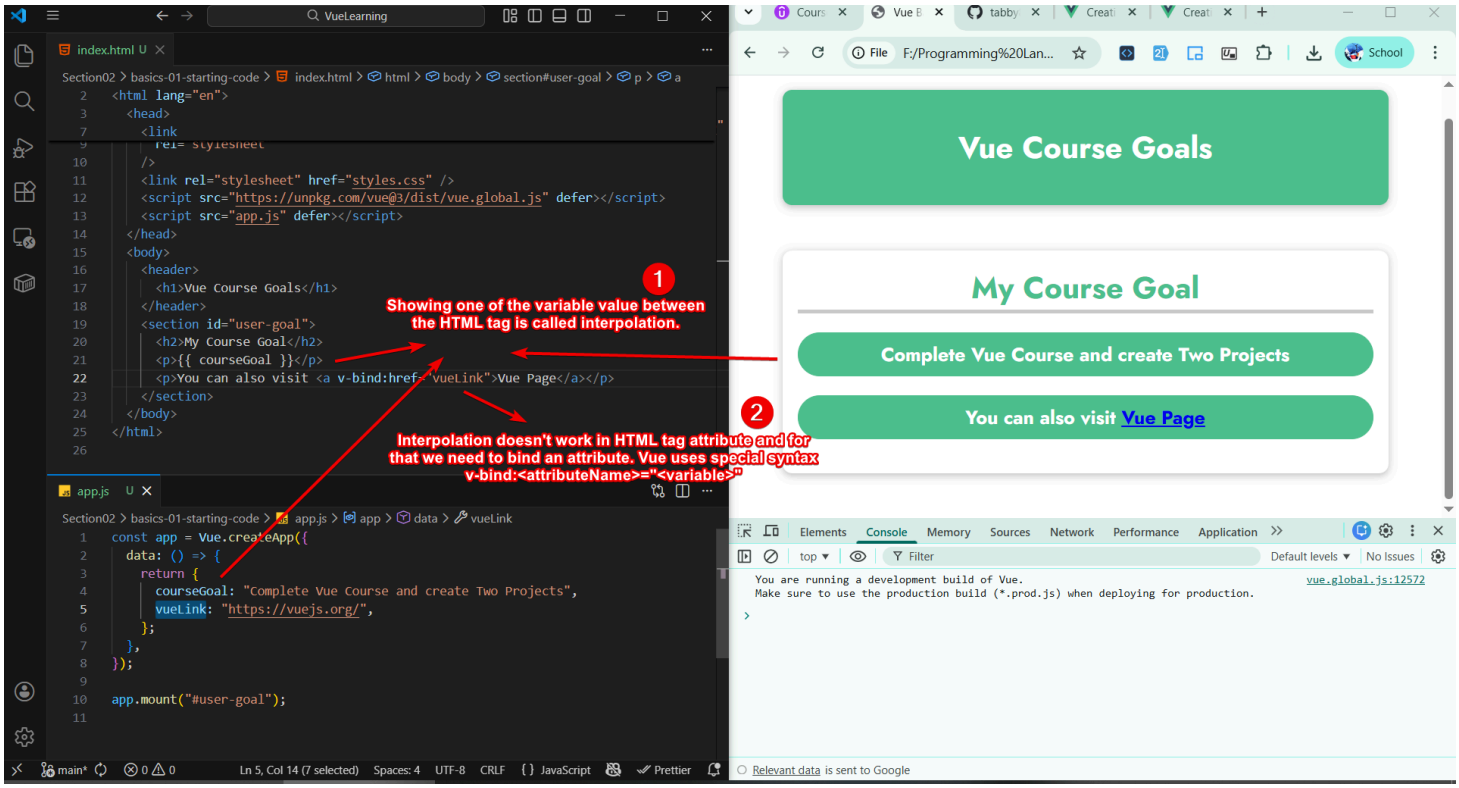
Section 02 - Vue Core Concepts

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 varibles 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");
```

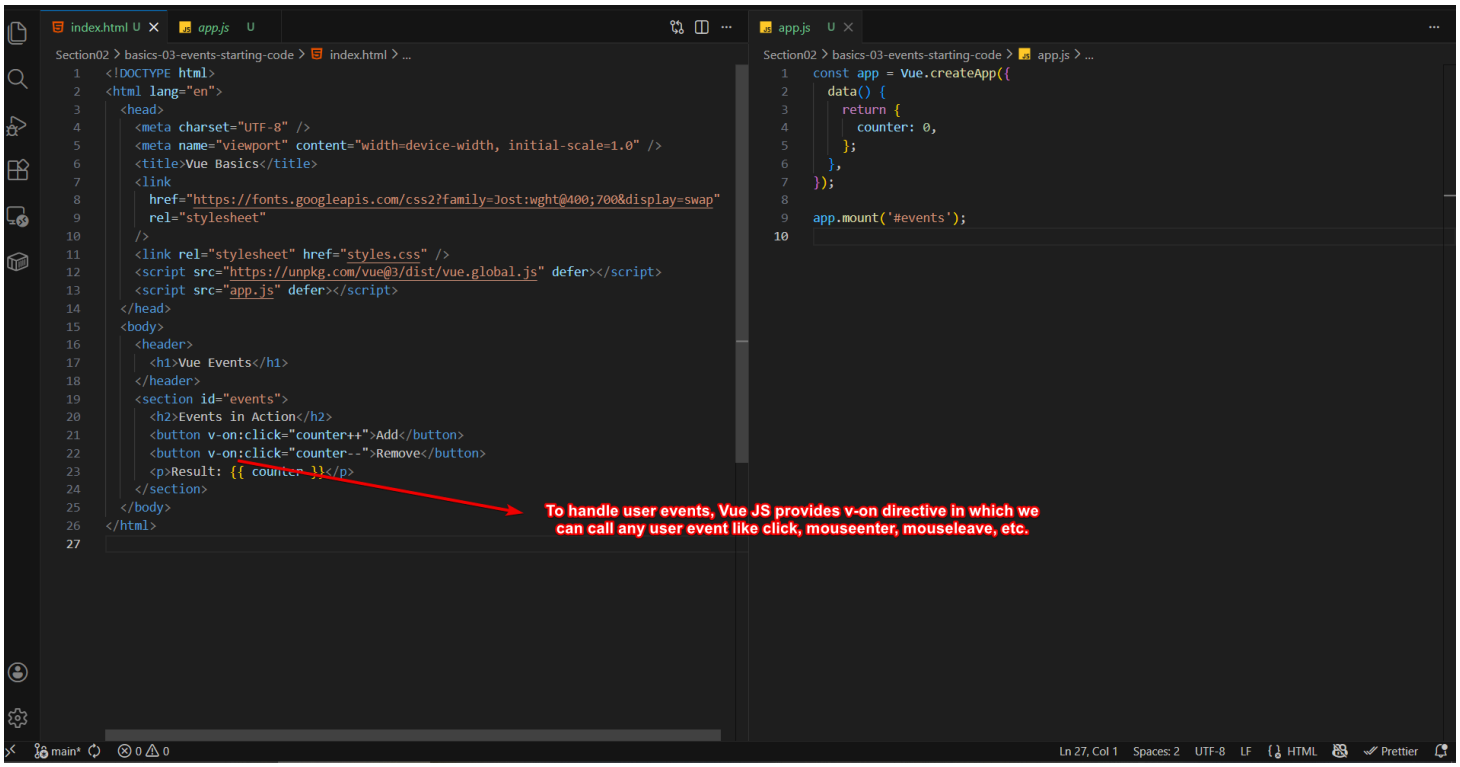
Rendering Raw HTML

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



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

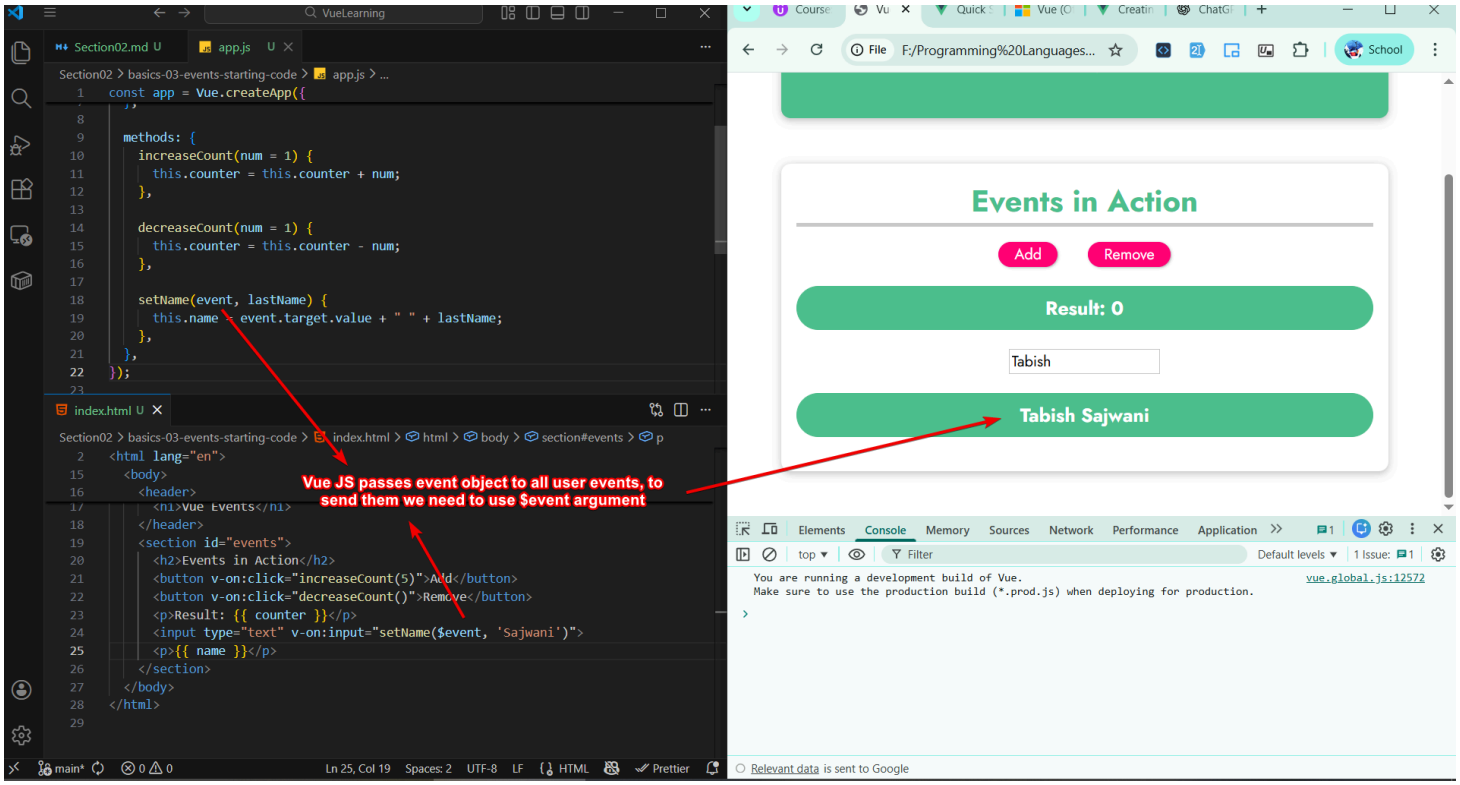
It is good practice to pass `callback functions` in event listeners

```
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>
```

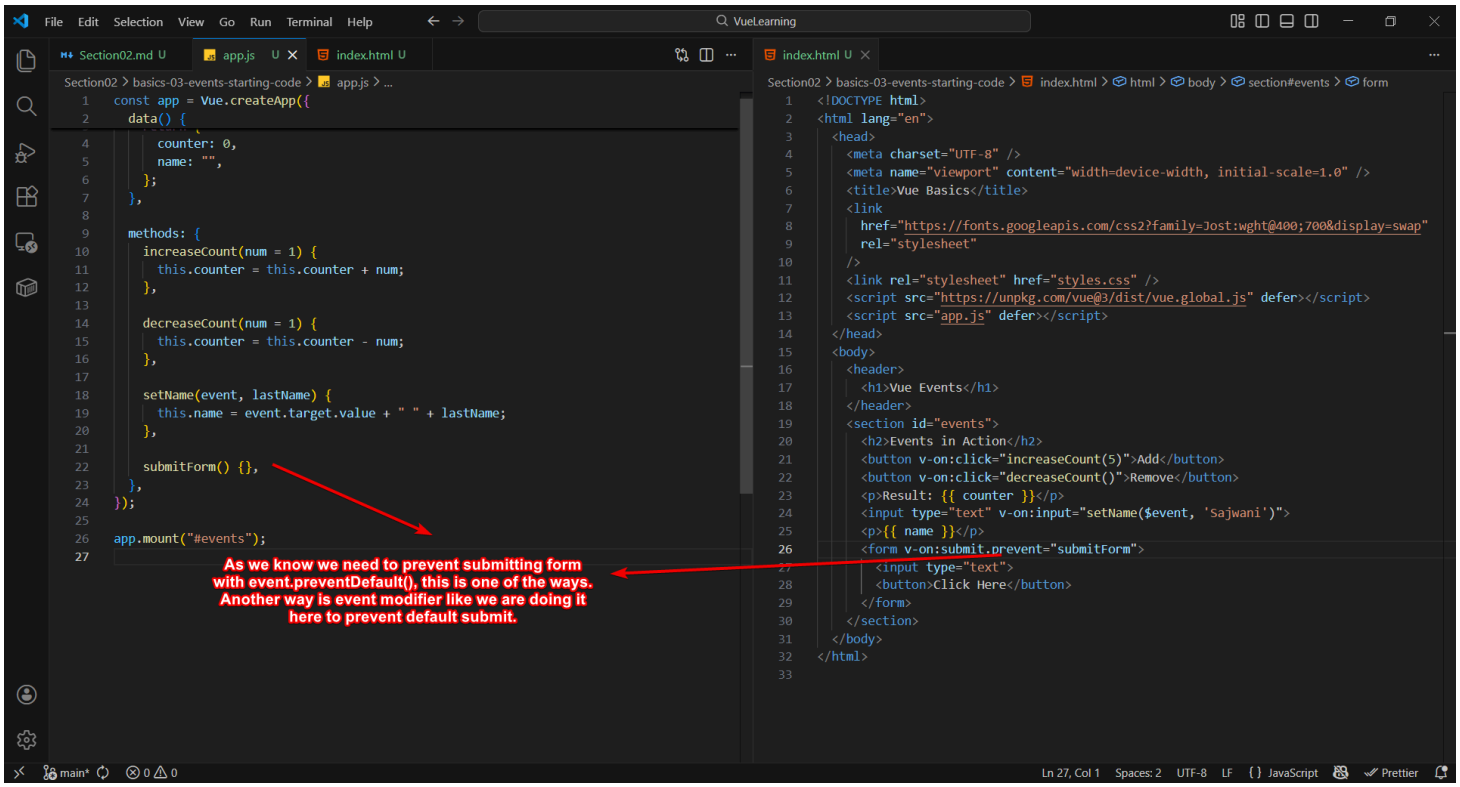
Passing Event Object

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.



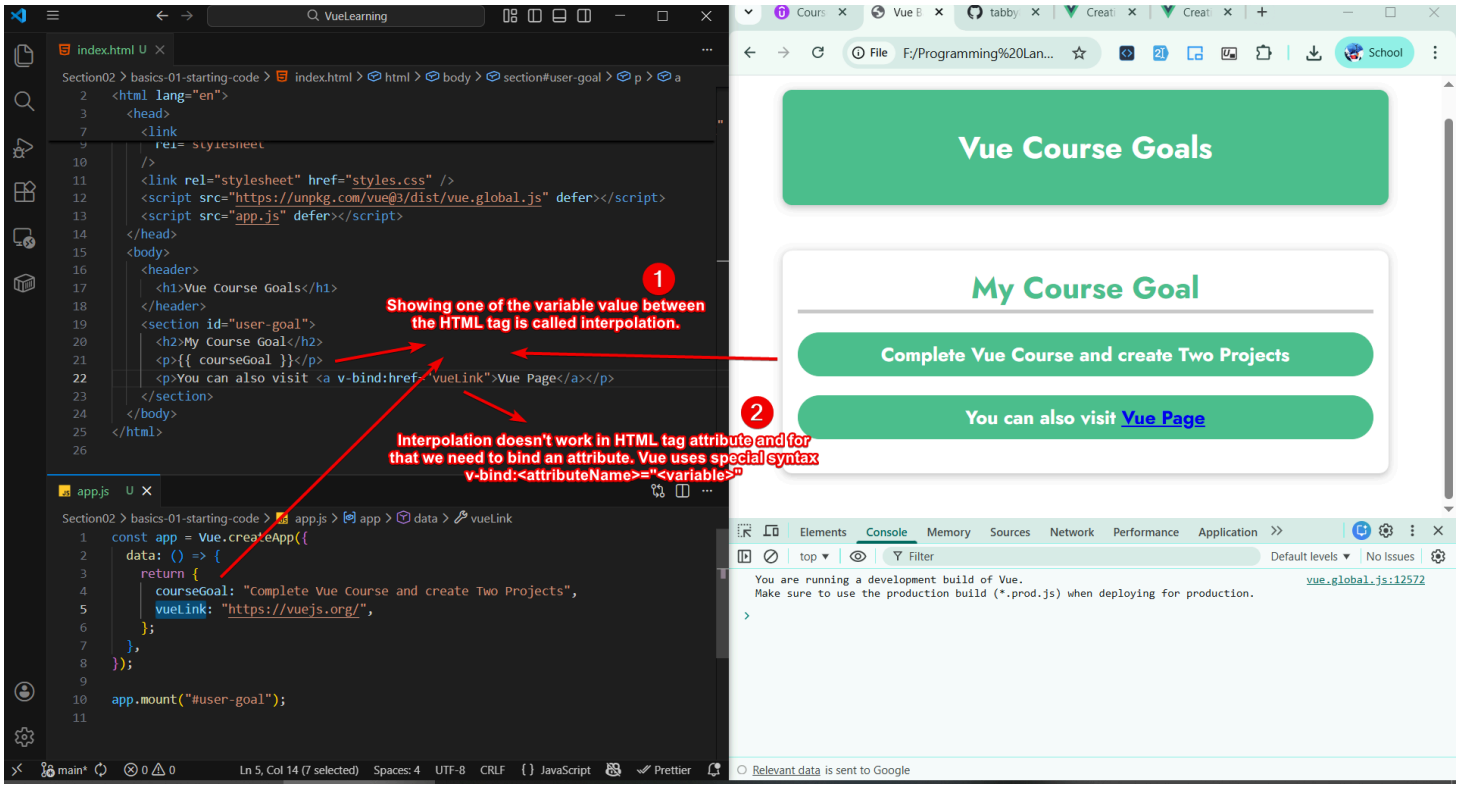
Calculating Value Once

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`

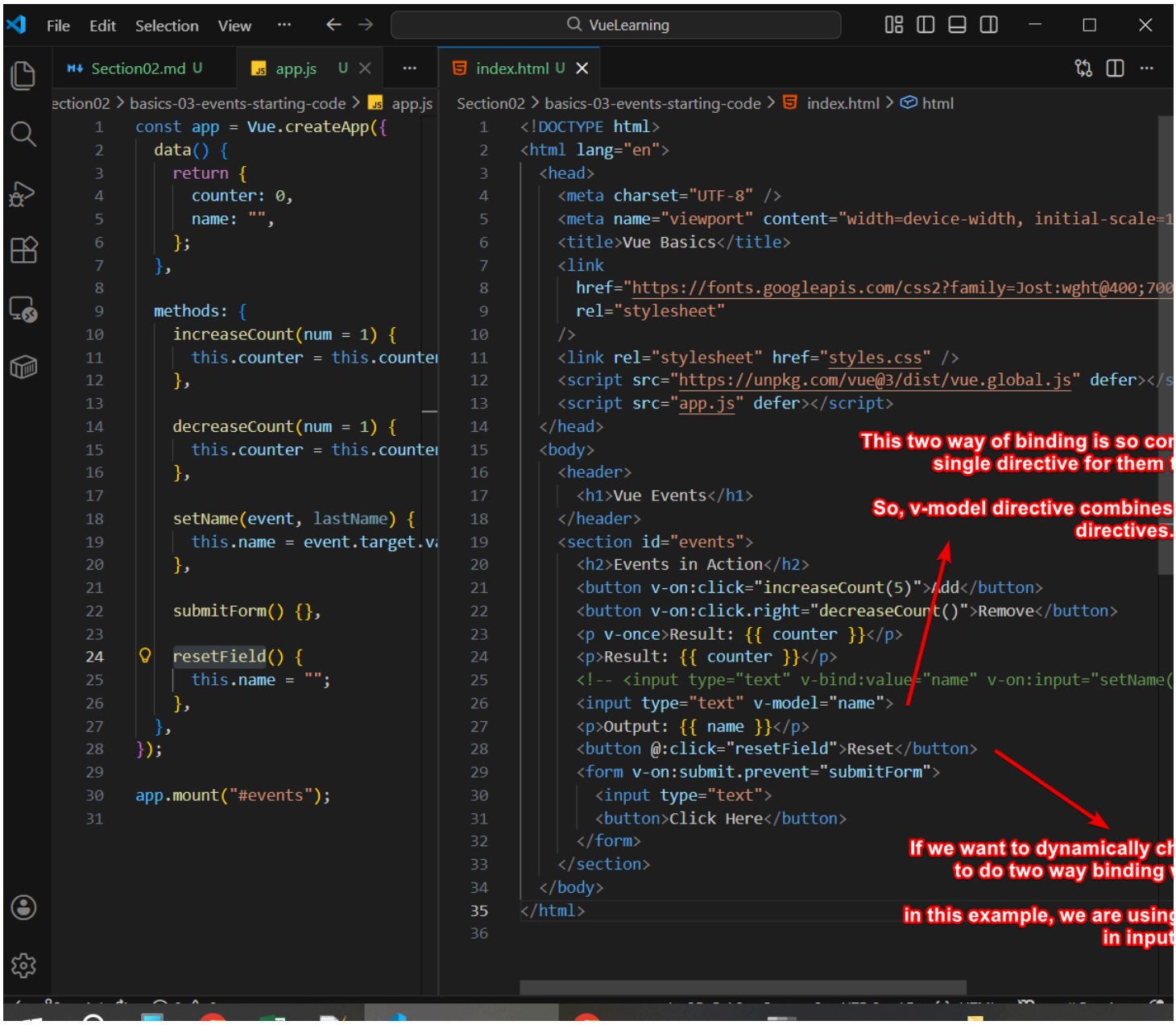
You can bind any component to change field dynamically using `v-bind` directive



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

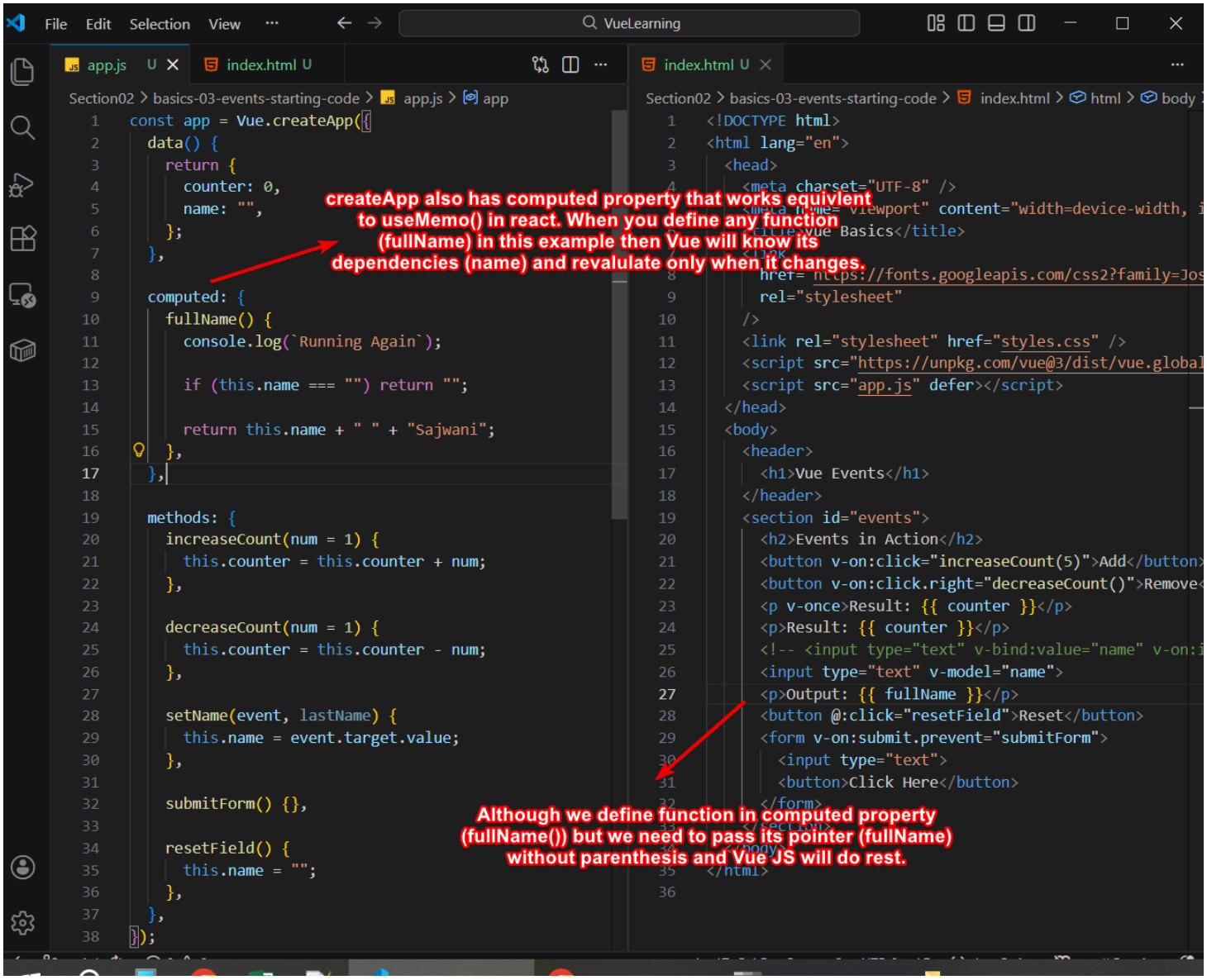
t-model is syntax sugar for

- reading value
- updating state when input change



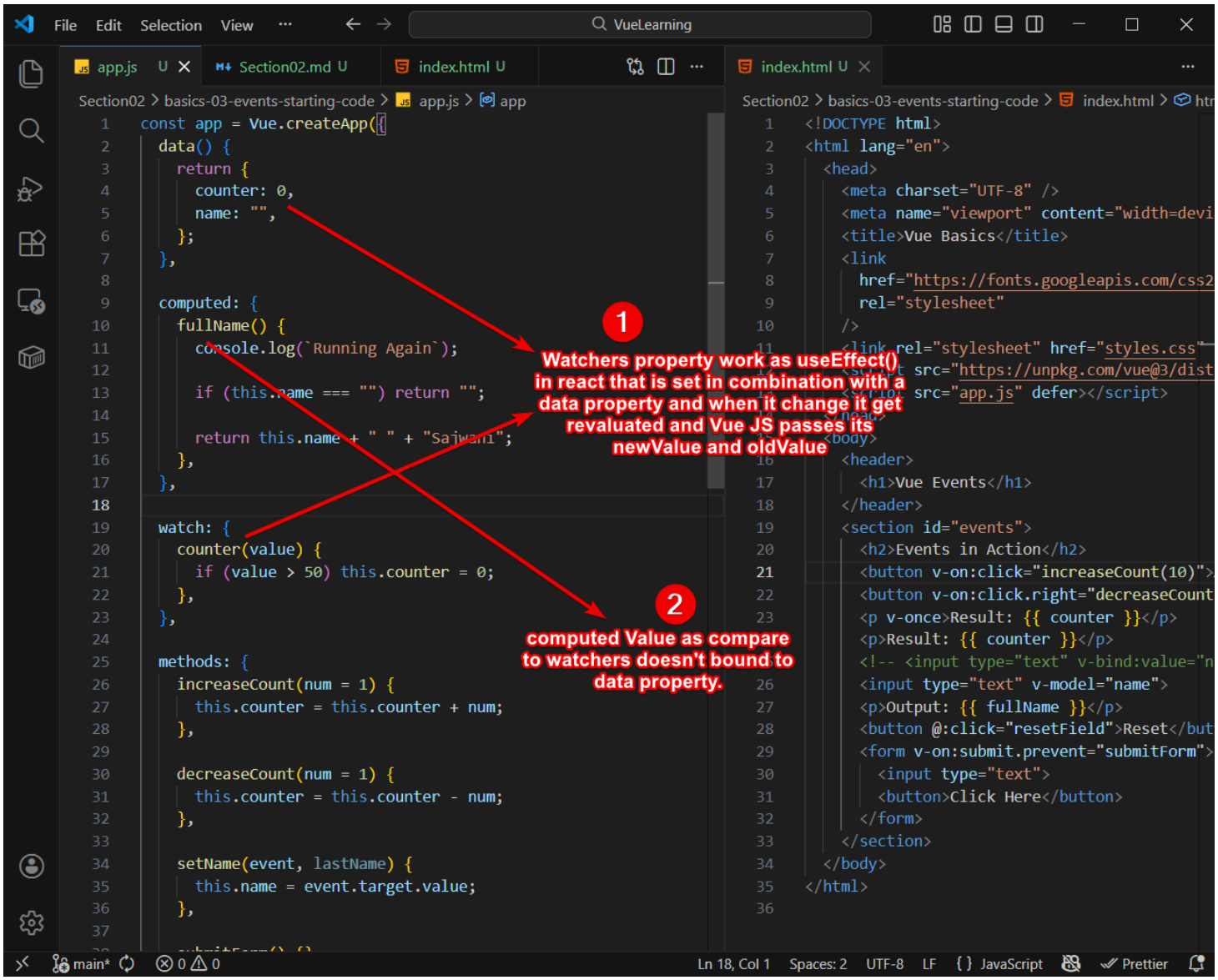
Computed Properties

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

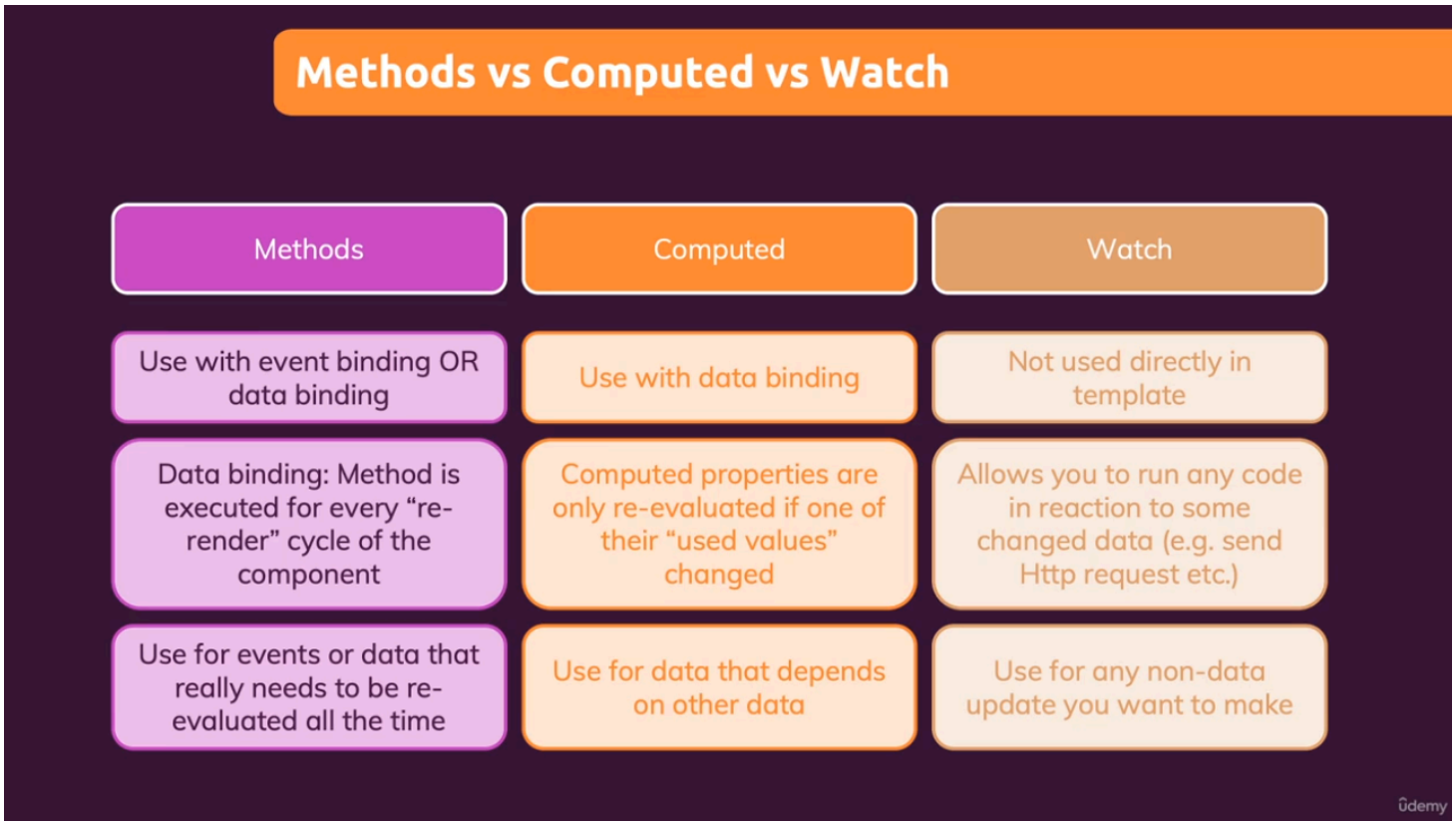


Watchers

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
Section02.md U
index.html U
Section02 > basics-03-events-starting-code > app.js > app
const app = Vue.createApp({
data() {
return {
counter: 0,
name: "",
};
},
computed: {
fullName() {
console.log(`Running Again`);
if (this.name === "") return "";
return this.name + " " + "Sajwani";
},
},
watch: {
counter(value) {
if (value > 50) this.counter = 0;
},
},
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;
},
1
Watchers property work as useEffect() in react that is set in combination with a data property and when it change it get revaluated and Vue JS passes its newValue and oldValue
2
computed Value as compare to watchers doesn't bound to data property.
index.html U
Section02 > basics-03-events-starting-code > index.html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=devi
<title>Vue Basics</title>
<link
href="https://fonts.googleapis.com/css2
rel="stylesheet"
/>
<link rel="stylesheet" href="styles.css"
<script src="https://unpkg.com/vue@3/dist
<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(10)">
<button v-on:click.right="decreaseCount
<p v-once>Result: {{ counter }}
Result: {{ counter }}
<!-- <input type="text" v-bind:value="n
<input type="text" v-model="name">
Output: {{ fullName }}
<button @:click="resetField">Reset</but
<form v-on:submit.prevent="submitForm">
<input type="text">
<button>Click Here</button>
</form>
</section>
</body>
</html>
main*
0 0
Ln 18, Col 1
Spaces: 2
UTF-8
LF
{} JavaScript
Prettier

Methods Vs. Computed Properties Vs. Watchers

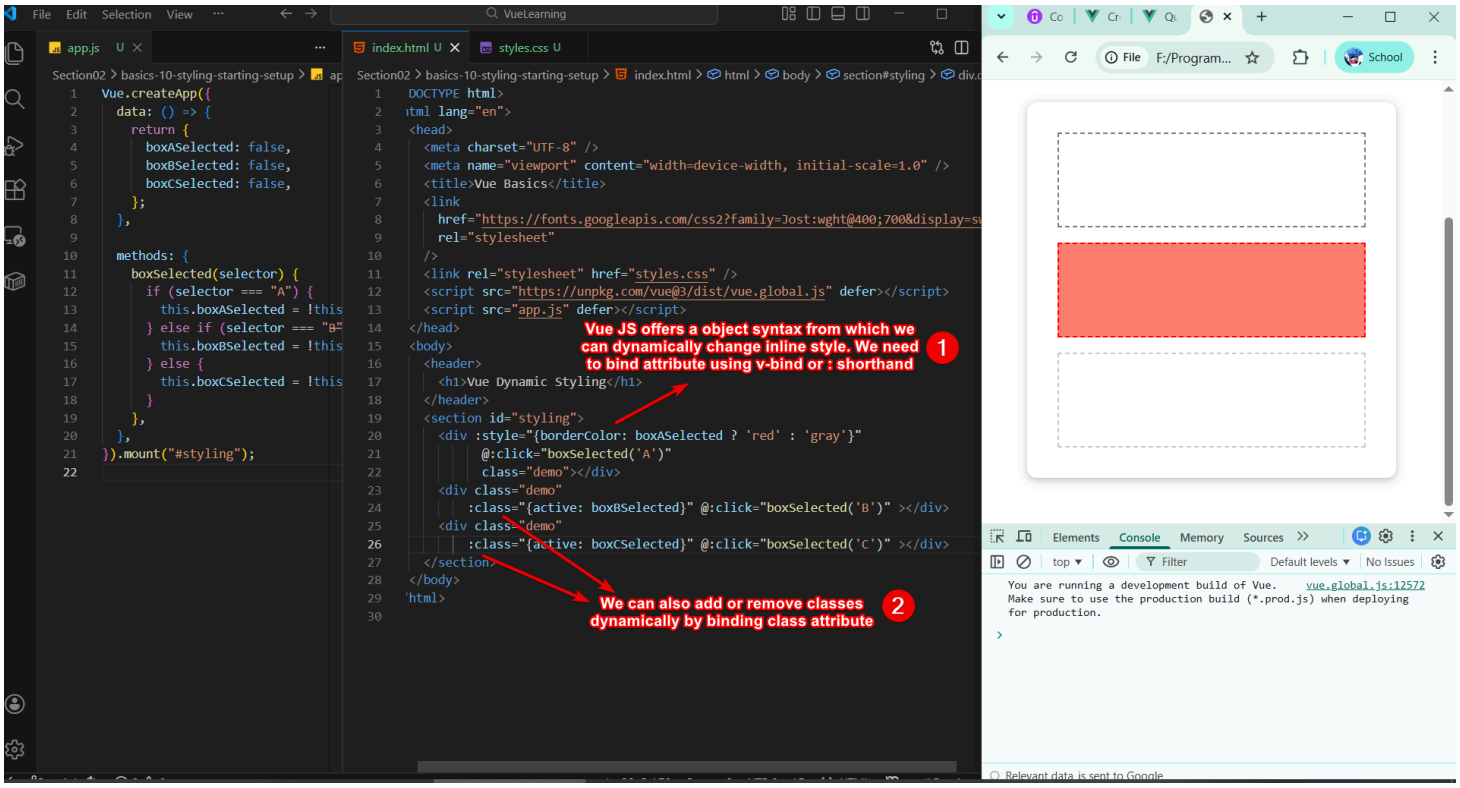


Shorthand for `v-bind` and `v-on`

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

Dynamic Styling

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

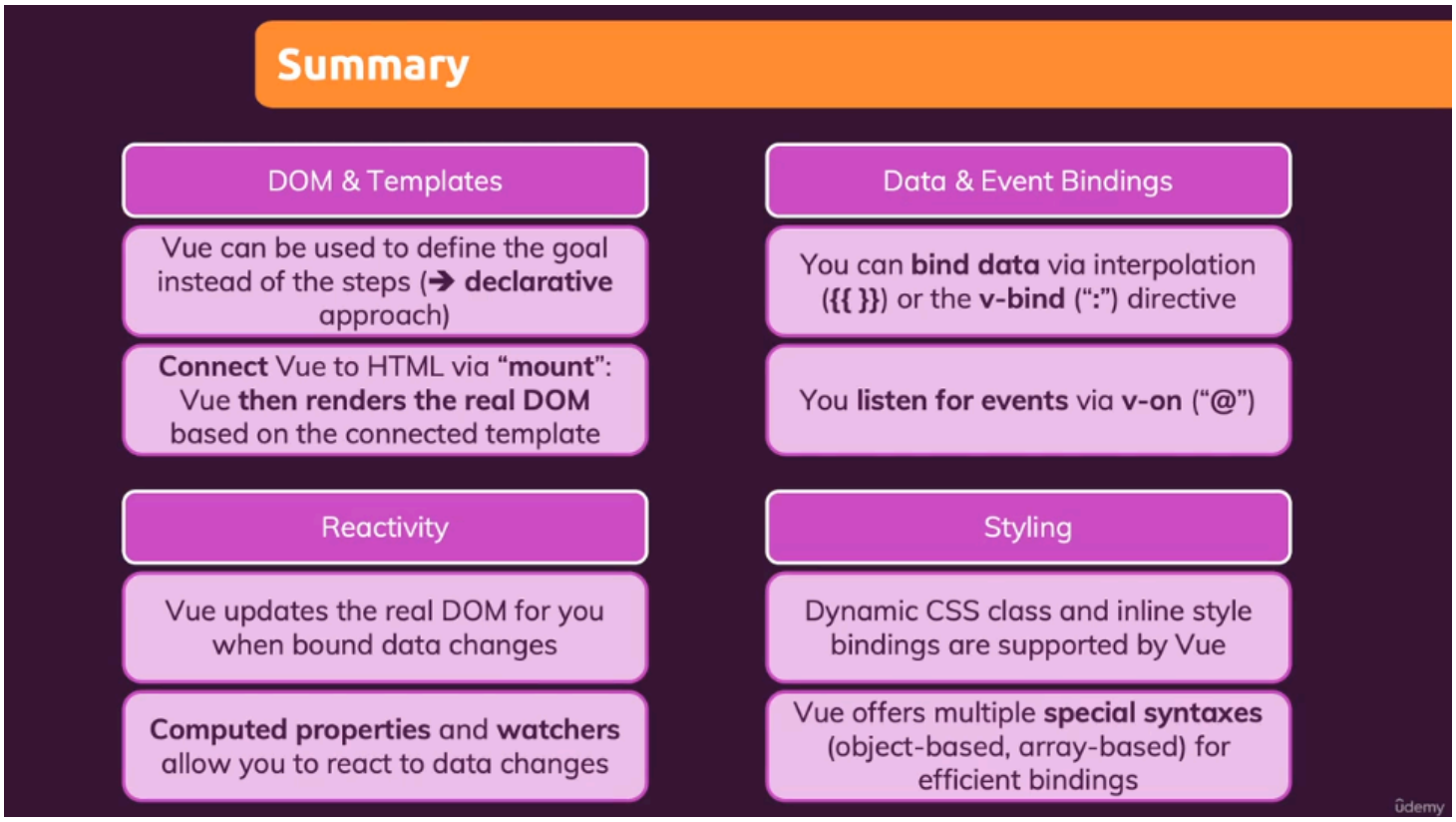

app.js
Section02 > basics-10-styling-starting-setup > app.js
Vue.createApp({
data: () => {
return {
boxASelected: false,
boxBSelected: false,
boxCSelected: false,
};
},
methods: {
boxSelected(selector) {
if (selector === "A") {
this.boxASelected = !this
} else if (selector === "B"
this.boxBSelected = !this
} else {
this.boxCSelected = !this
}
},
},
}).mount("#styling");
index.html U
styles.css U
Section02 > basics-10-styling-starting-setup > index.html > html > body > section#styling > div.demo
<!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=s
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 Dynamic Styling</h1>
</header>
<section id="styling">
<div :style="{borderColor: boxASelected ? 'red' : 'gray'}"
@:click="boxSelected('A')"
class="demo">
<div class="demo"
:class="{active: boxBSelected}" @:click="boxSelected('B')" >
<div class="demo"
:class="{active: boxCSelected}" @:click="boxSelected('C')" >
</section>
</body>
</html>
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
File F:/Program...
School
Elements
Console
Memory
Sources
top
Filter
Default levels
No Issues
You are running a development build of Vue.
Make sure to use the production build (*.prod.js) when deploying for production.
vue.global.js:12572
Relevant data is sent to Google

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 backgro
      <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
Data & Event Bindings
You can bind data via interpolation ({{ }}) or the v-bind (":") directive
You listen for events via v-on ("@")
Reactivity
Vue updates the real DOM for you when bound data changes
Computed properties and watchers allow you to react to data changes
Styling
Dynamic CSS class and inline style bindings are supported by Vue
Vue offers multiple special syntaxes (object-based, array-based) for efficient bindings
udemy