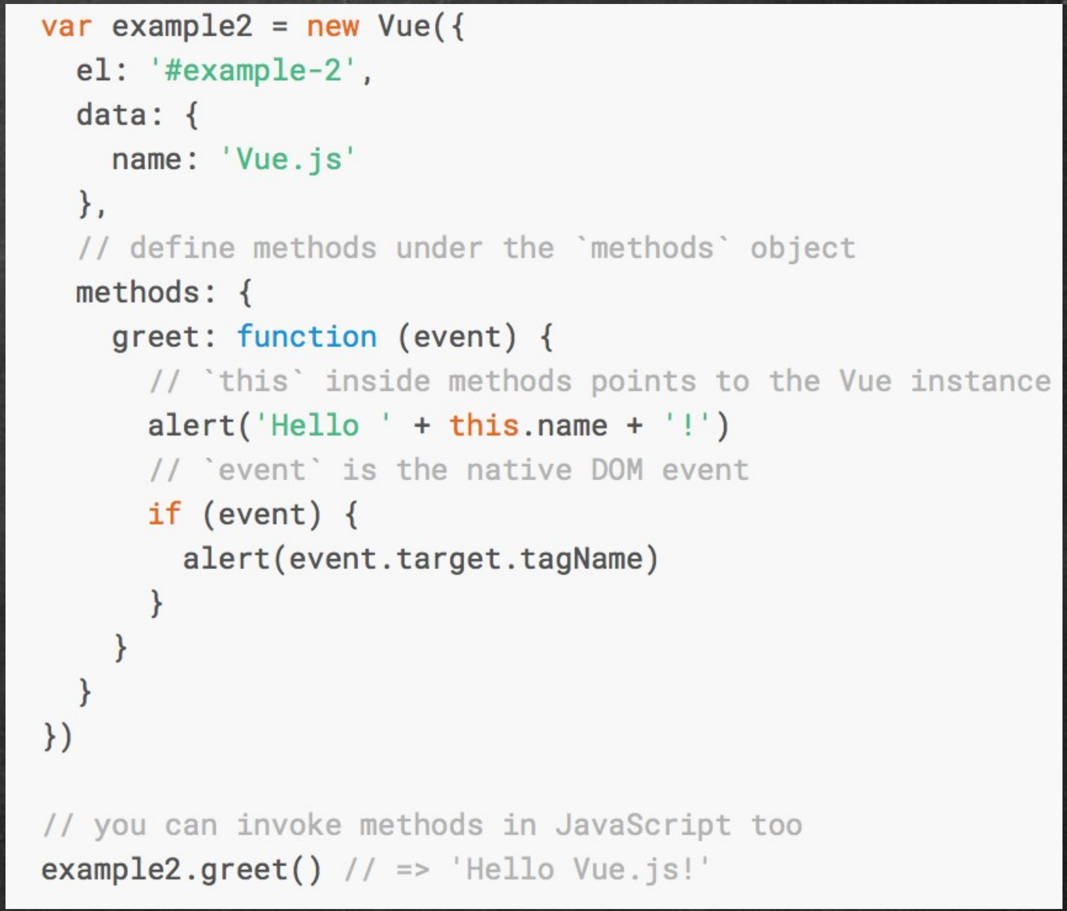
```javascript
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

Method Event Handlers

The logic for many event handlers will be more complex though, so keeping your JavaScript in the value of the v-on attribute simply isn’t feasible. That’s why v-on can also accept the name of a method you’d like to call.



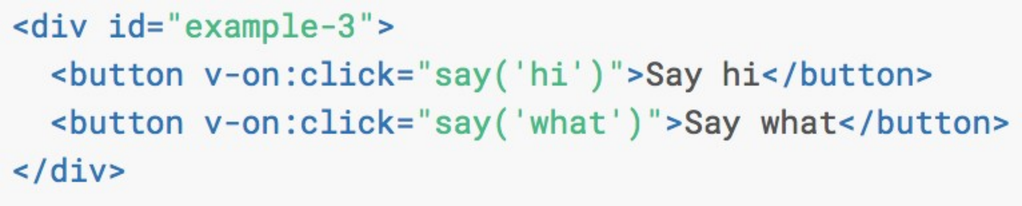
```html
<div id="example-2">
  <!-- `greet` is the name of a method defined below -->
  <button v-on:click="greet">Greet</button>
</div>
```



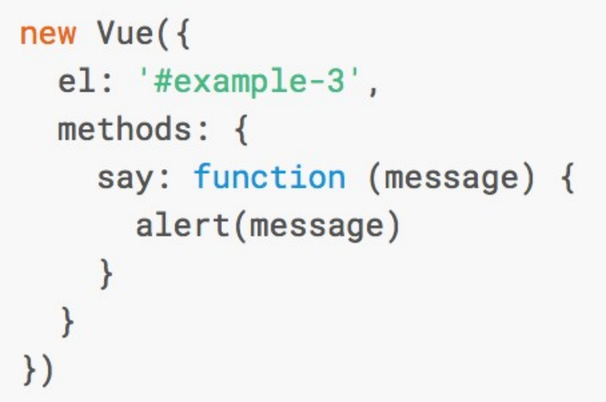
```javascript
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` inside methods points to the Vue instance
      alert('Hello ' + this.name + '!')
      // `event` is the native DOM event
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
})

// you can invoke methods in JavaScript too
example2.greet() // => 'Hello Vue.js!'
```

Methods in Inline Handlers

Instead of binding directly to a method name, we can also use methods in an inline JavaScript statement:



```
<div id="example-3">
  <button v-on:click="say('hi')">Say hi</button>
  <button v-on:click="say('what')">Say what</button>
</div>
```



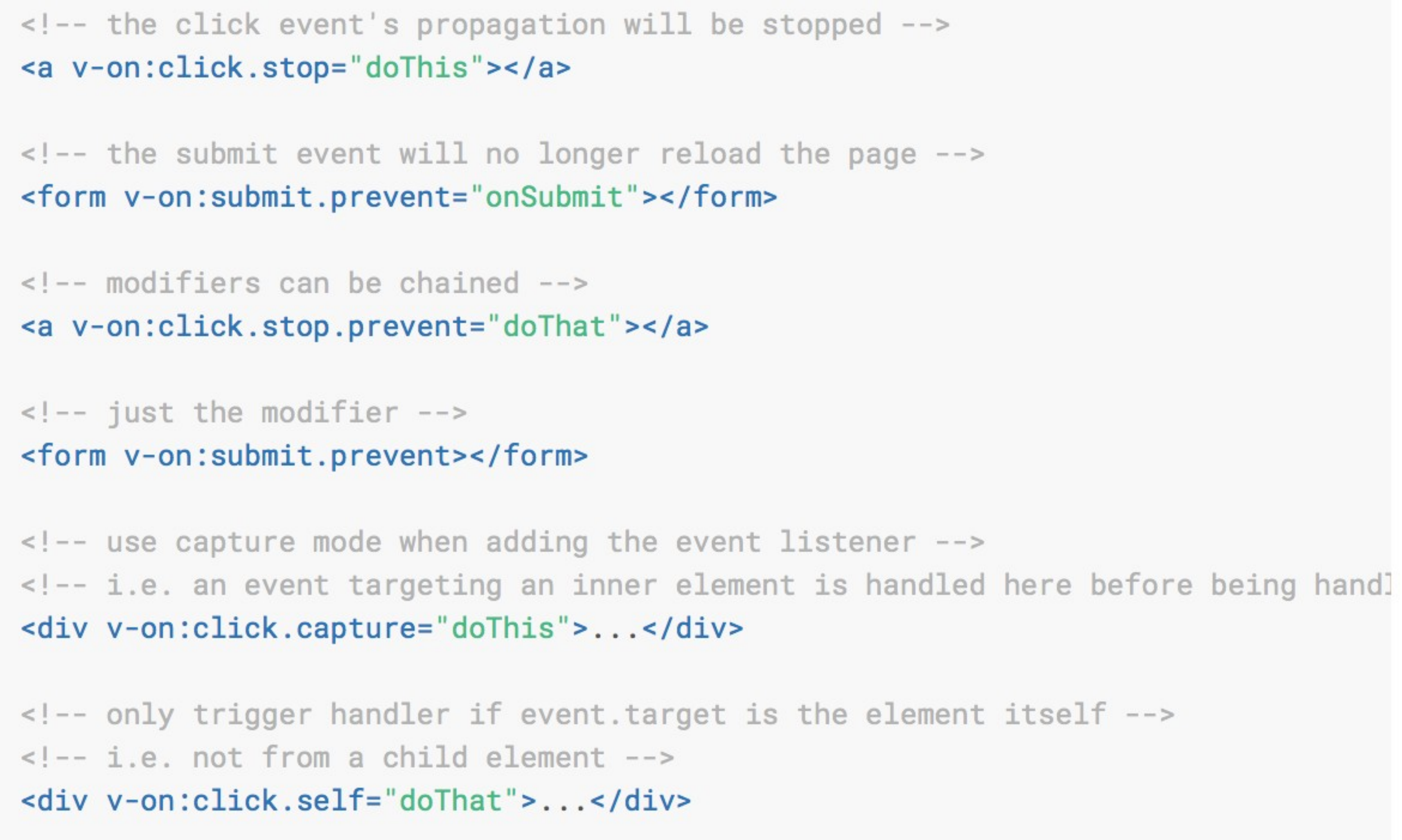
```
new Vue({
  el: '#example-3',
  methods: {
    say: function (message) {
      alert(message)
    }
  }
})
```

Event Modifiers

It is a very common need to call event.preventDefault() or event.stopPropagation()inside event handlers. Although we can do this easily inside methods, it would be better if the methods can be purely about data logic rather than having to deal with DOM event details.
To address this problem, Vue provides **event modifiers** for v-on. Recall that modifiers are directive postfixes denoted by a dot.

- **.stop**
- **.prevent**
- **.capture**
- **.self**
- **.once**

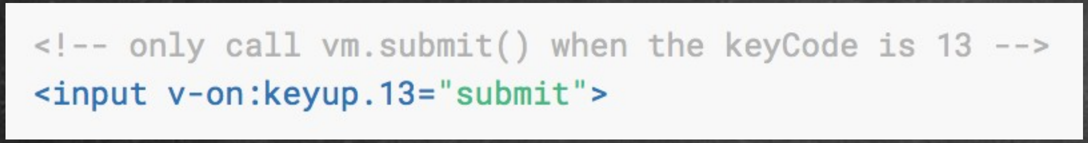
Event Modifiers



```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form v-on:submit.prevent></form>

<!-- use capture mode when adding the event listener -->
<!-- i.e. an event targeting an inner element is handled here before being handl
<div v-on:click.capture="doThis">...</div>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>
```

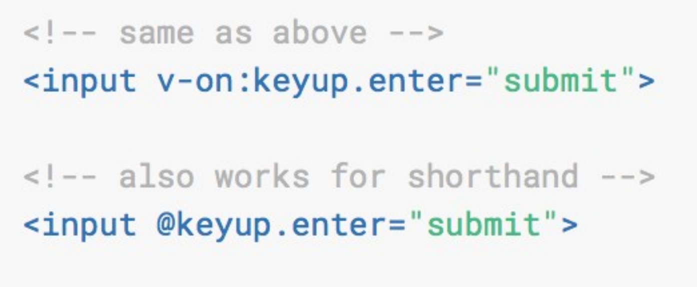
Key Modifiers

When listening for keyboard events, we often need to check for common key codes. Vue also allows adding key modifiers for v-on when listening for key events:



```
<!-- only call vm.submit() when the keyCode is 13 -->
<input v-on:keyup.13="submit">
```

Remembering all the keyCodes is a hassle, so Vue provides aliases for the most commonly used keys:



```
<!-- same as above -->
<input v-on:keyup.enter="submit">

<!-- also works for shorthand -->
<input @keyup.enter="submit">
```

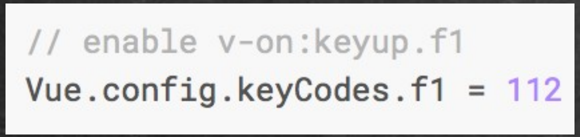
Key Modifiers

Here’s the full list of key modifier aliases:

- .enter
- .tab
- .delete (captures both “Delete” and “Backspace” keys)
- .esc
- .space
- .up
- .down
- .left
- .right

Key Modifiers

You can also **define custom key modifier aliases** via the global config.keyCodes object:



```
// enable v-on:keyup.f1
Vue.config.keyCodes.f1 = 112
```

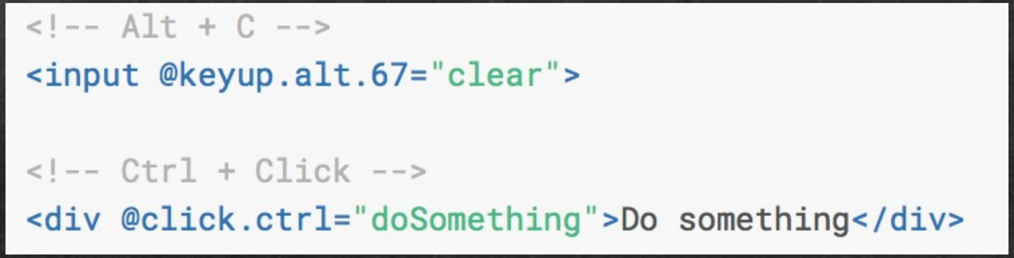
```
<input v-on:keyup.f1="submit">
```

Modifier Keys

You can use the following modifiers to trigger mouse or keyboard event listeners only when the corresponding modifier key is pressed:

- .ctrl
- .alt
- .shift
- .meta

Note: On Macintosh keyboards, meta is the command key (⌘). On Windows keyboards, meta is the windows key (⊞).



```
<!-- Alt + C -->
<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

Mouse Button Modifiers

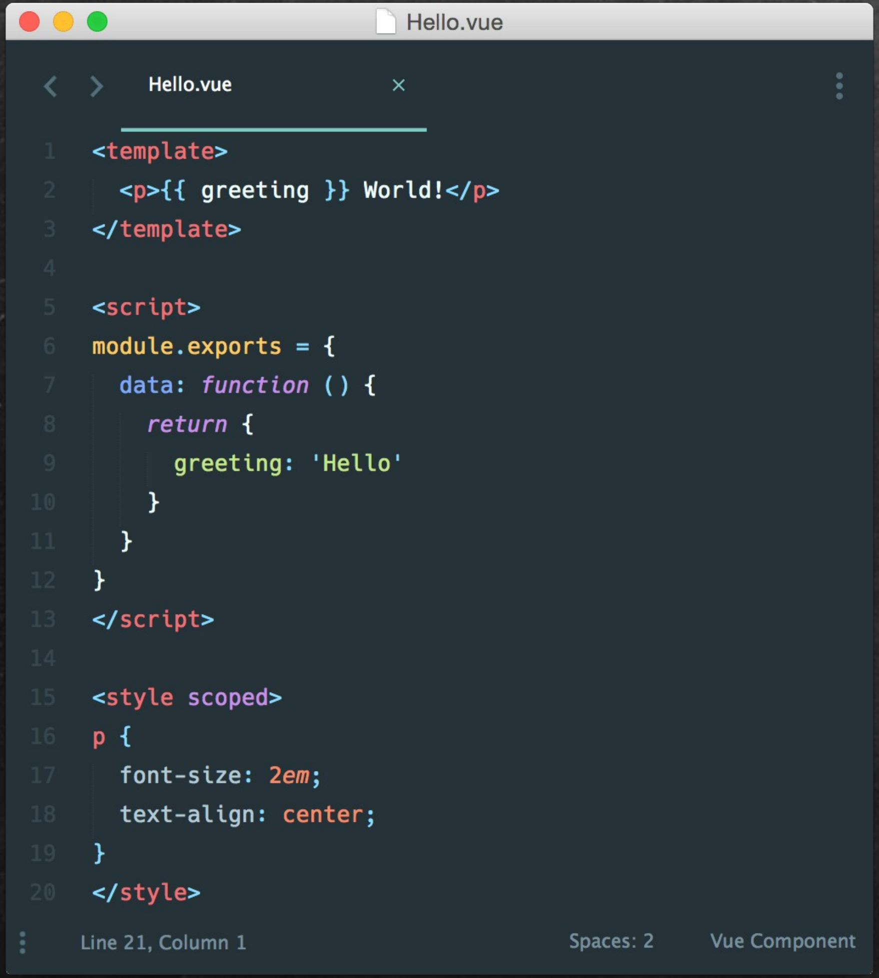
New in 2.2.0+

- .left
- .right
- .middle

These modifiers restrict the handler to events triggered by a specific mouse button.

https://jsfiddle.net/z1jhpewo/1/

SFC Concept (Single File Components)



```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Line 21, Column 1 Spaces: 2 Vue Component

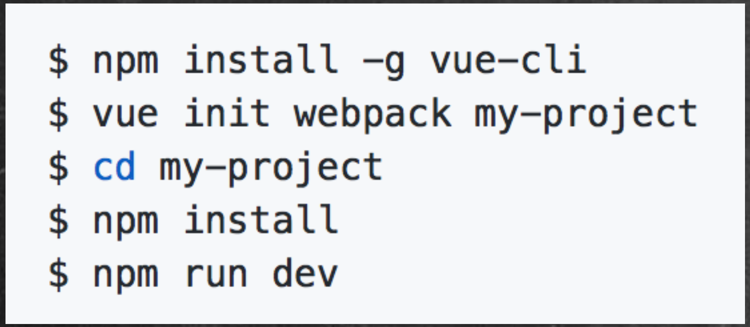
- **Complete syntax highlighting**
- **CommonJS modules**
- **Component-scoped CSS**
- **You can specify language attributes to write more specific code.**

What About Separation of Concerns?

One important thing to note is that **separation of concerns is not equal to separation of file types**

Development Environment in 1 min

vue-webpack-boilerplate is the best practice to start from scratch.



```
$ npm install -g vue-cli
$ vue init webpack my-project
$ cd my-project
$ npm install
$ npm run dev
```

Credits

- https://vuejs.org/v2/guide/
- https://vuejs.org/v2/guide/single-file-components.html
- https://www.packtpub.com/mapt/book/web_development/9781786468093
- https://app.pluralsight.com/library/courses/vuejs-getting-started/table-of-contents
- https://medium.com/the-vue-point/vue-2-0-is-here-ef1f26acf4b8
- https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176
- https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176