

# Vue Event Handling

Module 3: 12

**Last time in**



**VUE.JS**

# VUE IS A JS FRAMEWORK



**Vue is a progressive framework**

Implement by extending HTML with new tags or attributes

Event  
Management

Easy DOM  
Manipulation

Established  
Patterns

Reusable  
Components

Data Binding

URL  
Management

# VUE IS JAVASCRIPT



**==**



# Vue.js vs Vanilla.js



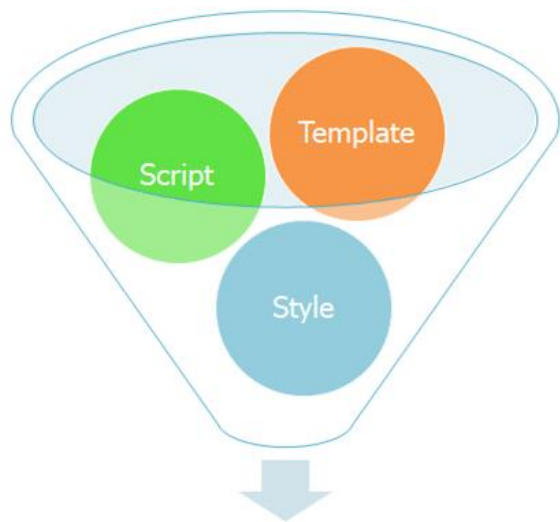
VS



[Vanilla.js](#) - it is just a simple, plain JavaScript code, without any additional libraries...

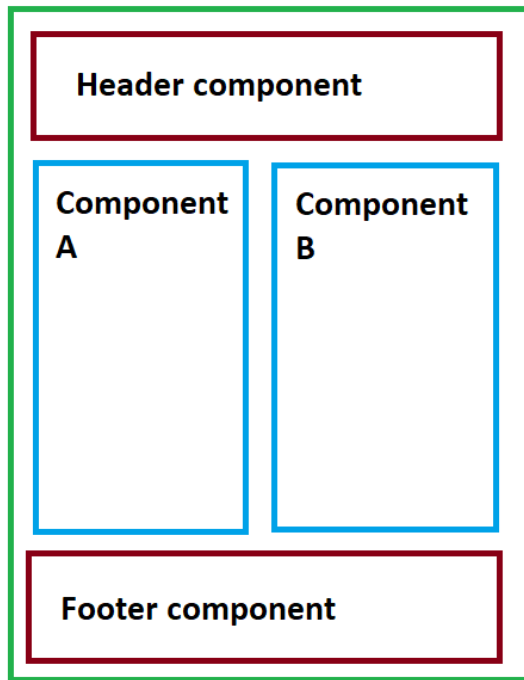
- Vue.js is a JavaScript framework
- Vanilla.js is a term how plain JavaScript code is referred.

# Components

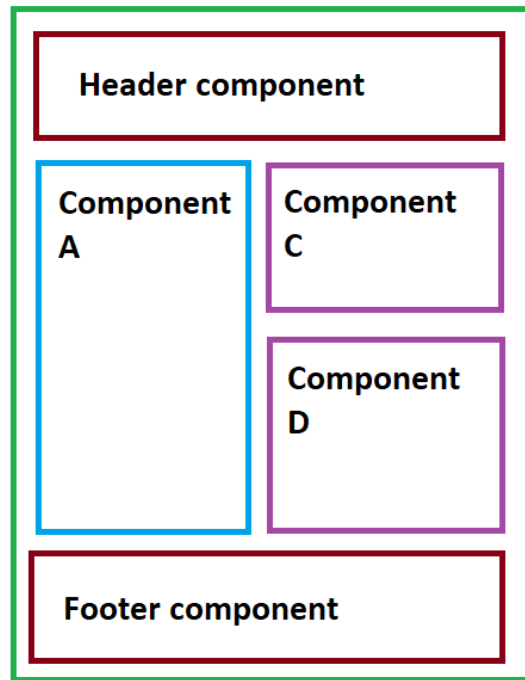


Vue.js Component

Page A



Page B



# 1-Way Data Binding

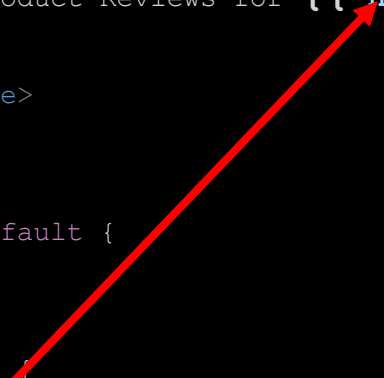
- Binds data from the view model data() return to be displayed in the template.
- If the data is changed Vue reacts by updating the display in the template.
- Uses {{ variable }}
- The double curly brace is called a mustache tag.

```
<template>
  <div class="main">

    <h2>Product Reviews for {{ name }}</h2>

    ...
  </template>

<script>
export default {
  ...
  data() {
    return {
      name: "Cigar Parties for Dummies",
    }
    ...
  }
}
</script>
```



# 2-Way Data Binding

- Binds data from the view model data() return to a form element.
- If the data is changed Vue reacts by updating the value of the form element.
- If the value of the form element is changed Vue reacts by updating the value of the data.
- Uses v-model="variable"

```
<template>
  <div id="addemail">
    ...

    <input type="text" id="name"
      v-model="userName">
    ...
  </template>

<script>
export default {
  data() {
    return {
      userName: '',
      ...
    }
  }
}
</script>
```

A red arrow originates from the `v-model="userName"` attribute in the `<input>` tag within the `<template>` block and points diagonally down and to the left towards the `userName: ''` property in the `data()` function's return object within the `<script>` block. This visualizes the connection between the view (the input field) and the model (the data property).

# Loops ( v-for )

- Repeats an HTML element by looping over an array in the data like a foreach loop.
- If the array is changed Vue reacts by redrawing the looped elements.
- A variable is created to hold each element of array, which can be used in the tag being repeated and its children.
- Required a key attribute on the tag being repeated that is bound with v-bind to a unique property in the array items.
- v-for="variable in array"
- v-bind:key="variable.uniqueProperty"

```
<template>
  <div class="main">
    ...
    <div
      v-for="review in reviews"
      v-bind:key="review.id"
    >
      <h4>{{ reviewReviewer }}</h4>
      ...
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      ...
      reviews: []
    }
  },
}
```





# Attribute Binding ( v-bind )

- Allows for a single, simple line of JavaScript to be evaluated for the value of an attribute.
- Can be bound to data in an v-for variable or in the data() to set the value of an attribute.
  - `v-bind:attribute="variable"`
- Can use a simple boolean expression to conditionally set the value of the attribute.
  - `v-bind:class="{ value: boolean }"`

Setting a value:

`v-bind:attribute="variable"`

```
<img
  ...
  v-bind:title="review.rating + ' Star
Review'"
/>
```

Conditionally setting a value:

`v-bind:attribute="{ value : booleanCondition }"`

```
<div
  ...
  v-bind:class="{ favorited: review.favorited }"
  ...
>
```

# Computed Properties

- Creates a calculated property that can be 1-way bound to the template.
- Uses values from the data() to calculate the value. If the data value is updated Vue reacts by computing the property again and updating the template with the new value.
- Must return a value and cannot take parameters
- Added to the computed: section of the script.
- Requires the **this** keyword to access data in other parts of the view model.

```
<template>
  ...
  <span class="amount">{{ averageRating }}</span>
  ...
</template>

<script>
export default {
  data() {
    return {
      ...
      reviews: []
    }
  },
  computed: {
    averageRating() {
      let sum = this.reviews.reduce(...);
      return sum / this.reviews.length;
    },
  }
}
```

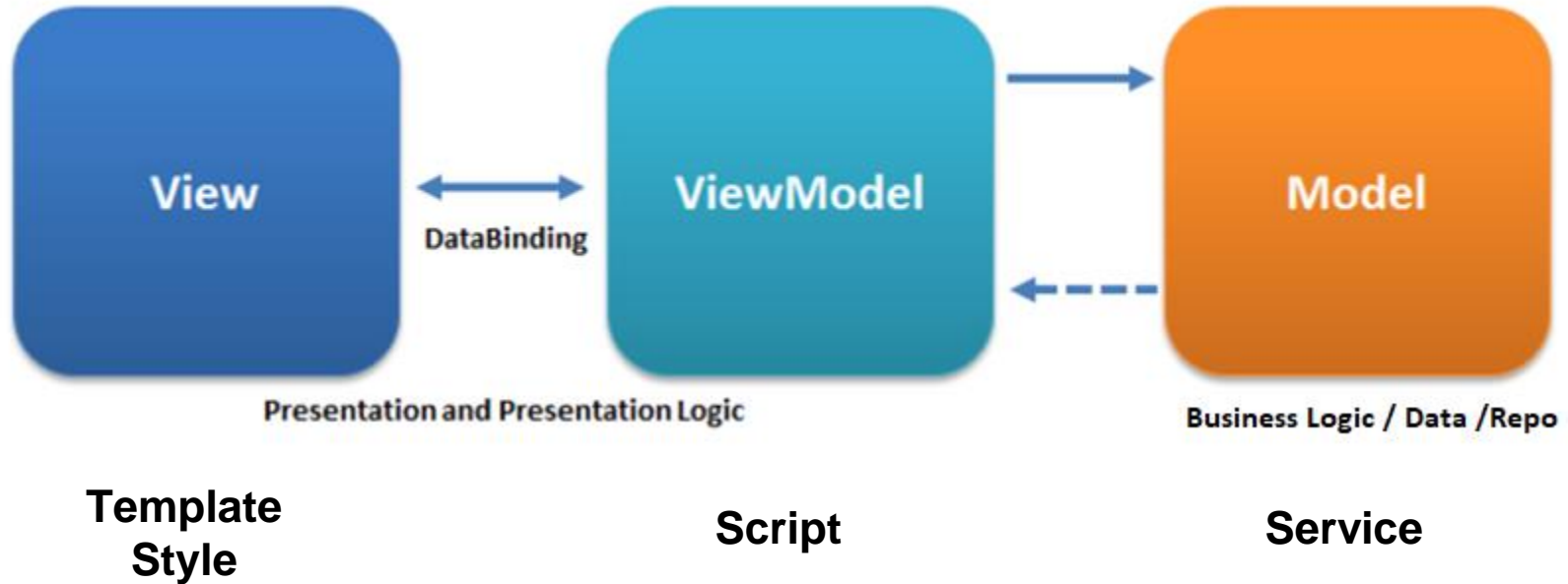
# Today in



# VUE.JS

1. MVVM (Model View ViewModel) Pattern
2. Binding a Form to a new Object
3. Methods
4. Vue Event Handling
  - a. Event Listening with v-on
  - b. Event Modifiers
    - i. Keyboard and Mouse Event modifiers
  - c. Event Actions
    - i. Inline Events
    - ii. Event Methods
  - d. The Event object

# MVVM (Model View ViewModel)



# Component ViewModel

The JavaScript code in the export default object of the Script. This includes the data, computed properties, and methods.

Code in the template has access to the ViewModel by default.

Outside of the template, the ViewModel can be referenced using the *this* keyword.

```
<script>
export default {
  name: 'product-review',
  data() {
    return {
      name: 'Cigar Parties for Dummies',
      description: 'Host and plan the perfect',
      newReview: {},
      reviews: []
    };
  },
  computed: {
    averageRating() {},
    numberOfOneStarReviews() {},
    numberOfTwoStarReviews() {},
    filteredReviews() {}
  },
  methods: {
    addNewReview() {},
    resetForm() {},
    numberOfReviews(numOfStars) {}
  }
};
</script>
```

# Binding Forms to a new Object

Forms can be bound to an empty object that represents it. When the form is bound the keys used will be created in the object.

Object in Code

```
person: {},
```

Form

```
<input id="firstName"
      type="text"
      v-model.trim="person.firstName">

<input id="lastName"
      type="text"
      v-model.trim="person.lastName">
```

Resulting Object

```
person: {
  firstName:
  value,
  lastName: value
}
```

# Vue Methods

- Added in the methods: {} section.
- Can have arguments.
- Methods can use the **this** keyword.  
**this** refers to the ViewModel of *this component* and can be used to access the data or other methods in the component.

```
<script>
export default {
  name: 'product-review',
  data() { ...
  },
  computed: { ...
  },
  methods: {
    addNewReview() {
      this.reviews.unshift(this.newReview);
      this.resetForm();
    },
    resetForm() {
      this.newReview = {};
      this.showForm = false;
    },
    numberOfReviews(numOfStars) {
      return this.reviews.reduce((currentCount, review) => {
        return currentCount + (review.rating === numOfStars);
      }, 0);
    }
  }
};
</script>
```

# VUE Methods

- A VUE method is similar to a function or method in other languages - they are called when needed, optionally taking in parameters and providing some kind of output.
- Just like with the computed section, the methods section is comprised of JavaScript, thus should be part of the script section in a VUE component.



# VUE Methods vs Computed Properties

Methods and Computed properties were designed for different purposes.

- You use a computed property, to generate “derived data” in which your output is based on the data in your JSON data model.
  - Computed values are cached once encountered.
- You use a method when you want a tool that resembles a function in other languages.
  - Methods are executed only when called.

# Defining VUE Methods

VUE methods go into their own section, they are a peer of the data and computed section.

```
<script>
export default {
  name: "product-review",
  data() {
    ...
  },
  computed: {
    ...
  },
  methods: {
    //your methods go here
  }
}
</script>
```

# Defining VUE Methods

VUE methods are defined in a similar fashion as computed properties, with successive methods split by a comma:

```
methods: {  
  numberOfReviews(reviews, starType) {  
    return reviews.reduce( (currentCount, review ) => {  
      return currentCount + ( review.rating === starType ? 1 : 0);  
    }, 0);  
  },  
  
  addNewReview() {  
    this.reviews.unshift(this.newReview);  
    this.resetForm();  
  },  
  
  resetForm() {  
    this.showForm = false;  
    this.newReview = {};  
  }  
}
```

- Here we have three distinct methods being defined.
- The first method shows that a method can take on parameters and return a value.

[unshift vs push](#)

# Calling VUE Methods

VUE methods work flexibly and can be called in the following contexts:

- Within a v-on directive in the template section (more on this later)
- By a computed property: When we do this, the computed property needs to use this i.e. **this.myMethod()**;
- By another method.

# Let's Create Some Methods





Vue  
Events

# Event Listening with v-on

- Adds an event listener for an element
- [v-on Documentation](#)
- Syntax: `v-on: <event>=" <action to take> "`

```
v-on:click="showForm = true"
```

- Can use `@` to alias v-on:

```
@click="showForm = true"
```

- The Action can either be a single line of JavaScript or a method call

```
v-on:click="showForm = true"
```

```
v-on:click="showForm() "
```

# Event Modifiers

- Syntax: `v-on:<event>.<modifier (optional)>=" <action to take> "`
- Modifiers ([documentation](#)):
  - `.stop` → Stops Propagation
  - `.prevent` → Prevents default behavior
  - `.once` → Event is triggered only once

```
v-on:click.prevent="showForm = true"
```

- Modifiers can be chained - and are applied in the order indicated

```
v-on:click.prevent.once="showForm = true"
```



# Keyboard and Mouse Event Modifiers

- Alias exist for Keyboard keys and Mouse Buttons for Keyboard and mouse events.
- For Keyboard any regular JavaScript [KeyboardEvent.key](#) value can be used by converting them to all [lower-kebab-case](#)

Vanilla JavaScript `PageDown` : `<input v-on:keyup.page-down="onPageDown">`

`ArrowDown`: `keyup.arrow-down`

- Mouse events have modifiers that can restrict the event to specific mouse buttons:

- `.left`

`on.click.right.prevent="handleRightMouseButtonClick() "`

- `.right`

- `.middle`

`v-on.click.right="handleRightMouseButtonClick() "`

`v-`

# Event Actions

- Event actions can be a single line of JavaScript

```
v-on:click="showForm = true"
```

- Event actions can also be a method

```
v-on:click="showFormMethod() "
```

- When the action is a method, if no arguments are passed then the parenthesis can be omitted

```
v-on:click="showFormMethod"
```

- Arguments (0...n) can be passed to to an method from an event action

```
v-on:click="showFormMethod(2, true) "
```

# The Event Object

The Event Object will be passed as an implicit variable to any method that does not take any arguments

```
v-on:click="showFormMethod"
```

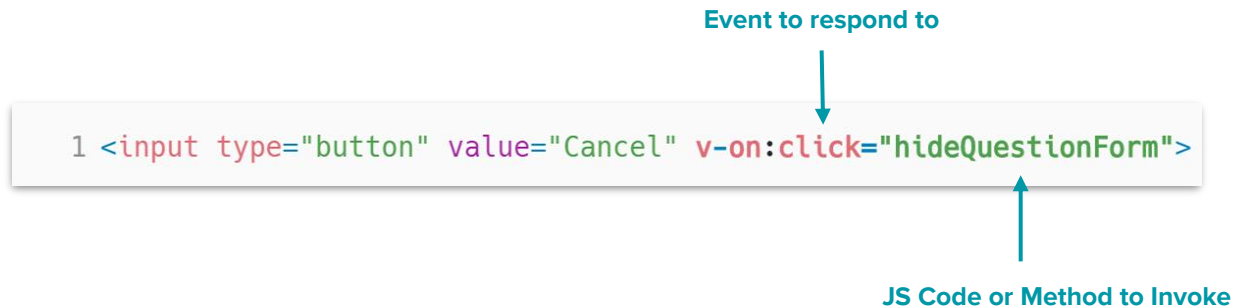
```
function showFormMethod(event) {  
    event.target;  
}
```

If a method takes other arguments, then the event object can be passed to a method using `$event`

```
v-on:click="showFormMethod(2, $event)"
```

```
function showFormMethod(num, event) {  
    event.target;  
}
```

# Another look at the **v-on** directive...



# The v-on directive

- The v-on directive takes on the following pattern:

**v-on:** **<<event>>** = '**<<action to take>>**'

- Here are some examples:

Here we say: when the user clicks on the span, set the JSON data property to counter + 1.

```
<span class="amount" v-on:click="counter += 1">Button has been clicked {{ counter }} times </span>
```

Here we say: when the user click on an anchor, call the method **addNewReivew**

```
<a v-on:click="addNewReview">Add new Review</a>
```

# v-on events

- `v-on:click="someMethod"`
- `v-on:change="someMethod"`
- `v-on:submit="someMethod"`
- `v-on:keyup="someMethod"`
- `v-on:blur="someMethod"`
- ...
- Basically anything we had before, just in Vue.

# v-on keyboard events

- `v-on:keyup.enter="someMethod"`
- `v-on:keyup.space="someMethod"`
- `v-on:keyup.page-down="someMethod"`
- `v-on:keyup.up="someMethod"`
- `v-on:keyup.down="someMethod"`
- `v-on:keyup.left="someMethod"`
- `v-on:keyup.right="someMethod"`

# v-on directive: inline handler

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

```
<script>
export default {
  name: "app",
  data() {
    return {
      counter: 0
    };
  },
}
```



## v-on directive: method event handler

```
<input type="submit" value="Save">  
<input type="button" value="Cancel" v-on:click.prevent="resetForm">  
</form>
```

```
  },  
  methods: {  
    resetForm() {  
      this.newReview = {};  
      this.showForm = false;  
    },  
  },  
}
```

# v-on modifiers

- **v-on:click.stop** - Identical to **event.stopPropagation()**
- **v-on:click.prevent** - Identical to **event.preventDefault()**

# Event modifiers

- Just like in Vanilla JS, we may want to prevent default action or stop propagation:

Here we saying: when the user submits the form, call the method **addNewReivew**

```
<form v-if="showForm === true" v-on:submit.prevent="addNewReview">
```

# Event modifiers: prevent

- The v-on directive can be modified with a prevent keyword, which prevents the default behavior of a HTML element from executing:

```
<form v-if="showForm === true" v-on:submit.prevent="addNewReview">
```

Note that we are overriding the default behavior of the form submission, and instead choosing to handle the scenario ourselves with our own method.

# Event Modifiers: stop

- The v-on directive can be modified with a stop keyword, disabling event bubbling up the DOM.

```
<a v-on:click.stop="modifyNewReview">
```

Note that we are stopping the propagation from bubbling up through the DOM.

# \$event variable

- We may need to pass the original DOM event to a method

```
<button v-on:click="warn('Form cannot be submitted yet.', $event)">
```

```
// ...  
methods: {  
  warn(message, event) {  
    if (event) {  
      event.preventDefault()  
    }  
    alert(message)  
  }  
};
```

```
<template>  
  <div id="app">  
    <a href="#" id="increase" class="btn" v-on:click="updateCounter($event)">Increase</a>  
    <a href="#" id="decrease" class="btn" v-on:click="updateCounter($event)">Decrease</a>  
    <p>The button was clicked {{ counter }} times</p>  
  </div>  
</template>
```

```
},  
methods: {  
  updateCounter(event) {  
    if (event.target.id === "increase") {  
      this.counter += 1;  
    } else {  
      this.counter -= 1;  
    }  
  }  
}
```

# v-if and v-else

```
var vm = new Vue({  
  el: '#example',  
  data: {  
    a: true,  
    b: false  
  }  
});
```

```
<!-- will render 'The condition is true' into the DOM -->  
<div id="example">  
  <h1 v-if="a">The condition is true</h1>  
</div>
```

```
<!-- in this case, nothing will be rendered except for the containing 'div' -->  
<div id="example">  
  <template v-if="b">  
    <h1>Heading</h1>  
    <p>Paragraph 1</p>  
    <p>Paragraph 2</p>  
  </template>  
</div>
```

```
<div v-if="'a' === 'b'"> This will never be rendered. </div>  
<template v-else>  
  <ul>  
    <li> You can also use templates with v-else. </li>  
    <li> All of the content within the template </li>  
    <li> will be rendered. </li>  
  </ul>  
</template>
```

# v-if

The v-if directive will render a DOM element only if certain conditions are met. Consider the following:

```
<template>
  <div class="main">
    <p>Only Bob can see this:</p>
    <p class="description" v-if="name === 'Bob'">Hello {{name}} this
      message will self destruct in 10 seconds.</p>
  </div>
</template>
```

```
<script>
export default {
  name: 'product-review',
  data() {
    return {
      name: 'Bob',
      description: 'secret agent'
    }
  }
}
</script>
```

Only Bob can see this:

Hello Bob this message will self destruct in 10 seconds.

Note that the second paragraph has a v-if directive.

The element will only display if the name attribute is Bob.



# v-else

The v-else directive ONLY renders if the v-if is false. Consider the following:

```
<template>
  <div class="main">
    <p>Only Bob can see this:</p>
    <p class="description" v-if="name === 'Bob'">Hello {{name}} this
      message will self destruct in 10 seconds.</p>
    <p class="description" v-else>YOU ARE NOT BOB! <p>
  </div>
</template>
```

```
<script>
export default {
  name: 'product-review',
  data() {
    return {
      name: 'Tim',
      description: 'secret agent'
    }
  }
}
</script>
```

Only Bob can see this:  
YOU ARE NOT BOB!

# v-show

```
1 <span class="showAnswer"  
2     v-show="!question.isAnswerVisible">  
3     {{showAnswerText}}  
4 </span>
```

# Toggling form visibility

Product Reviews for Cigar Parties for Dummies

Host and plan the perfect cigar party for all of your squirrelly friends.

2.75	1	0	2	1	0
Average Rating	1 Star Review	2 Star Reviews	3 Star Reviews	4 Star Review	5 Star Reviews

Add Review

Name:

Title:

Rating:

Review:

Save Cancel

Malcolm Gladwell

★★★★ What a book!

It certainly is a book. I mean, I can see that. Pages kept together with glue and there's writing on it, in some language.

Favorite? ☐

Tim Ferriss

★★★★★ Had a cigar party started in less than 4 hours.

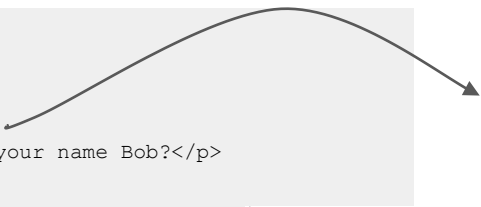
It should have been called the four hour cigar party. That's amazing. I have a new idea for muse because of this.

# v-show

The v-show will hide elements but still have them on the page. Consider the following:

```
<template>
  <div class="main">
    <p>Only Bob can see this:</p>
    <p class="description" v-show="a">Hello, is your name Bob?</p>
  </div>
</template>
```

```
<script>
export default {
  name: 'product-review',
  data() {
    return {
      a: false,
      description: 'secret agent'
    }
  }
}
</script>
```



```
▼ <div class="main">
  <p>Only Bob can see this:</p>
  <p class="description"> Hello Bob this message will self destruct in 10 seconds. </p>
  ...
  <p class="description" style="display: none;">Hello, is your name Bob?</p> == $0
  <p></p>
</div>
</div>
```