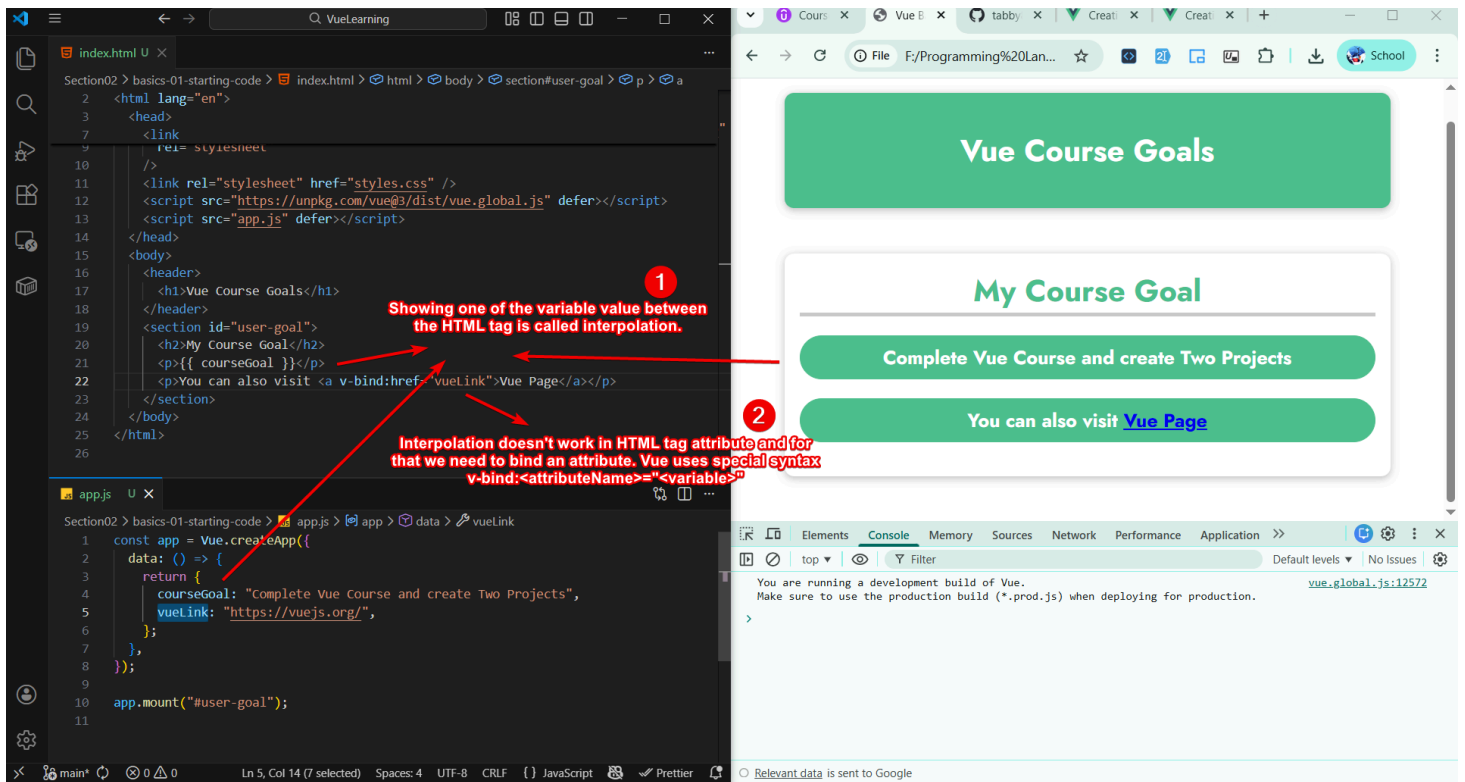


# Section 02 - Vue Core Concepts

## Using Interpolation and v-bind



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");

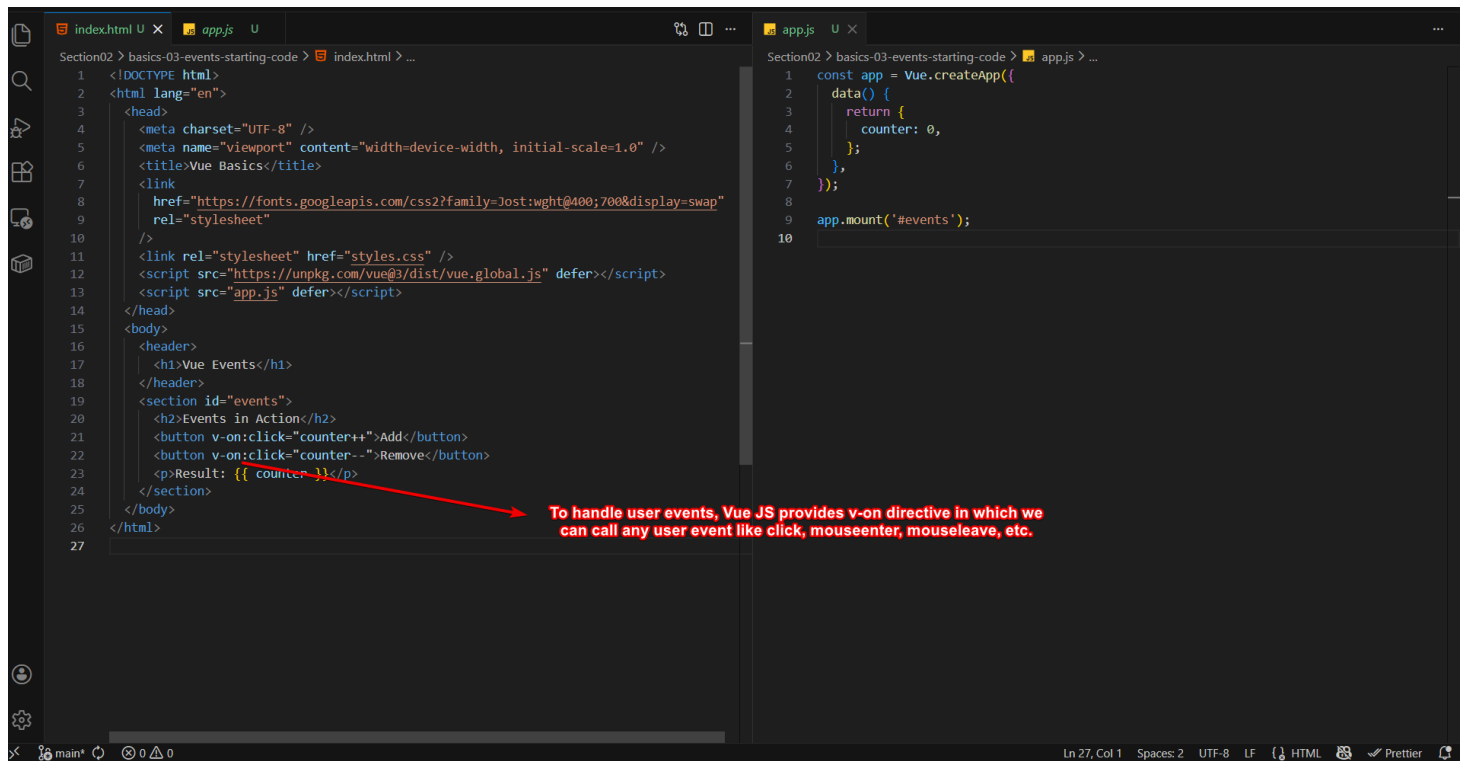
```

```

<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



```
index.html U X app.js U
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=1.0" />
6   <title>Vue Basics</title>
7   <link
8     href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap"
9     rel="stylesheet"
10  />
11  <link rel="stylesheet" href="styles.css" />
12  <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
13  <script src="app.js" defer></script>
14 </head>
15 <body>
16   <header>
17     <h1>Vue Events</h1>
18   </header>
19   <section id="events">
20     <h2>Events in Action</h2>
21     <button v-on:click="counter++">Add</button>
22     <button v-on:click="counter--">Remove</button>
23     <p>Result: {{ counter }}</p>
24   </section>
25 </body>
26 </html>
27

app.js U X
Section02 > basics-03-events-starting-code > app.js > ...
1 const app = Vue.createApp({
2   data() {
3     return {
4       counter: 0,
5     };
6   },
7 });
8
9 app.mount('#events');
10
```

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;
    },

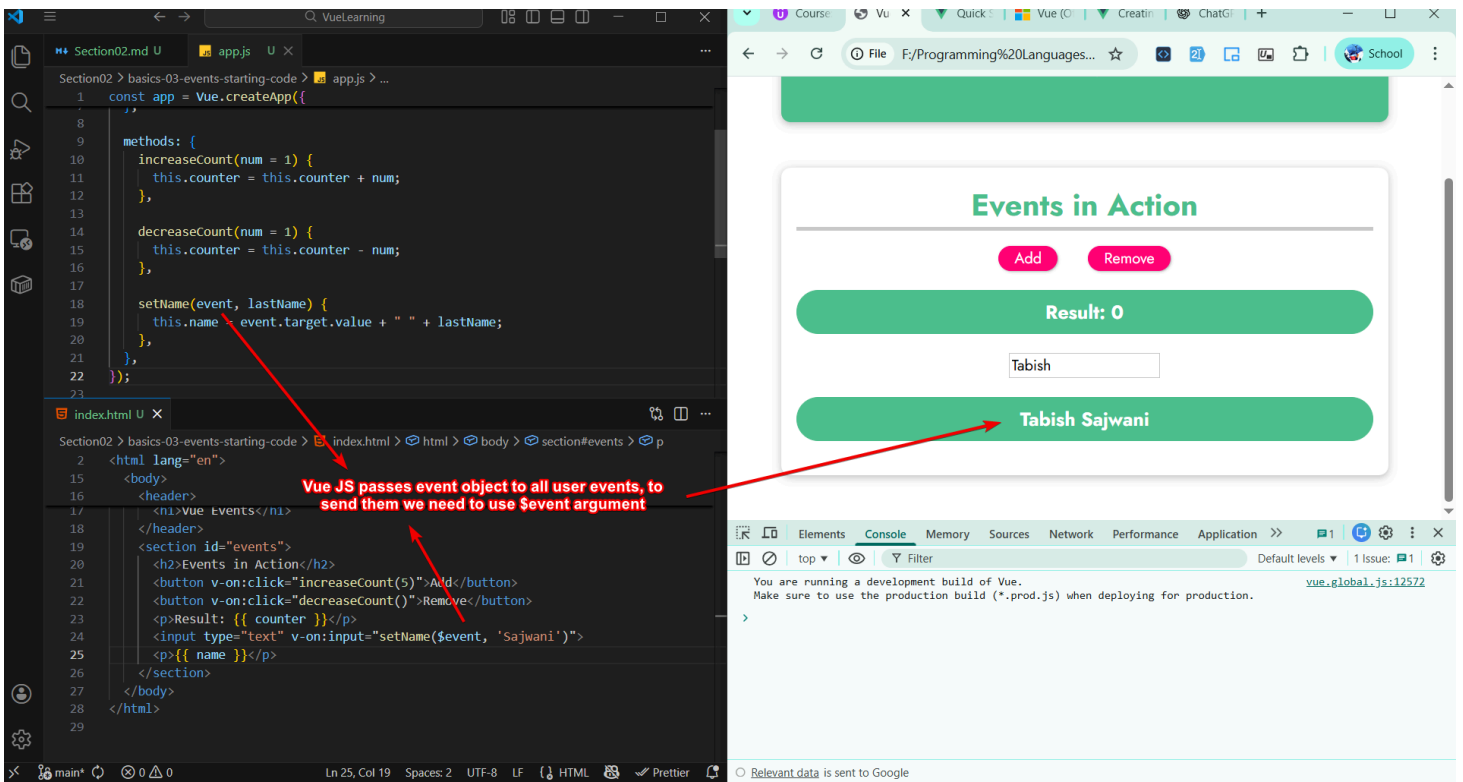
    decreaseCount(num = 1) {
      this.counter = this.counter - num;
    },
  },
});

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>
```

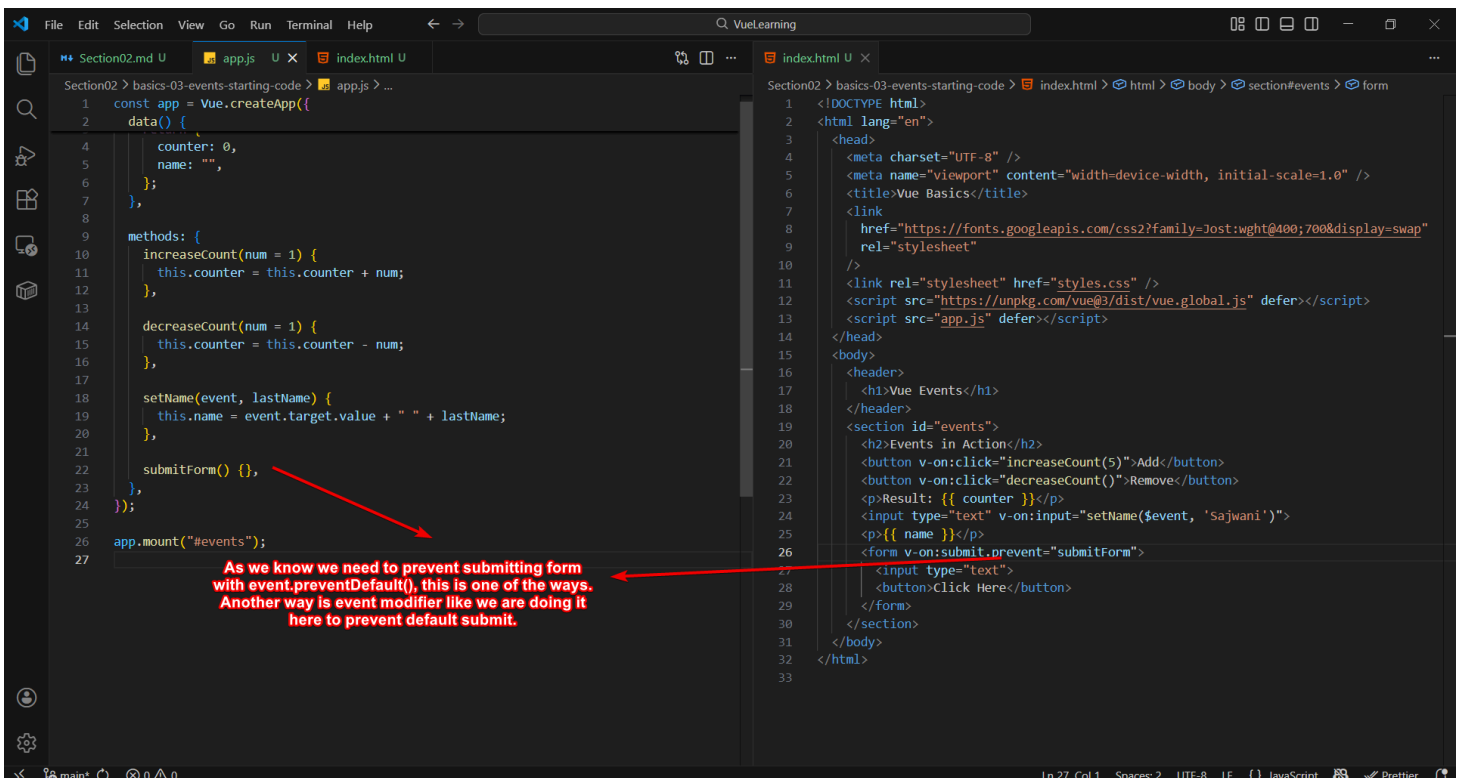
## Passing Event Object

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.



## Event Modifier

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



# 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 displays a Vue.js development environment with two main panels: a code editor on the left and a browser preview on the right.

**Code Editor (Left Panel):**

- index.html:** Contains HTML structure. Line 21 shows `<p>{{ courseGoal }}</p>`. Line 22 shows `<a v-bind:href="vueLink">Vue Page</a></p>`. A red arrow points from the `vueLink` variable in the `data` object to the `vueLink` attribute in the `v-bind:href` directive.
- app.js:** Contains the Vue application logic. Line 4 shows `courseGoal: "Complete Vue Course and create Two Projects",`. Line 5 shows `vueLink: "https://vuejs.org/",`.

**Browser Preview (Right Panel):**

- The rendered UI shows a green header "Vue Course Goals".
- Below it is a white box titled "My Course Goal" containing a green button labeled "Complete Vue Course and create Two Projects".
- At the bottom, a green button labeled "You can also visit [Vue Page](#)".

**Annotations:**

- 1:** A red circle and arrow pointing to the `<p>{{ courseGoal }}</p>` tag with the text: "Showing one of the variable value between the HTML tag is called interpolation."
- 2:** A red circle and arrow pointing to the `v-bind:href="vueLink"` directive 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>".

**Console (Bottom Right):**

- Shows a message: "You are running a development build of Vue. Make sure to use the production build (\*.prod.js) when deploying for production."
- A link `vue.global.js:12572` is visible.

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

t-model is syntax sugar for

- reading value
- updating state when input change



```
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 + lastName;
17     },
18     submitForm() {},
19     resetField() {
20       this.name = "";
21     },
22   },
23 });
24 app.mount("#events");
```

```
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
8       href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700"
9       rel="stylesheet" />
10    </head>
11    <body>
12      <header>
13        <h1>Vue Events</h1>
14      </header>
15      <section id="events">
16        <h2>Events in Action</h2>
17        <button v-on:click="increaseCount(5)">Add</button>
18        <button v-on:click.right="decreaseCount()">Remove</button>
19        <p v-once>Result: {{ counter }}</p>
20        <p>Result: {{ counter }}</p>
21        <!-- <input type="text" v-bind:value="name" v-on:input="setName()" />
22        <input type="text" v-model="name" />
23        <p>Output: {{ name }}</p>
24        <button @:click="resetField">Reset</button>
25        <form v-on:submit.prevent="submitForm">
26          <input type="text" />
27          <button>Click Here</button>
28        </form>
29      </section>
30    </body>
31  </html>
```

This two way of binding is so common that we have a single directive for them.

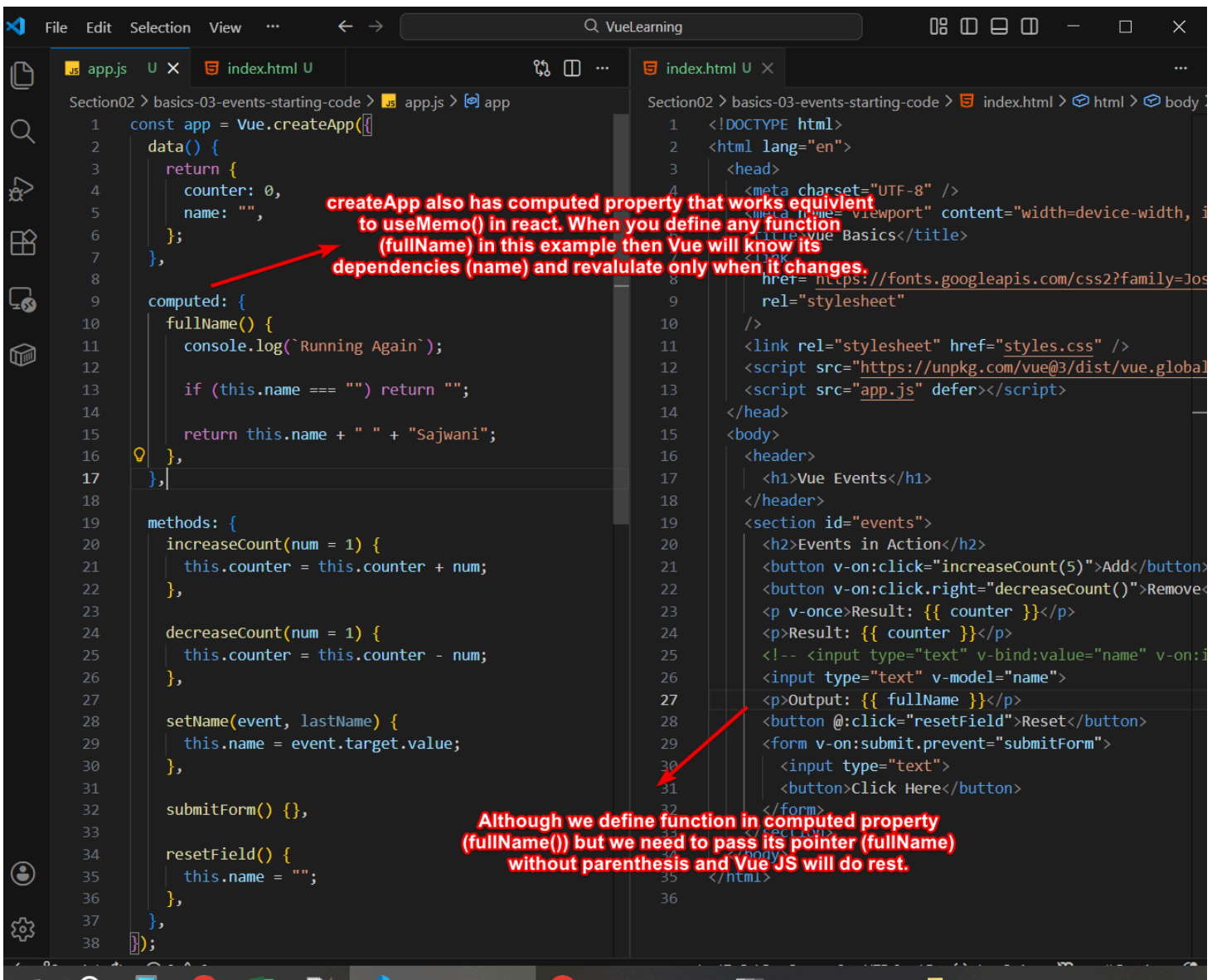
So, v-model directive combines v-bind and v-on directives.

If we want to dynamically change the value of an input field to do two way binding.

In this example, we are using v-model in input field.

## Computed Properties

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



## 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 ... VueLearning

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

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      if (this.name === "") return "";
12      return this.name + " " + "Saiwan1";
13    },
14  },
15  watch: {
16    counter(value) {
17      if (value > 50) this.counter = 0;
18    },
19  },
20  methods: {
21    increaseCount(num = 1) {
22      this.counter = this.counter + num;
23    },
24    decreaseCount(num = 1) {
25      this.counter = this.counter - num;
26    },
27    setName(event, lastName) {
28      this.name = event.target.value;
29    },
30  },
31  submitForm() {
32  }
33 })
```

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 new value and old value**

2

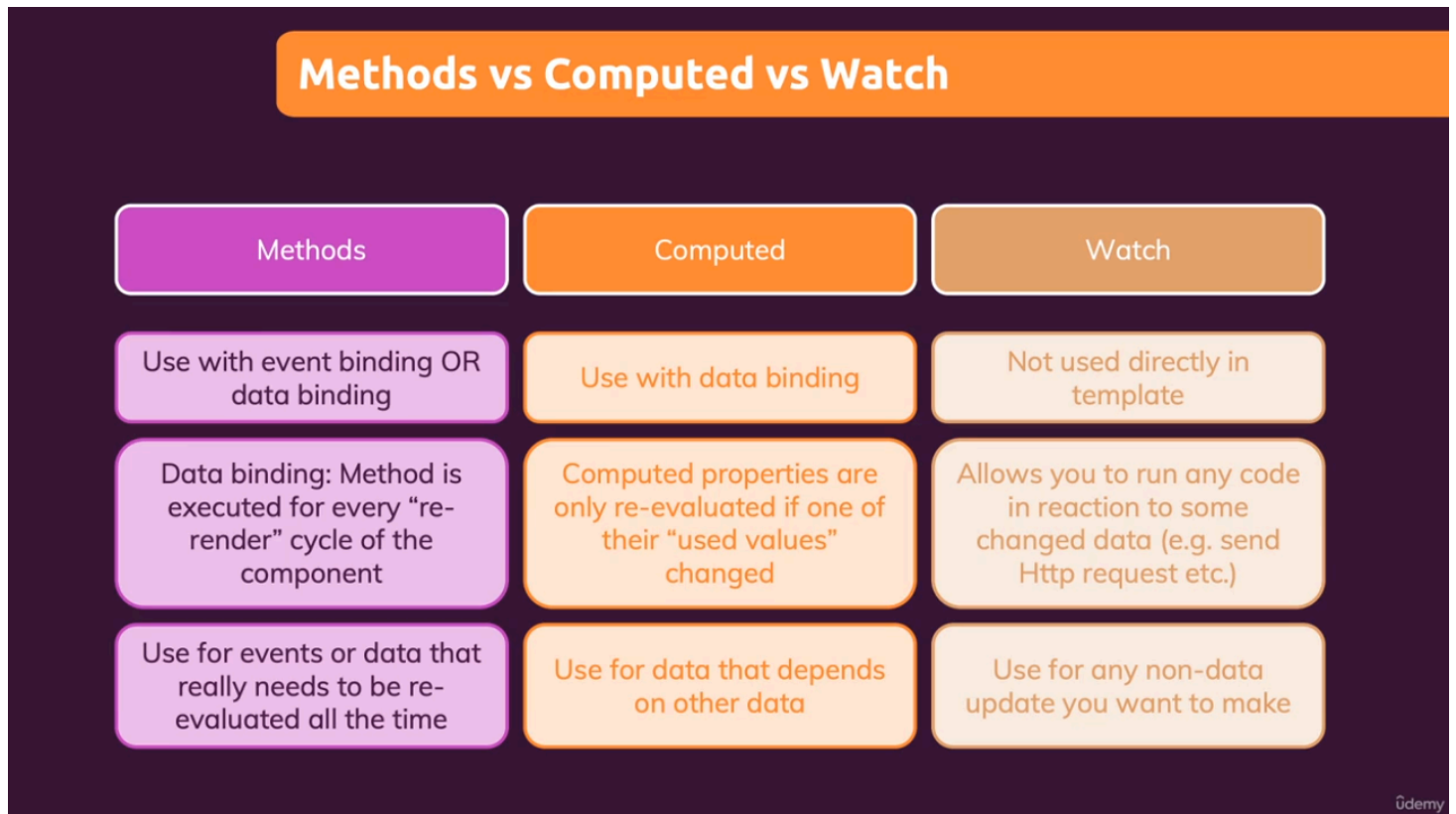
**computed Value as compare to watchers doesn't bound to data property.**

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

```
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
8     href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=block"
9     rel="stylesheet"
10  />
11   <link rel="stylesheet" href="styles.css" />
12   <script src="https://unpkg.com/vue@3/dist/vue.global.js" />
13   <script src="app.js" defer></script>
14 </head>
15 <body>
16   <div>
17     <h1>Vue Events</h1>
18   </div>
19   <div id="events">
20     <h2>Events in Action</h2>
21     <button v-on:click="increaseCount(10)">Increase Count</button>
22     <button v-on:click="decreaseCount(10)">Decrease Count</button>
23     <p v-once>Result: {{ counter }}</p>
24     <p>Result: {{ counter }}</p>
25     <!-- <input type="text" v-bind:value="name" />
26     <input type="text" v-model="name" />
27     <p>Output: {{ fullName }}</p>
28     <button @:click="resetField">Reset</button>
29     <form v-on:submit.prevent="submitForm">
30       <input type="text" />
31       <button>Click Here</button>
32     </form>
33   </div>
34 </body>
35 </html>
36
```

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

# 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`

Vue.js development environment showing code for styling and a browser preview.

**app.js**

```
1 Vue.createApp({
2   data: () => {
3     return {
4       boxASelected: false,
5       boxBSelected: false,
6       boxCSelected: false,
7     };
8   },
9   methods: {
10     boxSelected(selector) {
11       if (selector === "A") {
12         this.boxASelected = !this
13         this.boxASelected = !this
14       } else if (selector === "B") {
15         this.boxBSelected = !this
16         this.boxBSelected = !this
17       } else {
18         this.boxCSelected = !this
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
8     href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap"
9     rel="stylesheet"
10  />
11   <link rel="stylesheet" href="styles.css" />
12   <script src="https://unpkg.com/vue@3/dist/vue.global.js" defer></script>
13   <script src="app.js" defer></script>
14 </head>
15 <body>
16   <div>
17     <h1>Vue Dynamic Styling</h1>
18   </div>
19   <div id="styling">
20     <div :style="{bordercolor: boxASelected ? 'red' : 'gray'}"
21       @:click="boxSelected('A')"
22       class="demo"></div>
23     <div class="demo"
24       :class="{active: boxBSelected}" @:click="boxSelected('B')" ></div>
25     <div class="demo"
26       :class="{active: boxCSelected}" @:click="boxSelected('C')" ></div>
27   </div>
28 </body>
29 </html>
30
```

**styles.css**

```
1 .demo {
2   border: 1px solid gray;
3   padding: 10px;
4   margin: 10px;
5 }
6 .demo.active {
7   background-color: #f0f0f0;
8 }
9
```

**Browser Preview:**

The browser shows a development build of Vue. The console displays:

```
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.
```

**Annotations:**

- 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)

# 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