1) Базовая разметка:

<!doctype html>  
<html lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Урок 1</title>  
 <link rel="stylesheet" href="css/bootstrap.min.css">  
 <link href="css/styles.css" rel="stylesheet">  
 <script src="js/vue.js"></script>  
</head>  
<body>  
 <div class="wrapper">  
 <div class="sample">  
 <input type="text" v-model="***name***">  
 <hr>  
 <h2>Hello, {{ name }}</h2>  
 </div>  
 </div>  
  
  
 <script>  
  
 let ***app*** = ***Vue***.createApp({ // (создаем корневой компонент - могут быть другие объекты) создаем application - кусок разметки, которое вью контролирует  
 // сюда передается такой же объект, как и в new Vue()

data: ()=>({ // data - функция. которая возвращает объект () - скобки нужны для того, чтобы вернуть объект  
 name: ''  
 }),

});  
  
 let ***root*** = ***app***.mount('.sample'); // с помощью метода mount вписываем разметку в див sample  
 ***console***.log(***root***)  
 </script>  
</body>  
</html>

2) Как можно по другом переписать data ?

<script>  
 let ***app*** = ***Vue***.createApp({  
  
 data: function () {  
 return {  
 name: ''  
 }  
 }  
 });  
  
 let ***root*** = ***app***.mount('.sample');   
 ***console***.log(***root***)  
</script>

Но это древний способ.

Здесь реактивность данных работает на основе автообновления шаблона.

3) Как можно без v-model ?

<div class="wrapper">  
 <div class="sample">  
 <input type="text" v-bind:value="***name***" v-on:input="***name*** = ***$event***.target.value">  
 <hr>  
 <h2>Hello, {{ name }}</h2>  
 </div>  
</div>

4) С использованием метода

<div class="wrapper">  
 <div class="sample">  
 <input type="text" :value="***name***" @input="onInput">  
 <hr>  
 <h2>Hello, {{ name }}</h2>  
 </div>  
</div>

<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 name: ''  
 }  
 },  
 methods:{  
 onInput(e){  
 this.name = e.target.value  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');  
</script>

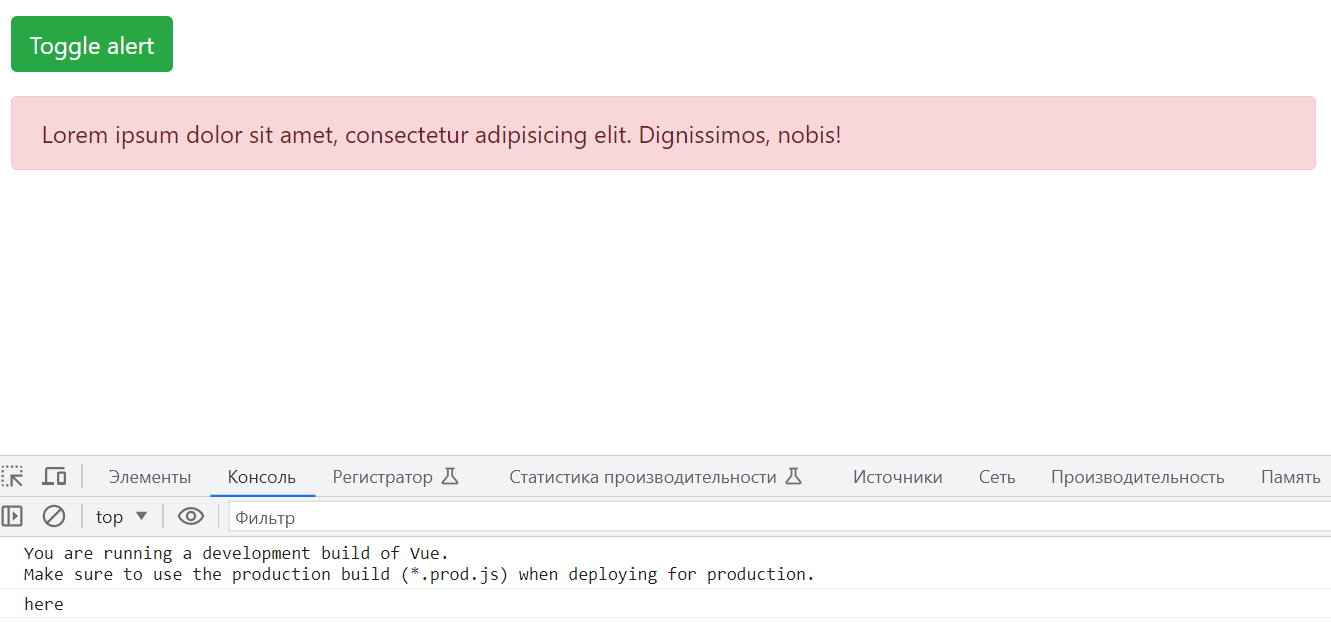
5) computed свойство

Производное поле. Т.е. данные вычисляются на основании других данных.

<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 showAlert: false,  
 name: '',  
 firstName:'',  
 lastName:'',  
 }  
 },  
 methods:{  
 onInput(e){  
 this.name = e.target.value  
 },  
 },  
 computed:{  
 fullName(){  
 ***console***.log('here')  
 return (this.firstName + ' ' + this.lastName).trim()  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');  
</script>

<h3>Hi {{fullName}}</h3> <!--но при таком потходе будет каждый раз рендерится страница-->

И теперь при toggle fullname не выполняется:



**Computed** это некий сеттер, который где-то там сохраняется.

**Вывод**:

**data(){…}** – чисто данные.

**computed** – данные расчитаные на основе других данных

**methods** – обработчики событий или произвольные функции, которые мы крайне не хотим вызывать в шаблоне

6) Практика по computed

Будет какой-то промокод. И базовая цена.

Рассчитать скидку на товар в зависимости от введенного промокода.

<div class="wrapper">  
 <div class="sample">  
 <input type="text" v-model.trim.lazy="promo"> <!--lazy - при потере фокуса-->  
  
<!-- {{getSale('em')}}-->  
<!-- {{getSale('some')}}-->  
 {{total}}  
 </div>  
</div>  
  
  
<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 promo:'',  
 price: 1000  
  
 }  
 },  
 methods:{  
 getSale(promo){  
 let codes = {  
 some: 10,  
 other: 20  
 }  
 return codes.hasOwnProperty(promo) ? codes[promo] : 0; // если промокод есть, то возьмем его  
 }  
 },  
 computed:{  
 total(){  
 let sale = this.getSale(this.promo) // размер скидки  
 return this.price \* (1 - sale / 1000)  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');

7) Усложним задачу.

Будем еще выводить скидку, так как только цена отображается

Будем сегментировать computed поля – разделять их на более мелкие единицы.

<div class="wrapper">  
 <div class="sample">  
 <input type="text" v-model.trim.lazy="promo"> <!--lazy - при потере фокуса-->  
  
 <!-- {{getSale('em')}}-->  
 <!-- {{getSale('some')}}-->  
 <div>Цена без скидки: {{price}}</div>  
 <div v-if="hasSale">  
 <div class="alert alert-danger">- {{ sale }}</div>  
 </div>  
 <div>Итоговая цена: {{total}}</div>  
 </div>  
</div>  
  
  
<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 promo: '',  
 price: 1000  
  
 }  
 },  
 methods: {  
 getSale(promo) {  
 let codes = {  
 some: 10,  
 other: 20  
 }  
 return codes.hasOwnProperty(promo) ? codes[promo] : 0; // если промокод есть, то возьмем его  
 }  
 },  
 computed: {  
 sale() {  
 return this.getSale(this.promo)  
 },  
 hasSale() {  
 return this.sale > 0  
 },  
 total() {  
 return this.price \* (1 - this.sale / 1000) // размер скидки  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');  
</script>

8) Computed-async

Бывают ситуации, когда computed поля бессильные.

Например, функция computed() становится асинхронной. Она стучиться на сервер за промокодами.

С эмулируем с помощью таймера.

<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 promo: '',  
 price: 1000  
  
 }  
 },  
 methods: {  
 getSale(promo, fn) {  
 setTimeout(()=>{  
 let codes = {  
 some: 10,  
 other: 20  
 }  
 let sale = codes.hasOwnProperty(promo) ? codes[promo] : 0; // если промокод есть, то возьмем его  
 fn(sale)  
 },500)  
  
 }  
 },  
 computed: {  
 sale() {  
 return this.getSale(this.promo)  
 },  
 hasSale() {  
 return this.sale > 0  
 },  
 total() {  
 return this.price \* (1 - this.sale / 1000) // размер скидки  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');  
</script>

Но так не будет работать, так как computed свойства должны быть синхронные, они не будут дожидаться окончания работы какой-то функции.

Тогда нужно отказаться от computed свойства, которое является асинхронным.

<script>  
 let ***app*** = ***Vue***.createApp({  
 data: function () {  
 return {  
 promo: '',  
 price: 1000,  
 sale: 0  
  
 }  
 },  
 methods: {  
 checkSale() {  
 this.getSale(this.promo, (sale) => { // передаем промокод и обработчик  
 this.sale = sale  
 })  
 },  
 getSale(promo, fn) {  
 setTimeout(() => {  
 let codes = {  
 some: 10,  
 other: 20,  
 }  
 let sale = codes.hasOwnProperty(promo) ? codes[promo] : 0;  
 return fn(sale)  
 }, 500)  
 }  
 },  
 computed: {  
 hasSale() {  
 return this.sale > 0  
 },  
 total() {  
 return this.price \* (1 - this.sale / 1000)  
 }  
 }  
 });  
 let ***root*** = ***app***.mount('.sample');  
</script>

**Вывод**: мы сделали асинхронную связь между promo / sale с помощью метода **checkSale()**