

# Section 02 - Vue Core Concepts

## Using Interpolation and v-bind

The screenshot shows a code editor and a browser side-by-side. The code editor has two tabs: 'index.html' and 'app.js'. The 'index.html' tab contains the following code:`1 <html lang="en">
2 <head>
3 <link rel="stylesheet" href="styles.css" />
4 <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
5 </head>
6 <body>
7 <header>
8 <h1>Vue Course Goals</h1>
9 </header>
10 <section id="user-goal">
11 <h2>My Course Goal</h2>
12 <p>{{ courseGoal }}</p>
13 <p>You can also visit <a v-bind:href="vueLink">Vue Page</a></p>
14 </section>
15 </body>
16 </html>`  
The 'app.js' tab contains the following code:`1 const app = Vue.createApp({
2 data: () => {
3 return {
4 courseGoal: "Complete Vue course and create Two Projects",
5 vueLink: "https://vuejs.org/",
6 };
7 },
8 );
9 app.mount("#user-goal");
10`

The browser window shows the output of the code. It features a green header with the text 'Vue Course Goals'. Below it is a card with the heading 'My Course Goal'. Inside the card, there is a button labeled 'Complete Vue Course and create Two Projects' and a link labeled 'You can also visit [Vue Page](#)'. A red callout box with the number '1' points to the interpolation in the HTML code: '

<{{ courseGoal }}>

'. Another red callout box with the number '2' points to the special syntax in the JavaScript code: 'v-bind:href="vueLink"'. Red arrows connect these callouts to their respective locations in the code.

Interpolation can also be used to return function value, or writing simple Javascript but not for complex if-statements and loops. Like

```

const app = Vue.createApp({
  data: () => {
    return {
      courseGoal: "Complete Vue Course and create Two Projects",
      vueLink: "https://vuejs.org/",
    };
  },
  methods: {
    createGoal() {
      if (Math.random() < 0.5) {
        return "Learn Vue";
      } else {
        return "Master Vue";
      }
    },
  },
});
app.mount("#user-goal");

```

```

<section id="user-goal">
  <h2>My Course Goal</h2>
  <p>{{ createGoal() }}</p>
  <p>You can also visit <a v-bind:href="vueLink">Vue Page</a></p>
</section>

```

`v-bind` is special feature in Vue that binds HTML attribute.

## Using Data Variables Inside App

We can also use data variables inside methods using `this` keyword, because Vue makes **data variable access globally** in `createApp` object. So, `this` refer to `createApp` object.

```
const app = Vue.createApp({
  data: () => {
    return {
      courseGoalA: "Complete Vue Course",
      courseGoalB: "Create Two Vue + Laravel Projects",
      vueLink: "https://vuejs.org/",
    };
  },
  methods: {
    createGoal() {
      if (Math.random() < 0.5) {
        return this.courseGoalA;
      } else {
        return this.courseGoalB;
      }
    },
  },
});
app.mount("#user-goal");
```

## Rendering Raw HTML

You can **render raw html** using `v-html` attribute in html tags

```

const app = Vue.createApp({
  data: () => {
    return {
      courseGoalA: "Complete Vue Course",
      courseGoalB: "<i>Create Two Vue + Laravel Projects</i>",
      vueLink: "https://vuejs.org/",
    };
  },
  methods: {
    createGoal() {
      if (Math.random() < 0.5) {
        return this.courseGoalA;
      } else {
        return this.courseGoalB;
      }
    },
  },
});
app.mount("#user-goal");

```

```

<body>
  <header>
    <h1>Vue Course Goals</h1>
  </header>
  <section id="user-goal">
    <h2>My Course Goal</h2>
    <!-- <p>{{ createGoal() }}</p> -->
    <p v-html="createGoal()"></p>
    <p>You can also visit <a v-bind:href="vueLink">Vue Page</a></p>
  </section>
</body>

```

# User Events

The screenshot shows a code editor with two files: `index.html` and `app.js`.

`index.html` content:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Vue Basics</title>
7     <link href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap" rel="stylesheet" />
8     <link rel="stylesheet" href="styles.css" />
9     <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
10    <script src="app.js" defer></script>
11  </head>
12  <body>
13    <header>
14      <h1>Vue Events</h1>
15    </header>
16    <section id="events">
17      <h2>Events in Action</h2>
18      <button v-on:click="counter++">Add</button>
19      <button v-on:click="counter--">Remove</button>
20      <p>Result: {{ counter }}</p>
21    </section>
22  </body>
23 </html>
```

`app.js` content:

```
1 const app = Vue.createApp({
2   data() {
3     return {
4       counter: 0,
5     };
6   },
7 });
8
9 app.mount('#events');
```

A red arrow points from the `v-on:click="counter++"` line in the `index.html` code to the explanatory note in the `app.js` code.

To handle user events, Vue JS provides `v-on` directive in which we can call any user event like `click`, `mouseenter`, `mouseleave`, etc.

Using `v-on` directive and then specifying event name, we can control event listener in Vue.js

It is good practice to pass `callback functions` in event listeners

```

const app = Vue.createApp({
  data() {
    return {
      counter: 0,
    };
  },
  methods: {
    increaseCount(num = 1) {
      this.counter = this.counter + num;
    },
    decreaseCount(num = 1) {
      this.counter = this.counter - num;
    },
  });
}

app.mount("#events");

```

```

<section id="events">
  <h2>Events in Action</h2>
  <button v-on:click="increaseCount(5)">Add</button>
  <button v-on:click="decreaseCount()">Remove</button>
  <p>Result: {{ counter }}</p>
</section>

```

## Passing Event Object

By default, Vue JS **sends event object to every event listener**. If you want to pass your argument as well then you need to first pass `$event` followed by your argument.

The screenshot shows a development environment with two tabs open: 'app.js' and 'index.html'. The 'app.js' tab contains the following code:

```
const app = Vue.createApp({  
  methods: {  
    increaseCount(num = 1) {  
      this.counter = this.counter + num;  
    },  
  
    decreaseCount(num = 1) {  
      this.counter = this.counter - num;  
    },  
  
    setName(event, lastName) {  
      this.name = event.target.value + " " + lastName;  
    },  
  },  
});
```

The 'index.html' tab contains the following code:

```
<html lang="en">  
<body>  
  <header>  
    <h1>Vue Events</h1>  
  </header>  
  <section id="events">  
    <h2>Events in Action</h2>  
    <button v-on:click="increaseCount(5)">Add</button>  
    <button v-on:click="decreaseCount()">Remove</button>  
    <p>Result: {{ counter }}</p>  
    <input type="text" v-on:input="setName($event, 'Sajwani')"/>  
    <p>{{ name }}</p>  
  </section>  
</body>  
</html>
```

A red arrow points from the '\$event' argument in the 'setName' method of 'app.js' to the 'v-on:input' directive in 'index.html', with the annotation: 'Vue JS passes event object to all user events, to send them we need to use \$event argument'.

The browser window on the right displays the application's UI. It has a green header bar with the title 'Events in Action'. Below it is a button labeled 'Add' and another labeled 'Remove'. A large green button displays the text 'Result: 0'. Below that is a text input field containing 'Tabish'. At the bottom, a green button displays the text 'Tabish Sajwani'. The browser's developer tools are visible at the bottom, showing the console tab with the message: 'You are running a development build of Vue. Make sure to use the production build (\*.prod.js) when deploying for production.' and the file 'vue\_global.js:12572'.

## Event Modifier

We can use event modifier in Vue Js. One of the example is using prevent to prevent default form behaviour.

The screenshot shows a code editor with two tabs open: `app.js` and `index.html`. The `app.js` file contains the following Vue component code:

```
1 const app = Vue.createApp({
2   data() {
3     return {
4       counter: 0,
5       name: '',
6     };
7   },
8
9   methods: {
10     increaseCount(num = 1) {
11       this.counter += num;
12     },
13
14     decreaseCount(num = 1) {
15       this.counter -= num;
16     },
17
18     setName(event, lastName) {
19       this.name = event.target.value + " " + lastName;
20     },
21
22     submitForm() {},
23   },
24 });
25
26 app.mount("#events");
27
```

The `index.html` file contains the following HTML and JavaScript code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Vue Basics</title>
<link href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap" rel="stylesheet" />
<link rel="stylesheet" href="styles.css" />
<script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
<script src="app.js" defer></script>
</head>
<body>
<header>
<h1>Vue Events</h1>
</header>
<section id="events">
<h2>Events in Action</h2>
<button v-on:click="increaseCount(5)">Add</button>
<button v-on:click="decreaseCount()">Remove</button>
<p>Result: {{ counter }}</p>
<input type="text" v-on:input="setName($event, 'Sajwani')"/>
<p>{{ name }}</p>
<form v-on:submit.prevent="submitForm">
<input type="text">
<button>Click Here</button>
</form>
</section>
</body>
</html>
```

A red arrow points from the explanatory text below to the `v-on:submit.prevent` directive in the `index.html` code. Another red arrow points from the explanatory text to the `submitForm()` method in the `app.js` code.

As we know we need to prevent submitting form with `event.preventDefault()`, this is one of the ways. Another way is event modifier like we are doing it here to prevent default submit.

# Calculating Value Once

If you want to **prevent Vue from recalculating** values again and again, you can use `v-once` directive.

```
<p v-once>Result: {{ counter }}</p>
```

## Changing Field Dynamically - `v-bind`

You can bind any component to change field dynamically using `v-bind` directive

The screenshot shows a developer environment with two tabs: `index.html` and `app.js`. The `index.html` tab displays the following code:`2 <html lang="en">
3 <head>
4 <link rel="stylesheet" href="styles.css" />
5 <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
6 <script src="app.js" defer></script>
7 </head>
8 <body>
9 <header>
10 <h1>Vue Course Goals</h1>
11 </header>
12 <section id="user-goal">
13 <h2>My Course Goal</h2>
14 <p>{{ courseGoal }}</p>
15 <p>You can also visit <a v-bind:href="vueLink">Vue Page</a></p>
16 </section>
17 </body>
18 </html>`

The `app.js` tab contains the following code:`1 const app = Vue.createApp({
2 data() => {
3 return {
4 courseGoal: "Complete Vue Course and create Two Projects",
5 vueLink: "https://vuejs.org/",
6 };
7 },
8 });
9 app.mount("#user-goal");
10`

Annotations in the code editor highlight specific parts of the code with red arrows and circles:

- Annotation 1: Points to the interpolation `{{ courseGoal }}` with the text: "Showing one of the variable value between the HTML tag is called interpolation."
- Annotation 2: Points to the attribute `v-bind:href="vueLink"` with the text: "Interpolation doesn't work in HTML tag attribute and for that we need to bind an attribute. Vue uses special syntax `v-bind:<attributeName>=<variable>"`"

The browser window on the right shows the resulting application interface:

- A green header bar with the text "Vue Course Goals".
- A white card with the title "My Course Goal".
- A button-like element with the text "Complete Vue Course and create Two Projects".
- A button-like element with the text "You can also visit [Vue Page](#)".

The browser's developer tools console at the bottom shows the message: "You are running a development build of Vue. Make sure to use the production build (\*.prod.js) when deploying for production." and the file path "vue.global.js:12572".

## Combining `v-bind` and `v-on` with `v-model`

`t-model` is syntax sugar for

- reading value
- updating state when input change

File Edit Selection View ... ← →

VueLearning

Section02 > basics-03-events-starting-code > app.js

```

1 const app = Vue.createApp({
2   data() {
3     return {
4       counter: 0,
5       name: '',
6     };
7   },
8   methods: {
9     increaseCount(num = 1) {
10       this.counter = this.counter + num;
11     },
12     decreaseCount(num = 1) {
13       this.counter = this.counter - num;
14     },
15     setName(event, lastName) {
16       this.name = event.target.value;
17     },
18     submitForm() {},
19     resetField() {
20       this.name = '';
21     },
22   },
23 });
24 app.mount("#events");
25 
```

Section02 > basics-03-events-starting-code > index.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1" />
6     <title>Vue Basics</title>
7     <link href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700" rel="stylesheet" />
8     <link rel="stylesheet" href="styles.css" />
9     <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
10    <script src="app.js" defer></script>
11  </head>
12  <body>
13    <header>
14      <h1>Vue Events</h1>
15    </header>
16    <section id="events">
17      <h2>Events in Action</h2>
18      <button v-on:click="increaseCount(5)">Add</button>
19      <button v-on:click.right="decreaseCount()">Remove</button>
20      <p v-once>Result: {{ counter }}</p>
21      <p>Result: {{ counter }}</p>
22      <!-- <input type="text" v-bind:value="name" v-on:input="setName" />
23      <input type="text" v-model="name">
24      <p>Output: {{ name }}</p>
25      <button @:click="resetField">Reset</button>
26      <form v-on:submit.prevent="submitForm">
27        <input type="text">
28        <button>Click Here</button>
29      </form>
30    </section>
31  </body>
32 </html>
33 
```

This two way of binding is so cool, single directive for them to do both.

So, v-model directive combines v-bind and v-on directives.

If we want to dynamically change the value, to do two way binding we can use v-model on input.

In this example, we are using v-bind on input.

## Computed Properties

Similar to `useMemo()` that cache property value and updates when one of its dependency changes, **computed properties** do same job.

Section02 > basics-03-events-starting-code > app.js > app

```

1  const app = Vue.createApp([
2    data() {
3      return {
4        counter: 0,
5        name: "",
6      };
7    },
8    computed: {
9      fullName() {
10        console.log(`Running Again`);
11
12        if (this.name === "") return "";
13
14        return this.name + " " + "Sajwani";
15      },
16    },
17  },
18
19  methods: {
20    increaseCount(num = 1) {
21      this.counter = this.counter + num;
22    },
23
24    decreaseCount(num = 1) {
25      this.counter = this.counter - num;
26    },
27
28    setName(event, lastName) {
29      this.name = event.target.value;
30    },
31
32    submitForm() {},
33
34    resetField() {
35      this.name = "";
36    },
37  },
38]);

```

**createApp also has computed property that works equivalent to useMemo() in react. When you define any function (fullName) in this example then Vue will know its dependencies (name) and reevaluate only when it changes.**

Section02 > basics-03-events-starting-code > index.html > html > body

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Vue Basics</title>
7      <link href="https://fonts.googleapis.com/css2?family=Josefin+Sans:wght@400;700;900" rel="stylesheet" />
8      <link rel="stylesheet" href="styles.css" />
9      <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js" defer></script>
10     <script src="app.js" defer></script>
11   </head>
12   <body>
13     <header>
14       <h1>Vue Events</h1>
15     </header>
16     <section id="events">
17       <h2>Events in Action</h2>
18       <button v-on:click="increaseCount(5)">Add</button>
19       <button v-on:click.right="decreaseCount()">Remove</button>
20       <p v-once>Result: {{ counter }}</p>
21       <p>Result: {{ counter }}</p>
22       <!-- <input type="text" v-bind:value="name" v-on:input="setName($event,lastName)"> -->
23       <input type="text" v-model="name" />
24       <p>Output: {{ fullName }}</p>
25       <button @:click="resetField">Reset</button>
26       <form v-on:submit.prevent="submitForm">
27         <input type="text">
28         <button>Click Here</button>
29       </form>
30     </section>
31   </body>
32 </html>
33
34
35
36
37
38

```

**Although we define function in computed property (fullName()) but we need to pass its pointer (fullName) without parenthesis and Vue JS will do rest.**

## Watchers

Watchers work like `useEffect()` and they must be bound to one of the data properties. Once data property changes, Vue JS runs its linked watcher by passing its old and new value.

File Edit Selection View ... ← → Q VueLearning

app.js U X Section02.md U index.html U

```

1 const app = Vue.createApp([
2   data() {
3     return {
4       counter: 0,
5       name: "",
6     };
7   },
8
9   computed: {
10     fullName() {
11       console.log(`Running Again`);
12
13       if (this.name === "") return "";
14
15       return this.name + " " + "Saiwan";
16     },
17   },
18
19   watch: {
20     counter(value) {
21       if (value > 50) this.counter = 0;
22     },
23   },
24
25   methods: {
26     increaseCount(num = 1) {
27       this.counter = this.counter + num;
28     },
29
30     decreaseCount(num = 1) {
31       this.counter = this.counter - num;
32     },
33
34     setName(event, lastName) {
35       this.name = event.target.value;
36     },
37   },
38 
```

index.html U

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Vue Basics</title>
7     <link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;700&display=swap" rel="stylesheet" />
8     <link rel="stylesheet" href="styles.css" />
9     <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js" defer></script>
10    <script src="app.js" defer></script>
11  </head>
12  <body>
13    <header>
14      <h1>Vue Events</h1>
15    </header>
16    <section id="events">
17      <h2>Events in Action</h2>
18      <button v-on:click="increaseCount(10)">Increase Counter</button>
19      <button v-on:click.right="decreaseCount(10)">Decrease Counter</button>
20      <p>Result: {{ counter }}</p>
21      <p>Result: {{ counter }}</p>
22      <!-- <input type="text" v-bind:value="name" />
23      <input type="text" v-model="name" />
24      <p>Output: {{ fullName }}</p>
25      <button @:click="resetField">Reset</button>
26      <form v-on:submit.prevent="submitForm">
27        <input type="text">
28        <button>Click Here</button>
29      </form>
30    </section>
31  </body>
32  </html>
33
34
35
36
37
38 
```

Ln 18, Col 1 Spaces: 2 UTF-8 LF {} JavaScript Prettier

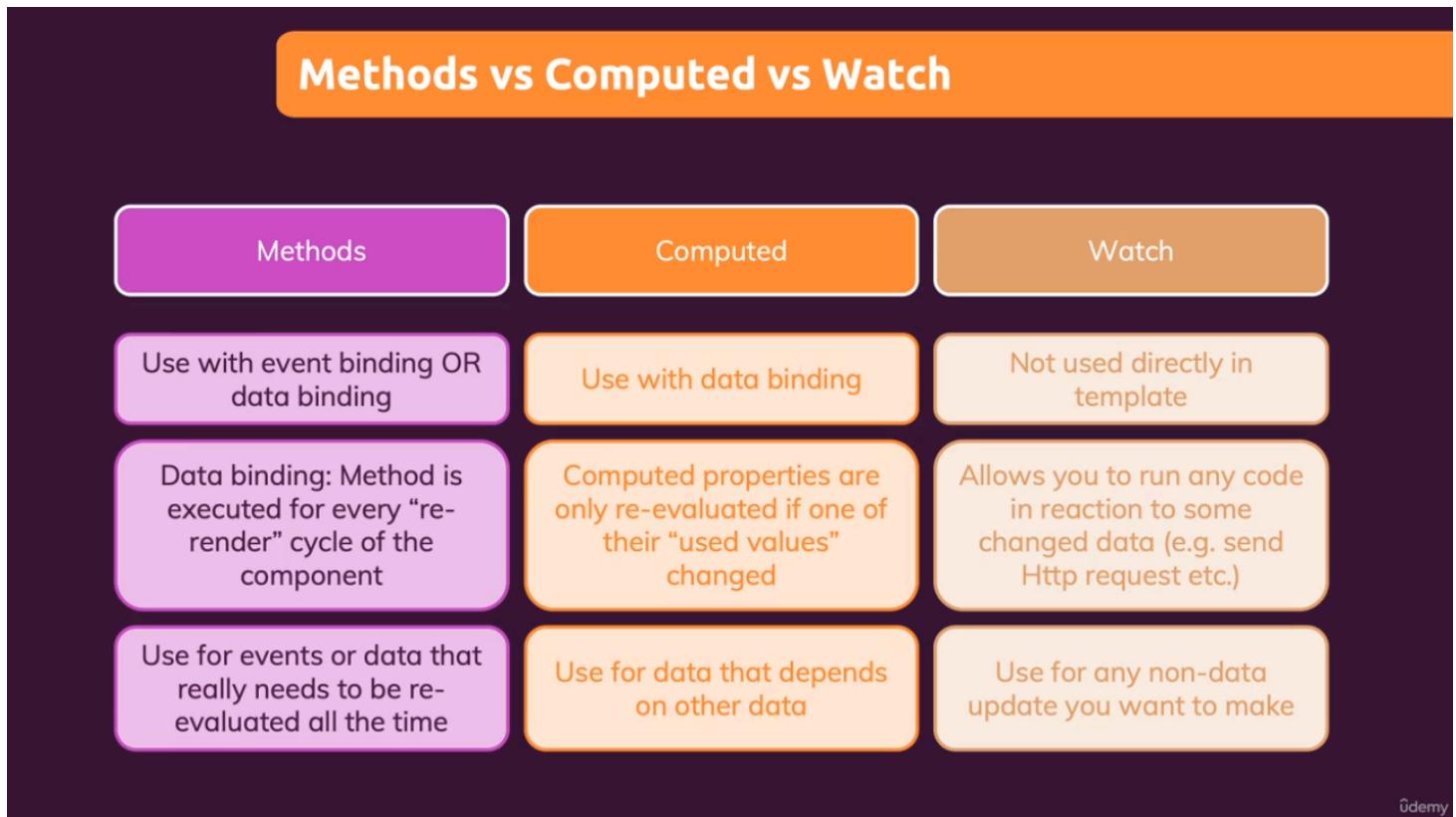
**1**

Watchers property work as `useEffect()` in react that is set in combination with a data property and when it change it get reevaluated and Vue JS passes its newValue and oldValue

**2**

computed Value as compare to watchers doesn't bound to data property.

# Methods Vs. Computed Properties Vs. Watchers



## Shorthand for `v-bind` and `v-on`

You can use `@` as a shorthand for `v-on` and `:` as a shorthand for `v-bind`

## Dynamic Styling

By binding `class` or `style` attributes we can do dynamic styling. Vue JS provides object syntax in which **object keys are class names** and values are `true/false`

The screenshot shows a development setup for a Vue.js application. On the left, the code editor displays `app.js` and `index.html`. The `index.html` file contains a template with dynamic styling logic. Red annotations highlight two specific features:

- An annotation labeled "1" points to the inline style binding `:style="borderColor: boxASelected ? 'red' : 'gray'"`, with the text: "Vue JS offers a object syntax from which we can dynamically change inline style. We need to bind attribute using v-bind or : shorthand".
- An annotation labeled "2" points to the class binding `:class="{active: boxBSelected}"`, with the text: "We can also add or remove classes dynamically by binding class attribute".

The browser preview on the right shows three red rectangular boxes with dashed borders, representing the state where `boxASelected` is true. The developer tools panel at the bottom shows a message about running a development build of Vue.

```
app.js
1  Vue.createApp({
2    data() {
3      return {
4        boxASelected: false,
5        boxBSelected: false,
6        boxCSelected: false,
7      };
8    },
9
10   methods: {
11     boxSelected(selector) {
12       if (selector === "A") {
13         this.boxASelected = !this
14       } else if (selector === "B") {
15         this.boxBSelected = !this
16       } else {
17         this.boxCSelected = !this
18       }
19     },
20   },
21 }).mount("#styling");
22 
```

```
index.html
1  DOCTYPE html>
2  html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Vue Basics</title>
7      <link href="https://fonts.googleapis.com/css?family=Jost:wght@400;700&display=swap" rel="stylesheet">
8
9      <link rel="stylesheet" href="styles.css" />
10     <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
11     <script src="app.js" defer></script>
12   </head>
13   <body>
14     <header>
15       <h1>Vue Dynamic Styling</h1>
16     </header>
17     <section id="styling">
18       <div :style="borderColor: boxASelected ? 'red' : 'gray'" @:click="boxSelected('A')"
19         class="demo">
20         <div class="demo"
21           :class="{active: boxBSelected}" @:click="boxSelected('B')"
22           <div class="demo"
23             :class="{active: boxCSelected}" @:click="boxSelected('C')"
24           </div>
25         </div>
26       </div>
27     </section>
28   </body>
29   <html>
30 
```

Vue JS offers a object syntax from which we can dynamically change inline style. We need to bind attribute using v-bind or : shorthand 1

We can also add or remove classes dynamically by binding class attribute 2

You are running a development build of Vue. vue.global.js:12572  
Make sure to use the production build (\*.prod.js) when deploying for production.

# Assignment 3

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vue Basics</title>
    <link href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap" rel="stylesheet"
  />
    <link rel="stylesheet" href="styles.css" />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
    <script src="app.js" defer></script>
  </head>
  <body>
    <header>
      <h1>Vue Styling</h1>
    </header>
    <section id="assignment">
      <!-- 1) Fetch the user input and use it as a CSS class -->
      <!-- The entered class should be added to the below paragraph -->
      <input type="text" @keyup.enter="updateClasses" />
      <!-- (available classes: "user1", "user2") -->
      <p :class="[userClasses, toggle]">
        Style me!
      </p>
      <button @click="togglePara">Toggle Paragraph</button>
      <!-- 2) Use the "visible" and "hidden" classes to show/ hide the above paragraph -->
      <!-- Clicking the button should toggle between the two options -->

      <!-- 3) Add dynamic inline styling to the below paragraph and let the user enter a background color -->
      <input type="text" @input="changeBgColor" />
      <p :style="{backgroundColor: bgColor}">Style me inline!</p>
    </section>
  </body>
</html>
```

```
Vue.createApp({
  data: () => {
    return {
      userClasses: "",
      isDisplay: true,
      bgColor: "white",
    };
  },
  computed: {
    toggle() {
      return this.isDisplay ? "visible" : "hidden";
    },
  },
  methods: {
    updateClasses(event) {
      this.userClasses = event.target.value;
    },
    togglePara() {
      this.isDisplay = !this.isDisplay;
    },
    changeBgColor(event) {
      this.bgColor = event.target.value;
    },
  },
}).mount("#assignment");
```

# Summary

## Summary

### DOM & Templates

Vue can be used to define the goal instead of the steps (→ **declarative approach**)

**Connect Vue to HTML via "mount":**  
Vue then renders the real DOM based on the connected template

### Reactivity

Vue updates the real DOM for you when bound data changes

**Computed properties** and **watchers** allow you to react to data changes

### Data & Event Bindings

You can bind data via interpolation (`{{ }}`) or the **v-bind** ("`:`") directive

You listen for events via **v-on** ("`@`")

### Styling

Dynamic CSS class and inline style bindings are supported by Vue

Vue offers multiple **special syntaxes** (object-based, array-based) for efficient bindings