



# Eventos

Inmaculada Rodríguez Santiago

Jordi Barea Colomer

Factors Humans i Computació, 22-23

# Eventos

*`v-on:<evento a escuchar>`*

`v-on:click=" "`  
`v-on:click.right=" "`  
`v-on:input=" "`  
`v-on:keyup.enter=" "`  
`v-on:submit=" "`  
`v-on:submit.prevent=""`  
...

**HTML**  
(directivas v-)



**Vue app**  
(datos y  
lógica)

La forma abreviada de **v-on:evento** es **@evento**

# Ejemplo básico eventos Vue

<https://es.vuejs.org/v2/guide/events.html>

`v-on:click="codigo  
a ejecutar cuando  
se hace click"`

```
<div id="example-1">  
  <button v-on:click="counter += 1"> Agregar 1 </button>  
  <p>Se ha hecho clic en el botón de arriba {{counter}} veces  
</p>  
</div>
```

```
const app = new Vue.createApp({  
  data() {  
    return {counter: 0};  
  }  
});  
app.mount("#example-1");
```

**Ejemplo de evento on-click**

Agregar 1

Se ha hecho clic en el botón  
de arriba 0 veces

<https://jsfiddle.net/inmarodriguez/m34stLy5/11/>

# preventDefault en JS

<https://es.vuejs.org/v2/guide/events.html>

<https://developer.mozilla.org/es/docs/Web/API/Event/preventDefault>

En Javascript, el método `event.preventDefault()` cancela la acción por defecto asignada a ese evento (esto es, cancela el evento si es cancelable).

Acciones por defecto, ejemplo:

- En un anchor, al clicar en el texto, se produce el evento que carga la url
- En un input field, al teclear enter se recarga la página
- En un formulario, al clicar en el botón submit se envían los datos del formulario al backend

Haz click en el checkbox

☐ Checkbox

<https://jsfiddle.net/inmarodriguez/v2gy40zw/5/>

# Modificador .prevent en Vue

<https://es.vuejs.org/v2/guide/events.html>

Bajar código de lista de tareas, usar prevent para que al teclear enter en el input field, se añada la tarea a la lista.

*`v-on:<evento a escuchar>.<modificador>`*

`v-on:click=" "`

`v-on:click.right=" "`

`v-on:input=" "`

`v-on:keyup.enter=" "`

`v-on:submit=" "`

`v-on:submit.prevent=" "`

...

# Eventos (objeto nativo “event”) |

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building\\_blocks/Events#event\\_objects](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#event_objects)

- A cualquier método que se llama en respuesta a un evento se le pasa de forma automática un objeto “*event*” de Javascript. Podemos acceder a ese objeto para obtener valores que nos interesan.
- Por ejemplo, con el elemento **<input>**: *event.target.value* representa el caracter que se está introduciendo. Código a continuación ...

# Eventos (objeto nativo "event") II

<https://jsfiddle.net/inmarodriguez/ugkqt3xz/4/>

holaaa

holaaa

app.js

index.html

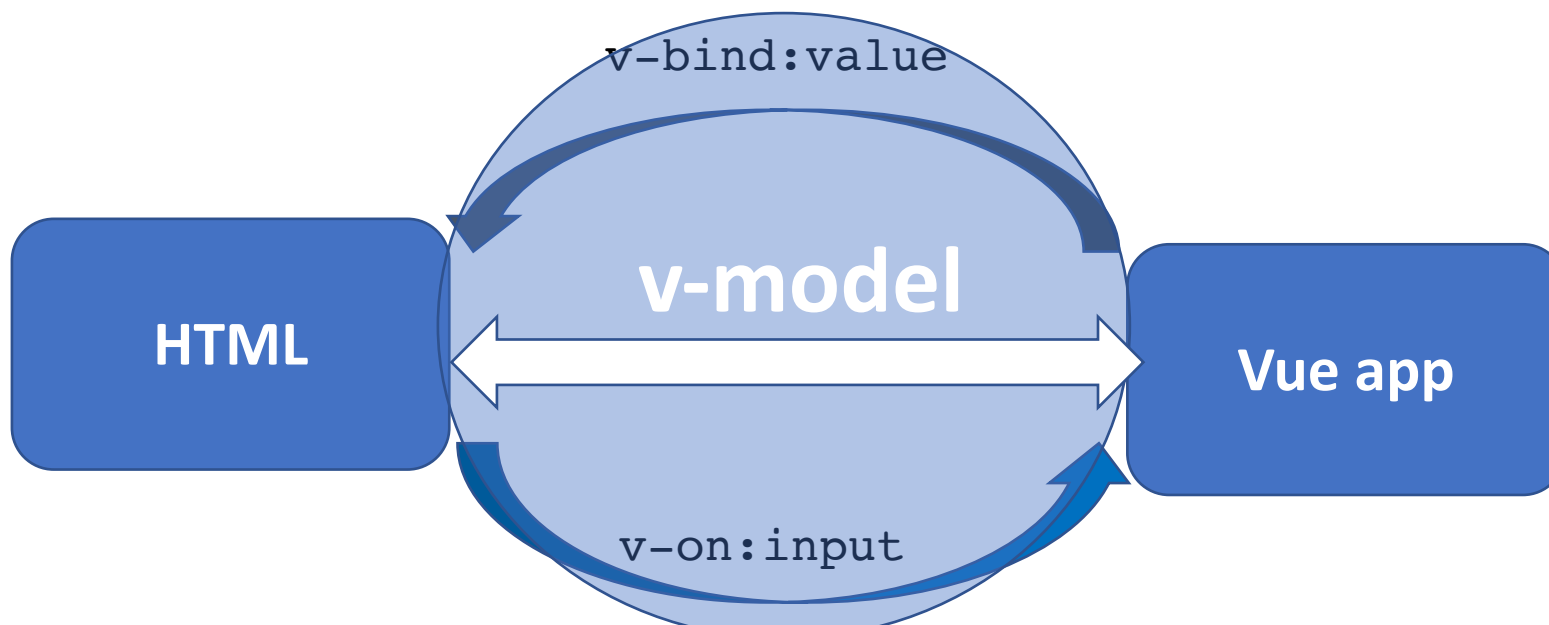
```
<div id="example-1">
  <input
    placeholder="Introduzca texto"
    v-on:keyup="muestraTexto" />
  <p>{{texto}}</p>
</div>
```

```
const app = Vue.createApp({
  data() {
    return {
      texto: "",
    };
  },
  methods: {
    muestraTexto(event) {
      this.texto = event.target.value;
    },
  },
});
```

# v-model (two-way data binding) I

Para enlazar el valor de un dato de Vue con un *input*, *textarea* y *select*

**v-model** = `v-bind:value + v-on:input`



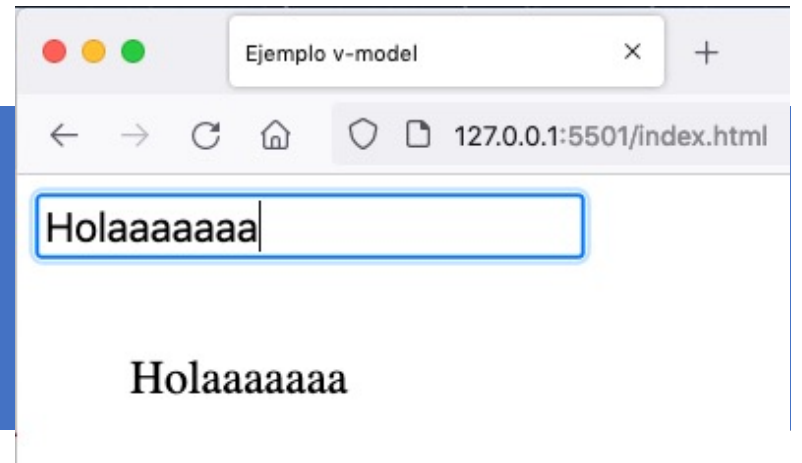
Si los datos de Vue cambian, el elemento input/textarea/select también cambia

y al revés

Si el input/textarea/select cambia los datos de Vue cambian



# v-model (two-way data binding) II



index.html

```
<div id="example-1">
  <input placeholder="Introduzca texto"
    v-model="texto" />
  <p>{{texto}}</p>
</div>
```

app.js

```
1  const app = Vue.createApp({
2    data() {
3      return {
4        texto: "",
5      };
6    },
7    methods: {},
8  });
```

# Computed properties I

- Cualquier método de Vue (*methods*), que no esté vinculado a un evento, se volverá a ejecutar por Vue siempre que algo en la pantalla cambie (se hace un re-render).
- Computed properties: son como los métodos pero que **sólo se volverán a ejecutar** si cambia una de sus dependencias (se produce “caching”).

# Computed properties II

- Son como los métodos pero **sólo se volverán a ejecutar** si cambia una de sus dependencias. Ej. Si `authors.books` no cambia, los accesos a `publishedBooksMessage()` siempre retornarán el valor computado previamente, solo con un cambio se vuelve a calcular.

<https://v3.vuejs.org/guide/computed.html#basic-example>

HTML

CSS

JS

Result

EDIT ON  
CODEPEN

LIVE

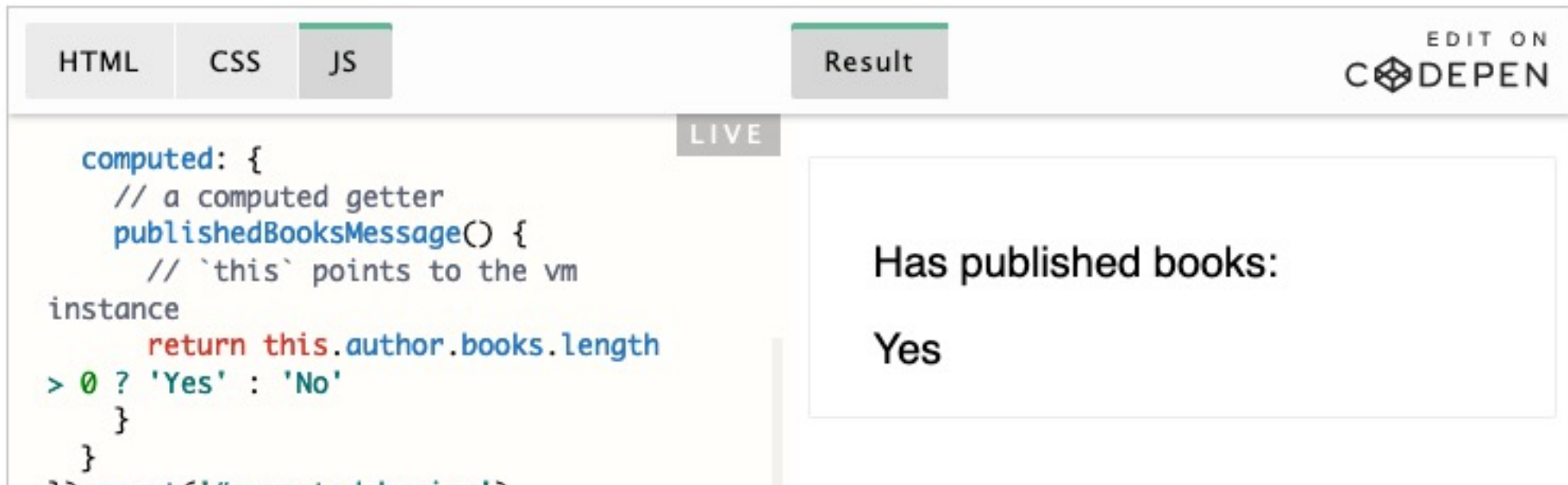
```
computed: {  
  // a computed getter  
  publishedBooksMessage() {  
    // `this` points to the vm  
    instance  
    return this.author.books.length  
  }  
}> 0 ? 'Yes' : 'No'
```

Has published books:  
Yes

# Computed properties II

- Una computed property permite componer nuevos datos a partir de otros.
- Se requiere que las computed properties sean **pure functions**. Esto es que retornan un valor nuevo, no pueden cambiar nada, y deben ser síncronas.

<https://v3.vuejs.org/guide/computed.html#basic-example>



The screenshot shows a CodePen live editor interface. At the top, there are tabs for 'HTML', 'CSS', and 'JS', with 'JS' being the active tab. To the right of these tabs is a 'Result' tab and a 'LIVE' button. In the top right corner, there is a link to 'EDIT ON CODEPEN'. The main area displays a JavaScript code snippet for a Vue.js component. The code defines a computed property named 'publishedBooksMessage' that returns 'Yes' if the author has books and 'No' otherwise. The code is as follows:

```
computed: {  
  // a computed getter  
  publishedBooksMessage() {  
    // `this` points to the vm  
    instance  
    return this.author.books.length  
  }  
}  
> 0 ? 'Yes' : 'No'
```

The 'Result' tab on the right shows the output of the code, which is 'Has published books: Yes'.

# Watchers

- Un Watcher es una función que queremos que se ejecute cuando alguna de sus dependencias cambian (es similar a las computed properties) pero...:
  - Se tienen que llamar como su dependencia (la data).
  - Se utilizan cuando queremos realizar operaciones asincrónicas o costosas en respuesta a datos que cambian.
- La función asignada a watch puede aceptar opcionalmente 2 parámetros. El primero es el nuevo valor de la propiedad. El segundo es el antiguo valor de la propiedad.

# Watchers (ejemplo)

Escribe un numero

El numero: 5

|                                  |                           |
|----------------------------------|---------------------------|
| El numero ha pasado de ser 0 a 1 | <a href="#">app.js:35</a> |
| El numero ha pasado de ser 1 a 2 | <a href="#">app.js:35</a> |
| El numero ha pasado de ser 2 a 3 | <a href="#">app.js:35</a> |
| El numero ha pasado de ser 3 a 4 | <a href="#">app.js:35</a> |
| El numero ha pasado de ser 4 a 5 | <a href="#">app.js:35</a> |

```
const miapp = Vue.createApp({
  data () {
    return {
      contador: 0,
      texto: 'texto inicial',
      numero: 0
    };
  },
  watch: { //cuando cambia la data el watcher numero se ejecuta
    numero(nuevoValor, valorAnterior){
      console.log("El numero ha pasado de ser %s a %s", valorAnterior, nuevoValor);
    }
  },
});
```

# Ejercicio

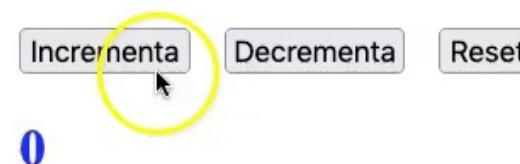
Bajar el código base del CV:  
EjercicioEventosCodigoBase

Generar una salida como la imagen de la derecha:

- En 1., con el botón “incrementa” se incrementa el valor del número de 2 en 2 y con “decrementa” lo decrementa en una unidad. Reset, resetea el valor a 0.
  - Usa estilos dinámicos `v-bind:style="{propiedad: calculaValorProp}"` para mostrar el número en color rojo si es negativo y en azul si es positivo. `calculaValorProp` será un computed en Vue que devuelve un valor para propiedad.
- En 2., usa “event” de JS para mostrar de forma dinámica en “El texto: “ el texto que se está escribiendo en el elemento input.
- En 3., idem que anterior pero con un número (usa v-model),

## Ejercicios de eventos

### 1. Evento click en elementos buttons



### 2. Objeto event en elemento input (tipo texto)

Escribe un texto

El texto: texto inicial

### 3. v-model en elementos input (tipo number)

Escribe un número