The Majesty of Vue.js

Alex Kyriakidis

Kostas Maniatis

The Majesty of Vue.js

Alex Kyriakidis, Kostas Maniatis and Evan You

This book is for sale at http://leanpub.com/vuejs

This version was published on 2016-10-12

Tweet This Book!

Please help Alex Kyriakidis, Kostas Maniatis and Evan You by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm learning @vuejs with @tmvuejs. Get it at https://leanpub.com/vuejs #vuejs

The suggested hashtag for this book is #vuejs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#vuejs

Contents

Introduction

About Vue.js

Vue.js Overview

Vue.js (pronounced /vju:/, like view) is a library for building interactive web interfaces. The goal of Vue.js is to provide the benefits of **reactive data binding** and **composable view components** with an API that is as simple as possible.

Vue.js itself is not a full-blown framework - it is focused on the view layer only. It is therefore very easy to pick up and to integrate with other libraries or existing projects. On the other hand, when used in combination with proper tooling and supporting libraries, Vue.js is also perfectly capable of powering sophisticated Single-Page Applications.

If you are an experienced frontend developer and you want to know how Vue.js compares to other libraries/frameworks, check out the Comparison with Other Frameworks chapter.

If you are interested to learn more information about Vue.js' core take a look at Vue.js official guide[1].

What people say about Vue.js

"Vue.js is what made me love JavaScript. It's extremely easy and enjoyable to use. It has a great ecosystem of plugins and tools that extend its basic services. You can quickly include it in any project, small or big, write a few lines of code and you are set. Vue.js is fast, lightweight and is the future of Front end development!"

—Alex Kyriakidis

"When I started picking up Javascript I got excited learning a ton of possibilities, but when my friend suggested to learn Vue.js and I followed his advice, things went wild. While reading and watching tutorials I kept thinking all the stuff I've done so far and how much easier it would be if I had invest time to learn Vue earlier. My opinion is that if you want to do your work fast, nice and easy Vue is the JS Framework you need. "

—Kostas Maniatis

---

[1] http://vuejs.org/guide/overview.html

"Mark my words: Vue.js will sky-rocket in popularity in 2016. It's that good."

— Jeffrey Way

"Vue is what I always wanted in a JavaScript framework. It's a framework that scales with you as a developer. You can sprinkle it onto one page, or build an advanced single page application with Vuex and Vue Router. It's truly the most polished JavaScript framework I've ever seen."

— Taylor Otwell

"Vue.js is the first framework I've found that feels just as natural to use in a server-rendered app as it does in a full-blown SPA. Whether I just need a small widget on a single page or I'm building a complex Javascript client, it never feels like not enough or like overkill."

— Adam Wathan

"Vue.js has been able to make a framework that is both simple to use and easy to understand. It's a breath of fresh air in a world where others are fighting to see who can make the most complex framework."

— Eric Barnes

"The reason I like Vue.js is because I'm a hybrid designer/developer. I've looked at React, Angular and a few others but the learning curve and terminology has always put me off. Vue.js is the first JS framework I understand. Also, not only is it easy to pick up for the less confidence JS'ers, such as myself, but I've noticed experienced Angular and React developers take note, and liking, Vue.js. This is pretty unprecedented in JS world and it's that reason I started London Vue.js Meetup."

—Jack Barham

Welcome

About the Book

This book will guide you through the path of the rapidly spreading Javascript Framework called Vue.js!

Some time ago, we started a new project based on Laravel and Vue.js. After thoroughly reading Vue.js guide and a few tutorials, we discovered lack of resources about Vue.js around the web. During the development of our project, we gained a lot of experience, so we came up with the idea to write this book in order to share our acquired knowledge with the world.

This book is written in an informal, intuitive, and easy-to-follow format, wherein all examples are appropriately detailed enough to provide adequate guidance to everyone.

We'll start from the very basics and through many examples we'll cover the most significant features of Vue.js. By the end of this book you will be able to create fast front end applications and increase the performance of your existing projects with Vue.js integration.

Who is this Book for

Everyone who has spent time to learn modern web development has seen Bootstrap, Javascript, and many Javascript frameworks. This book is for anyone interested in learning a lightweight and simple Javascript framework. No excessive knowledge is required, though it would be good to be familiar with HTML and Javascript. If you dont't know what the difference is between a string and an object, maybe you need to do some digging first.

This book is also useful for any reader who already know their way around Vue.js and want to expand their knowledge.

Get In Touch

In case you would like to contact us about the book, send us feedback, or other matters you would like to bring to our attention, don't hesitate to contact us.

Name	Email	Twitter
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Alex Kyriakidis	alex@tmvuejs.com	@hootlex
Kostas Maniatis	kostas@tmvuejs.com	@kostaskafcas

Homework

The best way to learn code is to write code, so we have prepared one exercise at the end of most chapters for you to solve and actually test yourself on what you have learned. We strongly recommend you to try as much as possible to solve them and through them gain a better understanding of Vue.js. Don't be afraid to test your ideas, a little effort goes a long way! Maybe a few different examples or ways will give you the right idea. Of course we are not merciless, hints and potential solutions will be provided!

You may begin your journey!

Sample Code

You can find most of the code examples used in the book on GitHub. You can browse around the code here[2].

If you prefer to download it, you will find a `.zip` file here[3].

This will save you from copying and pasting things out of the book, which would probably be terrible.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in the book we would be grateful if you could report it to us. By doing so, you can protect other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please submit an issue on our GitHub repository[4].

Conventions

The following notational conventions are used throughout the book.

A block of code is set as follows:

JavaScript

[2]https://github.com/hootlex/the-majesty-of-vuejs
[3]https://github.com/hootlex/the-majesty-of-vuejs/archive/v1.0.1.zip
[4]https://github.com/hootlex/the-majesty-of-vuejs

```
function(x, y){
    // this is a comment
}
```

Code words in text, data are shown as follows: “Use `.container` for a responsive fixed width container.”

New terms and **important words** are shown in bold.

Tips, notes, and warnings are shown as follows:

This is a Warning

This element indicates a warning or caution.

This is a Tip

This element signifies a tip or suggestion.

This is an Information box

Some special information here.

This is a Note

A note about the subject.

This is a Hint

A hint about the subject.

This is a Terminal Command

Commands to run in terminal.

This is a Comparison text

A small text comparing things relative to the subject.

This is a link to Github for the code examples

Links which lead to the repository of this book, where you can find the code samples of each chapter.

Vue.js Fundamentals

1. Interactivity

In this chapter, we are going to create and expand previous examples, learn new things concerning 'methods', 'event handling' and 'computed properties'. We will develop a few examples using different approaches. It's time to see how we can implement Vue's interactivity to get a small app, like a Calculator, running nice and easy.

1.1 Event Handling

HTML events are things that happen to HTML elements. When Vue.js is used in HTML pages, it can **react** to these events.

In HTML, events can represent everything from basic user interactions to things happening in the rendering model.

These are some examples of HTML events:

- A web page has finished loading
- An input field was changed
- A button was clicked
- A form was submitted

The point of event handling is that you can do something whenever an event takes place.

In Vue.js, to **listen** to DOM events you can use the `v-on` directive.

The `v-on` directive attaches an event listener to an element. The type of the event is denoted by the argument, for example `v-on:keyup` listens to the `keyup` event.

Info

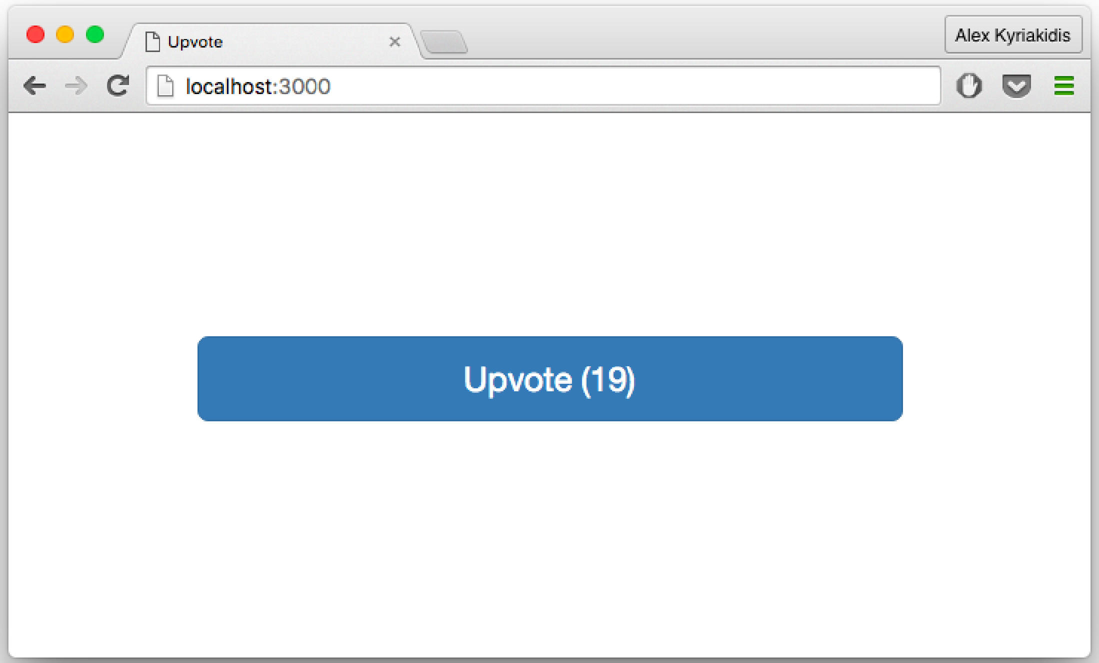
The `keyup` event occurs when the user releases a key. You can find a full list of HTML events here[1].

1.1.1 Handling Events Inline

Enough with the talking, let's move on and see event handling in action. Below, there is an 'Upvote' button which increases the number of upvotes every time it gets clicked.

[1]http://www.w3schools.com/tags/ref_eventattributes.asp

```html
<html>
<head>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
s" rel="stylesheet">
<title>Upvote</title>
</head>
<body>
    <div class="container">
        <button v-on:click="upvotes++">
            Upvote! {{upvotes}}
        </button>
    </div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
<script type="text/javascript">
new Vue({
    el: '.container',
    data: {
        upvotes: 0
    }
})
</script>
</html>
```



Upvotes counter

As you can see above, we have a basic setup and this time we use the class `container` in our view model. There is an `upvotes` variable within our data. In this case, we bind an event listener for `click`, with the statement that is right next to it. Inside the quotes we're simply increasing the count of upvotes by one, each time the button is pressed, using the increment operator (`upvotes++`).

Shown above is a very simple inline JavaScript statement.

1.1.2 Handling Events using Methods

Now we are going to do the exact same thing as before, using a method instead. A method in Vue.js is a block of code designed to perform a particular task. To execute a method, you have to define it and then invoke it.

```
<html>
<head>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
s" rel="stylesheet">
<title>Upvote</title>
</head>
<body>
    <div class="container">
        <button v-on:click="upvote">
            Upvote! {{upvotes}}
        </button>
    </div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
<script type="text/javascript">
new Vue({
    el: '.container',
    data: {
        upvotes: 0
    },
    // define methods under the **`methods`** object
    methods: {
        upvote: function(){
            // **`this`** inside methods points to the Vue instance
            this.upvotes++;
        }
    }
})
</script>
</html>
```

We are binding a click event listener to a method named '**upvote**'. It works just as before, but cleaner and easier to understand when reading your code.

Warning

Event handlers are restricted to execute **one statement only**.

1.1.3 Shorthand for `v-on`

When you find yourself using `v-on` all the time in a project, you will find out that your HTML will quickly becomes dirty. Thankfully, there is a shorthand for `v-on`, the `@` symbol. The `@` replaces the entire `v-on:` and when using it, the code looks *a lot cleaner*, but everyone has their own practices and this is totally optional.

Using the shorthand, the button of our previous example will be:

Listening to 'click' using `v-on:`

```
<button v-on:click="upvote">
    Upvote! {{upvotes}}
</button>
```

Listening to 'click' using `@` shorthand

```
<button @click="upvote">
    Upvote! {{upvotes}}
</button>
```

1.2 Event Modifiers

Now we will move on and create a Calculator app. To do so, we'll use a form with two inputs and one dropdown to select the desired operation.

Even though the following code seems fine, our calculator does not work as expected.

```
<html>
<head>
  <title>Calculator</title>
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\
css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <h1>Type 2 numbers and choose operation.</h1>
    <form class="form-inline">
        <!-- Notice here the special attribute 'number'
        is passed in order to parse inputs as numbers.-->
        <input v-model="a" number class="form-control">
        <select v-model="operator" class="form-control">
           <option selected>+</option>
           <option>-</option>
           <option>*</option>
           <option>/</option>
        </select>
        <!-- Notice here the special attribute 'number'
        is passed in order to parse inputs as numbers.-->
        <input v-model="b" number class="form-control">
        <button type="submit" @click="calculate"
        class="btn btn-primary">
            Calculate
        </button>
    </form>
    <h2>Result: {{a}}  {{operator}}  {{b}} = {{c}}</h2>
    <pre>
        {{$data | json}}
    </pre>
</div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
<script type="text/javascript">
  new Vue({
    el: '.container',
    data: {
      a: 1,
      b: 2,
      c: null,
      operator: " ",
  },
  methods:{
      calculate: function(){
        switch (this.operator) {
          case "+":
              this.c = this.a + this.b
              break;
          case "-":
              this.c = this.a - this.b
              break;
          case "*":
              this.c = this.a * this.b
              break;
          case "/":
              this.c = this.a / this.b
              break;
      }
  }
},
});
</script>
</html>
```

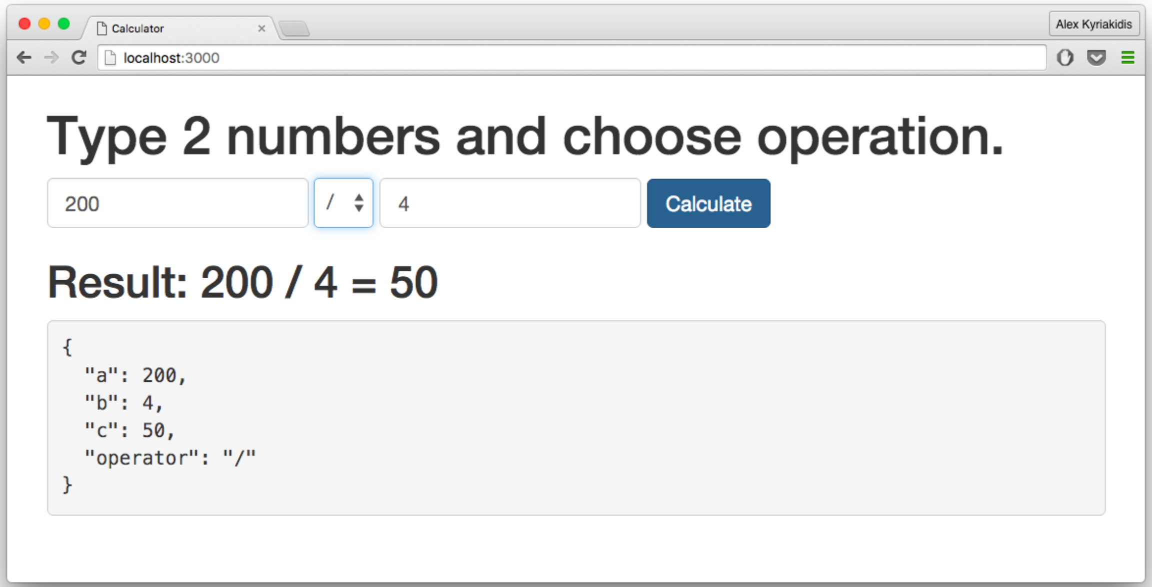
If you try and run this code yourself, you will find out that when the “calculate” button is clicked, instead of calculating, it reloads the page.

This makes sense because when you click “calculate”, in the background, you are submitting the form and thus the page reloads.

To prevent the submission of the form, we have to cancel the default action of the **`onsubmit`** event. It is a very common need to call **`event.preventDefault()`** inside our event handling method. In our case the event handling method is called **`calculate`**.

So, our method will become:

```
calculate: function(){
    event.preventDefault();
    switch (this.operator) {
        case "+":
            this.c = this.a + this.b
            break;
        case "-":
            this.c = this.a - this.b
            break;
        case "*":
            this.c = this.a * this.b
            break;
        case "/":
            this.c = this.a / this.b
            break;
    }
}
```



Type 2 numbers and choose operation.

200 / 4

Result: 200 / 4 = 50

```
{
  "a": 200,
  "b": 4,
  "c": 50,
  "operator": "/"
}
```

Using Event Modifiers to build a calculator.

Although we can do this easily inside methods, it would be better if the methods can be purely ignorant about data logic rather than having to deal with DOM event details.

Vue.js provides two event modifiers for `v-on` to prevent the event default behavior:

1. `.prevent`
2. `.stop`

So, using one of them, our submit button will change **from**:

```html
<button type="submit" @click="calculate">Calculate</button>
```

to:

```html
<button type="submit" @click.prevent="calculate">Calculate</button>
<!-- or -->
<button type="submit" @click.stop="calculate">Calculate</button>
```

And we can now safely remove `event.preventDefault()` from our `calculate` method.

1.3 Key Modifiers

If you hit enter when you focus on one of the inputs, you will notice that the page reloads again instead of calculating. This happens because we have prevented the behavior of the submit button but not of the inputs.

To fix this, we have to use '**Key Modifiers**'.

```
<input v-model="a" @keyup.enter="calculate">
<input v-model="b" @keyup.enter="calculate">
```

Tip

When you have a form with a lot of inputs/buttons/etc and you need to prevent their default submit behavior, you can modify the `submit` event of the form. Example: <form @submit.prevent="calculate">

Finally, the calculator is up and running.

1.4 Computed Properties

Vue.js inline expressions are very convenient, but for more complicated logic, you should use computed properties. Practically, computed properties are variables which their value depends on other factors.

Computed properties work like functions that you can use as properties. But there is a significant difference. Every time a dependency of a computed property changes, the value of the computed property re-evaluates.

In Vue.js, you define computed properties within the `computed` object inside your `Vue` instance.

```
<html>
<head>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
s" rel="stylesheet">
<title>Hello Vue</title>
</head>
<body>
<div class="container">
    a={{ a }}, b={{ b }}
   <pre>
```

```
        {{$data | json}}
    </pre>
</div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
<script type="text/javascript">
new Vue({
    el: '.container',
    data: {
      a: 1,
    },
    computed: {
      // a computed getter
      b: function () {
        // **`this`** points to the Vue instance
        return this.a + 1
      }
    }
});
</script>
</html>
```

This is a basic example demonstrating the use of computed properties. We've set two variables, the first, **a**, is set to 1 and the second, **b**, will be set by the returned result of the function inside the computed object. In this example the value of **b** will be set to 2.

```
<html>
<head>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
s" rel="stylesheet">
<title>Hello Vue</title>
</head>
<body>
<div class="container">
    a={{ a }}, b={{ b }}
   <input v-model="a">
   <pre>
        {{$data | json}}
   </pre>
</div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
```

```
<script type="text/javascript">
new Vue({
    el: '.container',
    data: {
      a: 1,
    },
    computed: {
      // a computed getter
      b: function () {
        // **`this`** points to the vm instance
        return this.a + 1
      }
    }
});
</script>
</html>
```

The above example is the same as the previous one, but with one difference. An input is binded to the `a` variable. The desired outcome would be to change the value of the binded attribute and immediately update the result of `b`. But notice here, that it does not work as we would expect.

If you run this code and enter an input for variable `a` the number 5, you expect that `b` will be set to 6. Sure, but it doesn't, `b` is set to 51.

Why is this happening? Well, as you might have already thought, `b` takes the given value from the input ("a") as a string, and appends the number 1 at the end of it.

One solution to solve this problem is to use the **`parseFloat()` function that parses a string and returns a floating point number**.

```
new Vue({
    el: '.container',
    data: {
      a: 1,
    },
    computed: {
      b: function () {
        return parseFloat(this.a) + 1
      }
    }
});
```

Another option that comes to mind, is to use the `<input type="number">` which is used for input fields that should contain a numeric value.

But there is a more neat way. With Vue.js, whenever you want your user's inputs to be automatically persisted as numbers, you can add the special attribute **`number`** to these inputs.

```
<body>
<div class="container">
    a={{ a }}, b={{ b }}
    <input v-model="a" number>
    <pre>
        {{$data | json}}
    </pre>
</div>
</body>
```

The **`number`** attribute is going to give us the desired result without any further effort.

To demonstrate a wider picture of computed properties, we are going to make use of them and build the calculator we have already shown, but this time using computed properties instead of methods.

Lets start with a simple example, where a computed property **`c`** contains the sum of **`a`** plus **`b`**.

```
<html>
<head>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
n.css" rel="stylesheet">
    <title>Hello Vue</title>
</head>
<body>
    <div class="container">
        <h1>Enter 2 numbers to calculate their sum.</h1>
        <form class="form-inline">
            <input v-model="a" number class="form-control">
            +
            <input v-model="b" number class="form-control">
        </form>
        <h2>Result: {{a}} + {{b}} = {{c}}</h2>
        <pre> {{$data | json}} </pre>
    </div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.js"></script>
<script type="text/javascript">
    new Vue({
      el: '.container',
      data: {
        a: 1,
        b: 2
    },
    computed: {
    c: function () {
      return this.a + this.b
  }
}
});
</script>
</html>
```

The initial code is ready, and at this point the user can type in 2 numbers and get the sum of these two. A calculator that can do the four basic operations is the goal, so let's continue building!

Since the HTML code will be the same with the calculator we build in the previous section of this chapter (except now we don't need a button), I am am going to show you here only the Javascript codeblock.

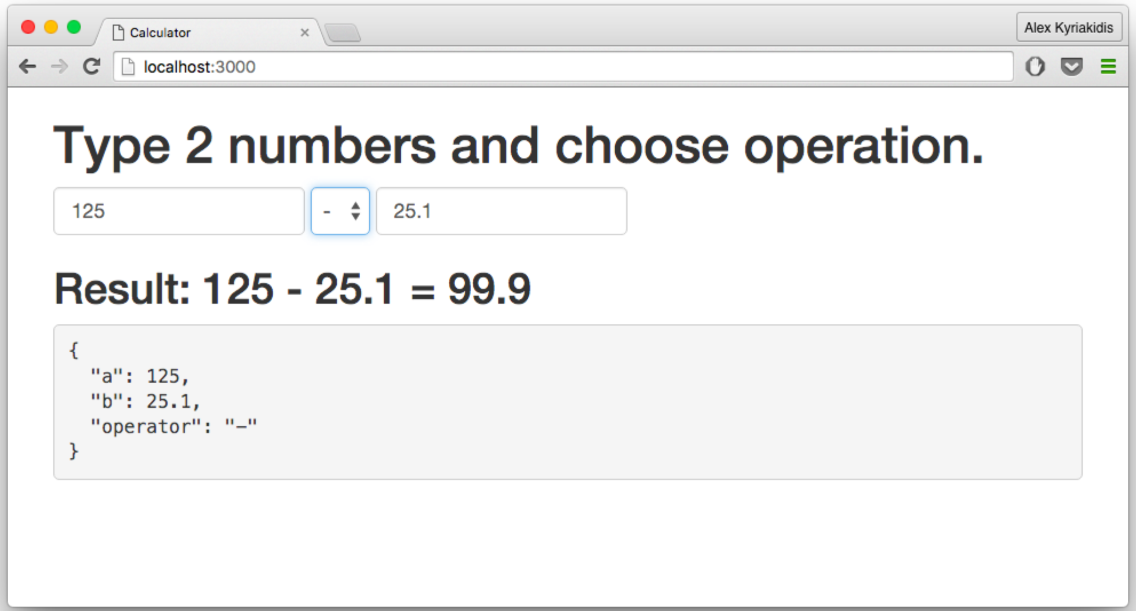
```
  new Vue({
          el: '.container',
          data: {
              a: 1,
              b: 2,
              operator: " ",
          },
          computed: {
              c: function () {
                  switch (this.operator) {
                      case "+":
                      return this.a + this.b
                      break;
                      case "-":
                      return this.a - this.b
                      break;
                      case "*":
                      return this.a * this.b
                      break;
                      case "/":
                      return this.a / this.b
                      break;
                  }
              }
```

```
        },
    });
```

The calculator is ready for use. The only thing we had to do, was to move whatever was inside `calculate` method to the computed property `c`! Whenever you change the value of `a` or `b` the result updates in real time! We don't need any buttons, events, or anything. **How awesome is that??**

Info

Note here that a normal approach would be to have an `if` statement to avoid error of division. The best part about this, is that there is already a prediction for this kind of flaws. If the user types 1/0 the result automatically becomes infinity! If the user types a text the displayed result is "not a number".



Type 2 numbers and choose operation.

125 - 25.1

Result: 125 - 25.1 = 99.9

```
{
  "a": 125,
  "b": 25.1,
  "operator": "-"
}
```

Calculator built with computed properties.

1.4.1 Using Computed Properties to Filter an Array

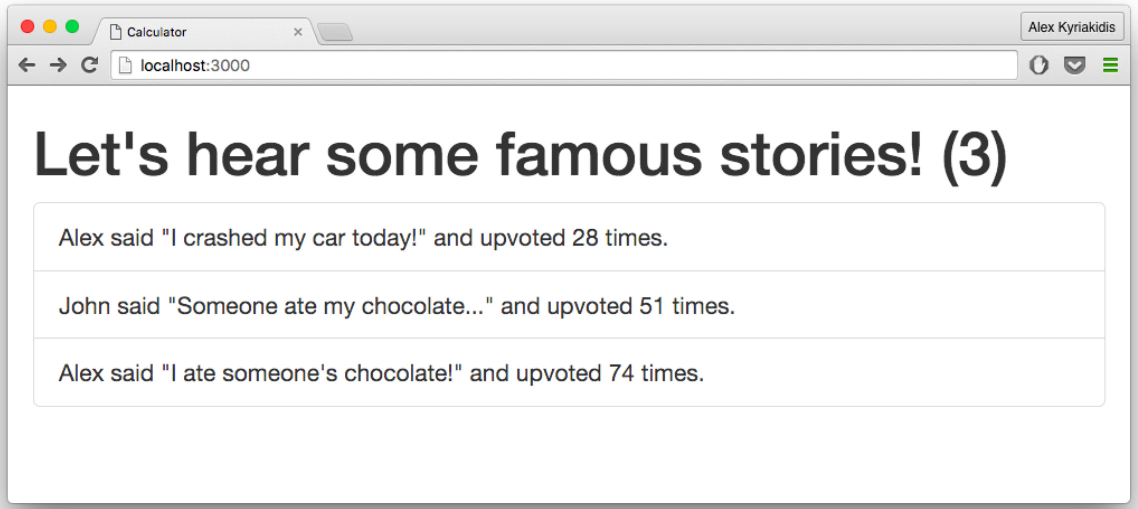
A *computed property* can also be used to filter an array. Using a computed property to perform array filtering gives you in-depth control and more flexibility, since it's full JavaScript, and allows you to access the filtered result elsewhere. For example you can get the length of a filtered array anywhere in your code.

To see how it's done, we will filter the **`famous`** stories as we did in the Custom Filter example. This time we will create a computed property that returns the filtered Array.

```
new Vue({
    el: '.container',
    data: {
        stories: [
            {
                plot: "I crashed my car today!",
                writer: "Alex",
                upvotes: 28
            },
            {
                plot: "Yesterday, someone stole my bag!",
                writer: "John",
                upvotes: 8
            },
            {
                plot: "Someone ate my chocolate...",
                writer: "John",
                upvotes: 51
            },
            {
                plot: "I ate someone's chocolate!",
                writer: "Alex",
                upvotes: 74
            },
        ]
    },
    computed: {
        famous: function() {
            return this.stories.filter(function(item){
                return item.upvotes > 25;
            });
        }
    }
})
```

In our HTML code, instead of **`stories`** array, we will render the **`famous`** computed property.

```html
<body>
    <div class="container">
    <h1>Let's hear some famous stories! ({{famous.length}})</h1>
        <ul class="list-group">
            <li v-for="story in famous"
            class="list-group-item"
            >
                {{ story.writer }} said "{{ story.plot }}"
                and upvoted {{ story.upvotes }} times.
            </li>
        </ul>
    </div>
</body>
```



Let's hear some famous stories! (3)

Alex said "I crashed my car today!" and upvoted 28 times.

John said "Someone ate my chocolate..." and upvoted 51 times.

Alex said "I ate someone's chocolate!" and upvoted 74 times.

Filter array using a computed property

That's it. We have filtered our array using a computed property. Did you notice how easily we managed to display the *number of famous stories* next to our heading message using **`{{famous.length}}`**?

Info

Although using a **computed property** to perform array filtering, gives you more flexibility, **array filters** can be more convenient for common use cases.

Code Examples

You can find the code examples of this chapter on GitHub[2].

[2]https://github.com/hootlex/the-majesty-of-vuejs/tree/master/examples/5.%20Interactivity

1.5 Homework

Now that you have a basic understanding of Vue's event handling, methods, computed properties etc, you should try something a bit more challenging. Start by creating an array of "Mayor" candidates. Each candidate has a "name" and a number of "votes". Use a button to increase the count of votes for each candidate. Use a computed property to determine who is the current "Mayor", and display his name.

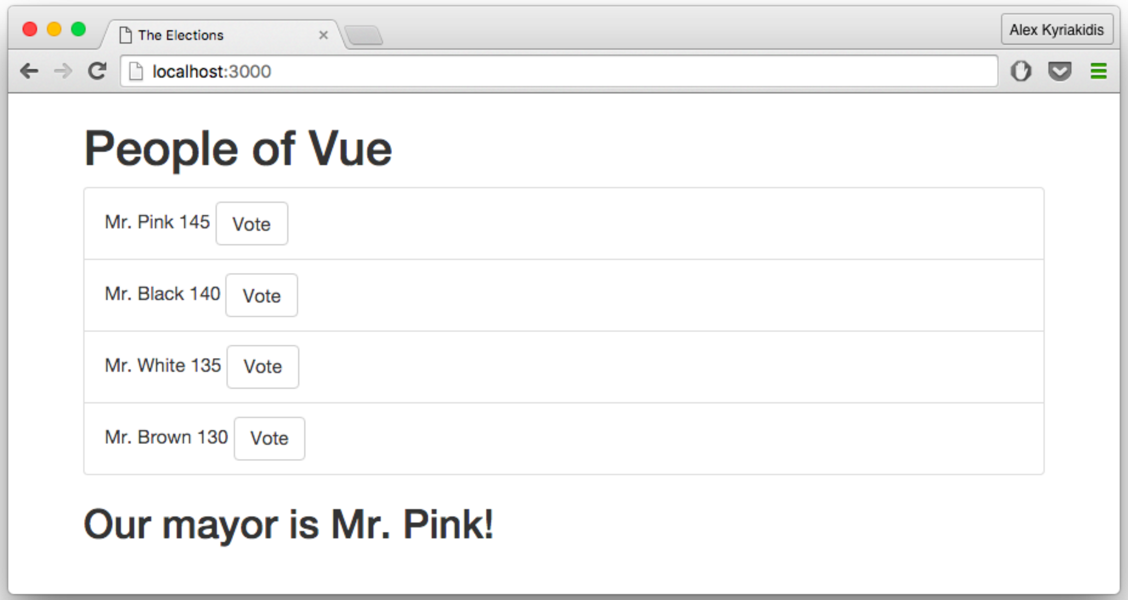
Finally when key '**c**' is pressed the elections start from the beginning, and all votes become 0.

Hint

Javascript's `sort()` and `map()` methods could prove very useful and Key modifiers will get you there.

Hint 2

To listen globally for events you should target the `body` element.



People of Vue

Mr. Pink 145 Vote

Mr. Black 140 Vote

Mr. White 135 Vote

Mr. Brown 130 Vote

Our mayor is Mr. Pink!

Example Output

You can find a potential solution to this exercise here[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter5.html