**Fuente: tutorialspoint.com // platzi.com // lenguajejs.com**

## ¿Qué es Vue.JS?

Vue.JS es un framework JavaScript progresivo de código abierto que se utiliza para desarrollar interfaces web interactivas. Es uno de los marcos famosos que se utilizan para simplificar el desarrollo web. Vue.JS se centra en la capa de vista. Se puede integrar fácilmente en grandes proyectos para el desarrollo front-end sin ningún problema.

La instalación de Vue.JS es muy fácil. Cualquier desarrollador puede comprender y crear interfaces web interactivas fácilmente en cuestión de tiempo. Vue.JS fue creado por Evan You, un ex empleado de Google. La primera versión de Vue.JS se lanzó en febrero de 2014.

### Virtual DOM

Vue.JS hace uso del DOM virtual, que también es utilizado por otros frameworks como React, Ember, etc. Los cambios no se realizan en el DOM, sino que se crea una réplica del DOM que está presente en forma de estructuras de datos JavaScript. Siempre que se deben realizar cambios, se realizan en las estructuras de datos de JavaScript y esta última se compara con la estructura de datos original. Luego, los cambios finales se actualizan al DOM real, que el usuario verá cambiar. Esto es bueno en términos de optimización, es menos costoso y los cambios se pueden realizar a un ritmo más rápido.

### Data Binding

La función de data binding ayuda a manipular o asignar valores a atributos HTML, cambiar el estilo, asignar clases con la ayuda de la directiva de enlace llamada **v-bind** disponible en Vue.JS.

### Components

Los componentes son una de las características importantes de Vue.JS que ayudan a crear elementos personalizados, que se pueden reutilizar en HTML.

### Event Handling

**v-on** es el atributo agregado a los elementos DOM para escuchar los eventos en Vue.JS.

### Computed Properties

Esta es una de las características más importantes de Vue.JS. Ayuda a escuchar los cambios realizados en los elementos de la interfaz de usuario y realiza los cálculos necesarios. No hay necesidad de codificación adicional para este fin.

### Templates

Vue.JS proporciona plantillas basadas en HTML que unen el DOM con los datos de la instancia de Vue. Vue compila las plantillas en funciones virtuales DOM Render. Podemos hacer uso de la plantilla de las funciones de render y para hacerlo tenemos que reemplazar la plantilla con la función de render.

### Directives

Vue.JS tiene directivas integradas como **v-if, v-else, v-show, v-on, v-bind** y **v-model**, que se utilizan para realizar varias acciones en la interfaz.

### Watchers

Los Watchers se aplican a los datos que cambian. Por ejemplo, elementos de input en formularios. Aquí, no tenemos que agregar ningún evento adicional. Los Watchers se encargan de manejar cualquier cambio de datos haciendo que el código sea simple y rápido.

### Liviano

Vue.JS es muy ligero y el rendimiento también es muy rápido.

Hay muchas formas de instalar Vue.JS. Algunas de las formas de cómo realizar la instalación se comentan a continuación.

### Usando CDN

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Introducción a Vue.JS</title>
    <link rel="stylesheet" href="css/estilos.css">
    <script src="https://unpkg.com/vue@3"></script>
</head>
```

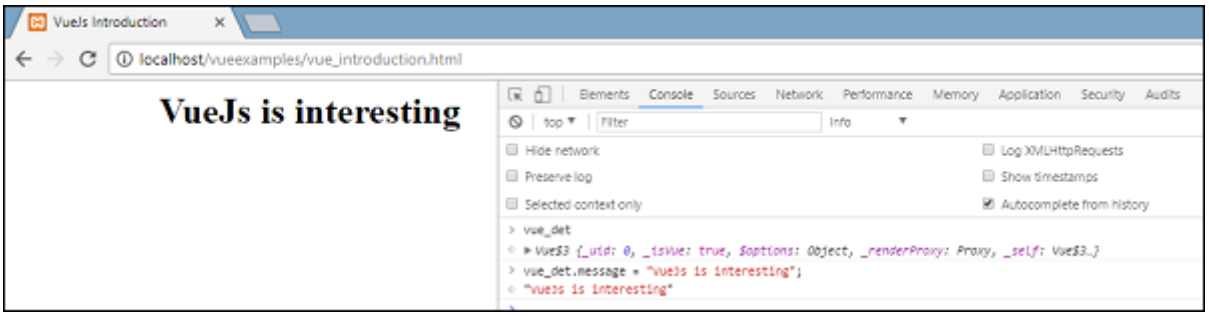
Para este ejemplo se incluyó el import del archivo js de Vue.JS dentro de la etiqueta <head>. El div presente tiene un mensaje en una interpolación {{}}. Esto interactúa con Vue.JS y muestra los datos en el navegador. Para mostrar el valor del mensaje en el DOM, se crea una instancia de Vue.JS.

```html
<div id="intro">
    <h1>{{ message }}</h1>
</div>
```

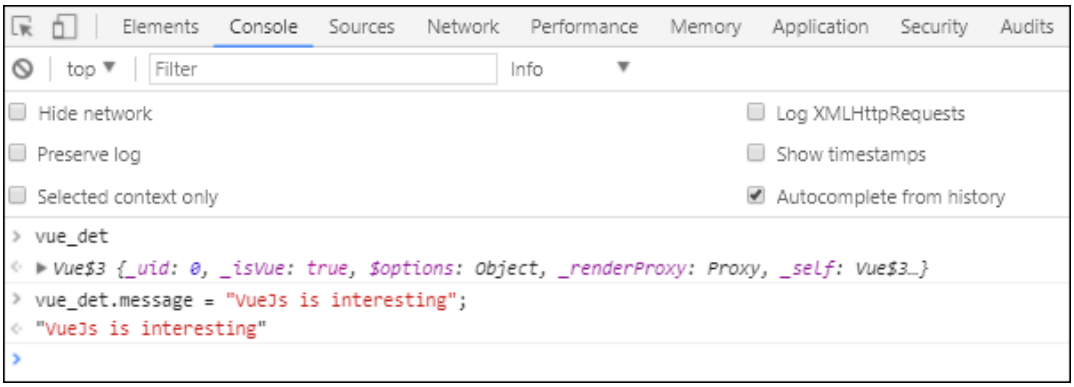
```javascript
const { createApp, ref, computed} = Vue;

//Ejemplo 1
const ejemplo1 = Vue.createApp({
    data(){
        return{
            message: 'My first VueJS Task'
        }
    }
}).mount("#intro");
```

Se llama a la instancia de Vue, la cual toma el ID del elemento del DOM, por ejemplo '#intro'. Hay datos dentro de dicha instancia con el mensaje al cual se le asigna el valor 'My first VueJS Task'. Vue.JS interactúa con el DOM y cambia el valor en el DOM {{message}} con el mensaje 'My first VueJS Task'.

También es posible cambiar el valor del mensaje en consola:




**Detalles en consola**




Para empezar con Vue.JS es necesario crear una instancia de Vue, que es conocida como la instancia Root de Vue.

**Sintaxis**

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Introducción a Vue.JS</title>
    <link rel="stylesheet" href="css/estilos.css">
    <script src="https://unpkg.com/vue@3"></script>
</head>

<body>

    <div id="ejemplo2">
        <ul>
            <li>Nombre : {{nombre}}</li>
            <li>Apellido : {{apellido}}</li>
            <li>Presentación: {{detalles()}}</li>
        </ul>
    </div>

    <script src="js/app.js"></script>
</body>

</html>
```

```
const { createApp} = Vue;

//Ejemplo 2
const ejemplo2 = Vue.createApp({
    data(){
        return{
            nombre: "Juan",
            apellido: "Pérez",
            direccion: "Buenos Aires"
        }
    },
    methods: {
        detalles() {
            return "Soy " + this.nombre + " " + this.apellido + " y vivo en " + this.direccion;

        }
    }
}).mount("#ejemplo2");
```

Hay un método llamado **mount**. Toma el id del elemento a modificar en el DOM.

```
<div id = "#ejemplo2"></div>
```

Lo que escribamos dentro de los {{}} afectará únicamente al div seleccionado y a nada más fuera de él. Posteriormente se definen los datos, nombre, apellido y dirección.

```
<div id = "ejemplo2">
```

```
<div id="ejemplo2">
    <ul>
        <li>Nombre : {{nombre}}</li>
        <li>Apellido : {{apellido}}</li>
        <li>Presentación: {{detalles()}}</li>
    </ul>
</div>
```

El valor del nombre se reemplazará dentro de la primera interpolación y así sucesivamente con los demás parámetros.

Posteriormente se define una función llamada mydetails() que devuelve un valor que será asignado a la interpolación siguiente:

```
<h1>{{detalles()}}</h1>
```

**Output**

---

- Nombre : Juan
- Apellido : Pérez
- Presentación: Soy Juan Pérez y vivo en Buenos Aires

Si necesitamos insertar código HTML en el dom desde Vue.JS se utiliza la directiva **v-html**. Desde el momento que se usa Vue.JS ya sabe que tiene que mostrar dicho contenido como HTML en el navegador.

```html
<div id="ejemplo3">
    <ul>
        <li>Nombre : {{nombre}}</li>
        <li>Apellido : {{apellido}}</li>
    </ul>
    <div v-html="contenidoHtml"></div>
    <img v-bind:src="imgsrc" v-bind:alt="imgalt" width="250" height="250">
</div>
```

```javascript
const ejemplo3 = Vue.createApp({
    data(){
        return{
            nombre: "Juan",
            apellido: "Pérez",
            contenidoHtml: "<div><h2>Imagen de un paisaje</h2></div>",
            imgsrc: "img/paisaje.jpg",
            imgalt: "Imagen de un paisaje"
        }
    }
}).mount("#ejemplo3");
```

## Imagen de un paisaje

**v-once**

Todo elemento que posea la directiva **v-once** será renderizado sólo una vez.

<span v-once>Mensaje: {{ msg }}

**Componentes**

```
<div id="ejemplo4">
    <custom-component></custom-component>
</div>
<div id="ejemplo5">
    <custom-component-two></custom-component-two>
</div>
```

```
const CustomComponent1 = {
    template: `<div><h2>{{buttonText}}</h2></div>`,
    data() {
        return {
            'buttonText': "Componente Custom en Vue.JS 3"
        }
    }
}

const ejemplo4 = Vue.createApp({
    components: {
        'custom-component': CustomComponent1
    }
}).mount("#ejemplo4");
```

```
const CustomComponent2 = {
    template: `<div v-on:mouseover = "cambiarNombre()" v-on:mouseout = "reestablecerNombre();">
                    <h1>Componente creado por <span id = "nombre">{{nombre}}</span></h1>
                </div>`,
    data(){
        return{
            nombre: "Juan"
        }
    },
    methods: {
        cambiarNombre() {
            this.nombre = "Martín";
        },
        reestablecerNombre() {
            this.nombre = "Juan";
        }
    }
}

const ejemplo5 = Vue.createApp({
    components: {
        'custom-component-two': CustomComponent2
    }
}).mount("#ejemplo5");
```

Se crean dos div con id ejemplo4 y ejemplo5, junto con dos instancias Vue que hacen referencia a dichos ids. Se crean dos componentes que serán asignados a cada una instancias.

# Componente Custom en Vue.JS 3

# Componente creado por Juan

Dentro del componente se agrega un template el cual tiene asignado código HTML. Esta es la manera de registrar globalmente un componente en Vue.JS, que puede ser reutilizado en cualquier instancia Vue.

El nombre personalizado de las etiquetas será reemplazado por el código del template en escrito en la creación del componente. En el navegador no existe ningún componente con etiquetas <custom-component></custom-component>.

También se puede modificar el comportamiento de un componente global dentro de una instancia Vue.

En el ejemplo anterior, el template dentro del componente hace que cuando el mouse se posicione encima del mismo ejecute la función cambiarNombre() y cuando el mismo salga del componente se ejecute la función reestablecerNombre(). Ambas funciones trabajan sobre la propiedad name del componente, cambiandola.

**Componentes dinámicos**

Los componentes dinámicos se crean con la etiqueta <component></component> y se ligan con un elemento del DOM de la siguiente manera.

Los componentes dinámicos son creados de la siguiente manera:

El componente posee una directiva **v-bind:is**, cuyo valor es “view” y un valor asignado a ese **view**. View está definido en la instancia Vue.

```
<div id="ejemplo6">
    <component v-bind:is="vista"></component>
</div>
```

```
const ejemplo6 = Vue.createApp({
    data(){
        return{
            vista: 'componente'
        }
    },
    components: {
        'componente': {
            template: `<div>
                            <span style = "font-size:3rem;color:green;">
                                Componente dinámico
                            </span>
                        </div>`
        }
    }
}).mount("#ejemplo6");
```

**Propiedades computadas**

Las propiedades computadas son unas variables pre-calculadas que puedes reutilizar en Vue sin calcularlas cada vez que las utilices, por lo que pueden ser especialmente interesantes para cálculos u operaciones costosas. Dichas propiedades computadas solo se vuelven a recalcular si detectan que uno de los parámetros implicados cambian, de modo que al calcularse una vez, suele cachear los resultados.

```
<div id="ejemplo7">
    Nombre : <input type="text" v-model="nombre" /><br><br> Apellido : <input type="text" v-model="apellido" /><br><br>
    <h2>Mi nombre es {{nombre}} {{apellido}}</h2>
    <h2>Usando una propiedad computada: {{getNombreCompleto}}</h2>
</div>
```

```
const { createApp, computed} = Vue;

//Ejemplo 7
const ejemplo7 = Vue.createApp({
    data(){
        return{
            nombre: "",
            apellido: ""
        }
    },
    computed: {
        getNombreCompleto() {
            return this.nombre + " " + this.apellido;
        }
    }
}).mount("#ejemplo7");
```

Se crea el documento con dos elementos input tipo caja de texto, los cuales están ligados a la instancia Vue con las directivas **v-model**.

Posteriormente tenemos la propiedad computada getNombreCompleto, que devuelve el nombre y el apellido que ingresó el usuario concatenados.

```
computed :{
  getNombreCompleto(){
    return this.nombre +" "+ this.apellido;
  }
}
```

Lo que se escribe en los textboxes es lo mismo que devolverá la función, cuando los valores de los textboxes cambien así también lo hará lo devuelto por la función, gracias a utilizar las propiedades computadas que se llaman automáticamente cuando los textboxes cambien.

**Acceder a los elementos del DOM utilizando $refs**

Todo el tiempo necesitamos acceder a elementos del DOM utilizando JavaScript.
Vue.JS nos permite hacerlo de una manera supremamente sencilla.
Las instancias de Vue.JS cuentan con diversas propiedades, una de ellas es $refs.
Visto en código sería algo como:

app.$refs
vm.$refs

Donde “app” o “vm” representan la instancia misma de Vue (por convención se utiliza “app” o “vm” para nombrar a la instancia de Vue, pero puedes utilizar el nombre que desees, también puedes utilizar la palabra reservada “this” para referirte a la instancia. En éste ejemplo utilizaremos el nombre de la instancia, en cuyo caso es “app”) y $refs sería una propiedad propia de la instancia.

Ahora bien, ¿qué es examente “$refs”?
Es un objeto, dentro de él se van a almacenar todos los elementos del DOM que cuenten con el atributo “ref”. El atributo “ref” vendría a ser algo así como darle un ID al elemento.
<input ref="entrada"></input>

Podemos tener todos los elementos que deseemos con el atributo “ref”, siempre y cuando el valor del atributo sea diferente para cada elemento.
<input ref="entrada"></input>
<input ref="entrada2"></input>

Para acceder al objeto que almacena estos elementos bastaría con llamarlo de la siguiente manera:
app.$refs

Y para acceder al elemento dentro del objeto:
app.$refs.entrada

Vamos a aplicar lo anterior con una aplicación sencilla
Lo que hace la aplicación es añadir a un párrafo el texto que escribamos en una entrada:

Lo primero es crear el HTML y la instancia de Vue.

Vamos colocarle los atributos “ref” a los elementos cuyas propiedades queremos acceder, que en este caso son el input y el párrafo.

Ahora vamos a escribir los métodos **addText()** y **deleteText()** de los botones en la instancia de Vue.

Y por último ponemos los botones a la escucha de los métodos.

```
const ejemplo8 = Vue.createApp({
    methods: {
        addText() {
            const text          = this.$refs.text.value;
            const textField     = this.$refs.textField;
            textField.innerHTML = textField.innerHTML + '<br>' + text;
        },
        deleteText() {
            const textField     = this.$refs.textField;
            textField.innerHTML = '';
        }
    }
}).mount("#ejemplo8");
```

```
<div id="ejemplo8">
    <h1>Accediendo a Elementos del DOM utilizando $refs</h1>
    <input type="text" ref="text">
    <br><br>
    <button type="button" @click="addText">Guardar</button>
    <button type="button" @click="deleteText">Borrar</button>
    <p ref="textField"></p>
</div>
```

**Fuente:tutorialspoint.com**

**Watchers**

```
<div id="ejemplo1">
    <p>Kilometros:</p><input type="text" v-model="kilometros">
    <p>Metros:</p><input type="text" v-model="metros">
</div>
```

```
const { createApp, watch} = Vue;

//Ejemplo 1
const ejemplo1 = Vue.createApp({
    data(){
        return{
            kilometros: 0,
            metros: 0
        }
    },
    watch: {
        kilometros(val) {
            this.kilometros = val;
            this.metros = val * 1000;
        },
        metros(val) {
            this.kilometros = val / 1000;
            this.metros = val;
        }
    }
}).mount("#ejemplo1");
```

Se crean dos textboxes, uno con kilómetros y otro con metros. En **data** ambas propiedades son inicializadas en cero. Existe dentro de la instancia Vue un objeto **watch** que se crea con dos funciones, cuyo objetivo es convertir de kilómetros a metros y viceversa.

Cada vez que se ingrese algún valor en los textboxes, **watch** se encarga de actualizarlos calculando lo que está declarado en las funciones.

Estos watchers sólo deben usarse cuando las propiedades calculadas se nos quedan cortas, que suele ser en situaciones en las que necesitamos asincronía o se trata de operaciones muy costosas para cambiar datos.


**Más sobre manipulación de atributos**

```
<div id="ejemplo2">
    <p>{{title}}</p><br>
    <a v-bind:href="hreflink" target="_blank">Ir a Google.</a><br/>
</div>
```

```
const ejemplo2 = Vue.createApp({
    data(){
        return{
            title: "Ejemplo 2",
            hreflink: "http://www.google.com"
        }
    }
}).mount("#ejemplo2");
```

En el ejemplo anterior se presentan tres supuestas formas de enlazar el atributo **href** a los enlaces. Pero sólo una es correcta al revisar el ejemplo en el navegador.

Como fue demostrado en anteriores ejemplos, para enlazar atributos desde una instancia Vue al DOM se utiliza la directiva **v-bind:atributo**. En este caso trabajaremos sobre **href.**

```
<a v-bind:href = "hreflink" target = "_blank">Click Me </a>
```

Vue.JS nos brinda un atajo a dicha directiva eliminando **v-bind.** Ninguna de las propiedades de Vue.JS será mostrada en el inspector del navegador.

```
<a :href = "hreflink" target = "_blank">Click Me </a>
```

**Enlazando clases HTML**

Para enlazar clases se utiliza también la directiva **v-bind**, pero utilizando el atributo **class**.

```
const ejemplo3 = Vue.createApp({
    data(){
        return{
            title: "Ejemplo3",
            isActive: true,
            hasError: false
        }
    }
}).mount("#ejemplo3");
```

```
.active {
    color: #4F8A10;
    background-color: #DFF2BF;
}

.displayError{
    color: #D8000C;
    background-color: #FFBABA;
}
```

```
<div id="ejemplo3">
    <div class="info" v-bind:class="{ active: isActive, displayError: hasError }">
        <p>{{title}}</p>
    </div>
</div>
```

isactive es una variable booleana, la cual puede contener en su interior true o false lo cual hará que se aplique o no la clase **active** al div enlazado a la instancia Vue. La clase active cambia el fondo del elemento a un color rojo.

Ejemplo3

**Enlazar elementos input de un formulario**

```
<div id="ejemplo4">
    <h2>Textbox</h2>
    <input v-model="nombre" placeholder="Nombre">
    <h3>Nombre: {{nombre}}</h3>
    <h2>Textarea</h2>
    <textarea v-model="mensaje" placeholder="Mensaje"></textarea>
    <h1>
        <p>{{mensaje}}</p>
    </h1>
    <h2>Checkbox</h2>
    <input type="checkbox" id="checkbox" v-model="checked"> {{checked}}
    <h2>Radio</h2>
    <input type="radio" id="blanco" value="Blanco" v-model="seleccionado">Blanco
    <input type="radio" id="negro" value="Negro" v-model="seleccionado">Negro
    <h3>Elemento seleccionado: {{seleccionado}} </h3>
    <h2>Select</h2>
    <select v-model="lenguaje">
        <option disabled value="">Seleccionar...</option>
        <option>Java</option>
        <option>Javascript</option>
        <option>Php</option>
        <option>C</option>
        <option>C++</option>
        <option>Python</option>
     </select>
    <h3>Seleccionado: {{lenguaje}}</h3>
</div>
<hr>
```

Lo que escribamos en el textbox se mostrará debajo. **v-model** tiene asignado el valor de la propiedad **name** y dicho nombre es mostrado en {{name}}, que muestra lo que tenga escrito en su interior el textbox, lo mismo sucede con el checkbox y el textarea.

```
//Ejemplo 4
const ejemplo4 = Vue.createApp({
    data(){
        return{
            nombre: 'Juan',
            mensaje: 'Soy un mensaje',
            checked: true,
            seleccionado: 'Negro',
            lenguaje: "C++"
        }
    }
}).mount("#ejemplo4");
```

```html
<div id="ejemplo6">
    <h3>Sumar 100 + 200 = {{total}}</h3>
    <button v-on:click="mostrarResultado">Presione</button>
    <!--<button @click="mostrarResultado">Presione</button>-->
</div>
<div id="ejemplo5">
    <span style="font-size:25px;">Edad:</span><input v-model.number="age" type="number">
    <p>Edad: {{age}}</p>
    <br>
    <span style="font-size:25px;">Mensaje:</span> <input v-model.lazy="msg">
    <p>Mensaje: {{msg}}</p>
    <br>
    <span style="font-size:25px;">Ingrese Mensaje: </span><input v-model.trim="message">
    <p>Mensaje: {{message}}</p>
</div>
```

```javascript
const ejemplo5 = Vue.createApp({
    data(){
        return{
            age: 0,
            msg: '',
            message: ''
        }
    }
}).mount("#ejemplo5");
```

El modificador **number** sólo permite el ingreso de números, no tomará otra entrada salvo que sean números.

```
<span style = "font-size:25px;">Edad:</span> <input v-model.number = "age" type = "number">
```

El modificador **lazy** mostrará el contenido presente en el textbox cuando el usuario abandone el mismo.

```
<span style = "font-size:25px;">Mensaje:</span> <input v-model.lazy = "msg">
```

El modificador **trim** eliminará los espacios al principio y al final de lo ingresado en el textbox.

```
<span style = "font-size:25px;">Ingrese Mensaje : </span><input v-model.trim =
"message">
```

### Eventos

Utilizaremos la directiva **v-on** en los elementos HTML del DOM para escuchar a los eventos.

### Evento de Click

```html
<div id="ejemplo6">
    <h3>Sumar 100 + 200 = {{total}}</h3>
    <button v-on:click="mostrarResultado">Presione</button>
    <!--<button @click="mostrarResultado">Presione</button>-->
</div>
```

```
const ejemplo6 = Vue.createApp({
    data(){
        return{
            num1: 100,
            num2: 200,
            total: 0
        }
    },
    methods: {
        mostrarResultado(event) {
            console.log(event);
            return this.total = this.num1 + this.num2;
        }
    }
}).mount("#ejemplo6");
```

**Sumar 100 + 200 = 300**

Presione

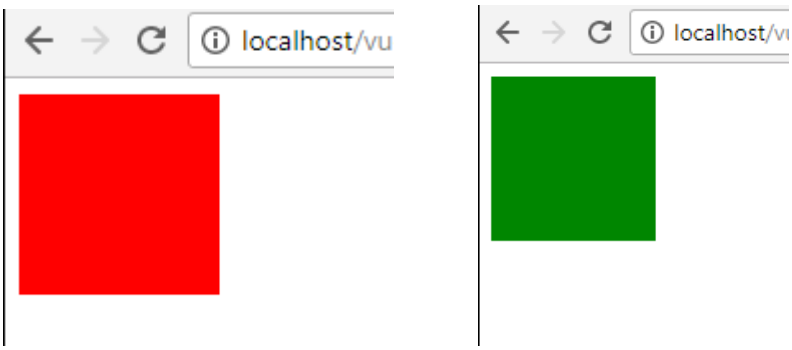
**Eventos mouseover, mouseout**

Se crea un div con ancho y alto de 100 px. Se le asigna un background de color rojo. Cuando el mouse se posiciona sobre él se cambiará dicho fondo al color verde y cuando el mouse salga del elemento volverá al color rojo mediante los métodos changebgcolor y originalcolor.

```
<div v-bind:style = "styleobj" v-on:mouseover = "changebgcolor" v-on:mouseout =  "originalcolor"></div>
```

```
const ejemplo7 = Vue.createApp({
    data(){
        return{
            styleobj: {
                width: "100px",
                height: "100px",
                backgroundColor: "red"
            }
        }
    },
    methods: {
        changebgcolor() {
            this.styleobj.backgroundColor = "green";
        },
        originalcolor() {
            this.styleobj.backgroundColor = "red";
        }
    }
}).mount("#ejemplo7");
```

```html
<div id="ejemplo7">
    <div v-bind:style="styleobj" @mouseover="changebgcolor" @mouseout="originalcolor"></div>
</div>
```




**Modificadores de eventos**

Vue.JS tiene disponibles modificadores de eventos en la directiva **v-on**.

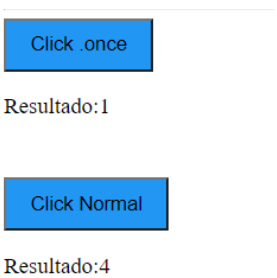
**.once**

Permite que el evento se ejecute sólo una vez.

```html
<div id="ejemplo8">
    <button @click.once="buttonclickedonce" v-bind:style="styleobj">Click .once</button>
    <p>Resultado:{{clicknum}}</p>
    <br><br>
    <button @click="buttonclicked" v-bind:style="styleobj">Click Normal</button>
    <p>Resultado:{{clicknum1}}</p>
</div>
```

```javascript
const ejemplo8 = Vue.createApp({
    data(){
        return{
            clicknum: 0,
            clicknum1: 0,
            styleobj: {
                backgroundColor: '#2196F3!important',
                cursor: 'pointer',
                padding: '8px 16px',
                verticalAlign: 'middle'
            }
        }
    },
    methods: {
        buttonclickedonce() {
            this.clicknum++;
        },
        buttonclicked() {
            this.clicknum1++;
        }
    }
}).mount("#ejemplo8");
```

En el anterior ejemplo se crean dos botones. Uno que posee el modificador **.once** y otro no. Uno podrá ejecutar tantas veces como el usuario desee el evento y el otro (con el modificador .once) no.



**.prevent**

```
<a href = "http://www.google.com" v-on:click.prevent = "clickme">Click Me</a>
```

Si el link no posee el modificador **.prevent** se abrirá el enlace al que apunta el link. Dicho comportamiento es el que tienen por defecto los enlaces al ser clickeados. Con .prevent evitamos dicho comportamiento y ejecutamos la función especificada en las comillas.

```
<a href = "http://www.google.com" v-on:click.prevent = "clickme" target = "_blank" v-bind:style = "styleobj">Click Me</a>
```

**Modificadores de teclas**

Vue.JS ofrece modificadores de teclas mediante los cuales se manejan los eventos. Si necesitaramos que un textbox llame a un método definido en una instancia Vue sólo cuando se pulsa la tecla enter deberíamos agregar un modificador.

```
<input type = "text"  v-on:keyup.enter = "showinputvalue"/>
```

Podemos utilizar múltiples teclas. Por ejemplo: V-on.keyup.ctrl.enter

```
<div id="ejemplo9">
    <h3>Ingrese nombre y presione Enter</h3>
    <input type="text" v-on:keyup.enter="mostrarValor" v-bind:style="styleobj" placeholder="Ingrese nombre" />
    <h2>{{name}}</h2>
</div>
```

```
const ejemplo9 = Vue.createApp({
    data(){
        return{
            name: '',
            styleobj: {
                width: "30%",
                padding: "12px 20px",
                margin: "8px 0",
                boxSizing: "border-box"
            }
        }
    },
    methods: {
        mostrarValor(event) {
            this.name = event.target.value;
        }
    }
}).mount("#ejemplo9");
```

**Renderizado condicional**

**v-if**

```
<div id="ejemplo10">
    <button v-on:click="showdata" v-bind:style="styleobj">Presionar</button>
    <span style="font-size:25px;font-weight:bold;">{{show}}</span>
    <h1 v-if="show">Tag h1</h1>
    <h2 v-else>Tag h2</h2>
    <div v-show="show">
        <h2>V-Show:</h2>
        <img src="img/paisaje.jpg" width="100" height="100">
    </div>
</div>
```

Se crea un botón con dos etiquetas de encabezados con un mensaje en el interior de ellas. Una variable llamada **show** es declarada e inicializada con un valor **true**. Su valor se muestra cerca del botón. Cada vez que se clickea el botón se llama al método **showdata** que cambia el estado de dicha variable de false a true y viceversa

Si el valor de la variable show es falso la etiqueta <h1> no se mostrará.

```
const ejemplo10 = Vue.createApp({
    data(){
        return{
            show: true,
            styleobj: {
                backgroundColor: '#2196F3!important',
                cursor: 'pointer',
                padding: '8px 16px',
                verticalAlign: 'middle'
            }
        }
    },
    methods: {
        showdata() {
            this.show = !this.show;
        }
    }
}).mount("#ejemplo10");
```

**v-else** se añade utilizando la siguiente sintaxis:

```
<h1 v-if = "show">Tag h1</h1>
<h2 v-else>Tag h2</h2>
```

Ahora, si **show** es true, se mostrará “Tag h1”, si es false, se mostrará “Tag h2”

### v-show

v-show se comporta igual que v-if. También muestra y oculta los elementos basado en una condición. La diferencia es que v-if los elimina del DOM si la condición es falsa y v-show sólo los oculta.

## Renderizado de Listas

### v-for

Se declara como array una variable llamada **items**. En la sección de métodos hay uno que se llama showinputvalue, el cual es asignado al textbox y recibe el nombre de las frutas. En el método dichos nombres serán agregados al array mediante el siguiente código.

```
showinputvalue(event) {
  this.items.push(event.target.value);
}
```

Usaremos **v-for** para mostrar las frutas ingresadas al array mediante el siguiente código. v-for sirve para iterar sobre los elementos presentes en el array.

Para iterar sobre el array en bucle utilizamos v-for = ”a in items” en donde **a** tendrá en su interior cada uno de los valores del array de forma secuencial y los mostrará mientras que queden ítems restantes que recorrer.

Para conseguir el índice se agrega entre paréntesis y separado por una coma de **a**, la palabra **index**, cuyo valor es posteriormente mostrado en {{index}}.

```
<li v-for = "(a, index) in items">{{index}}--{{a}}</li>
```

En **(a, index)**, **a** es el valor e **index** es la llave.

```
const ejemplo11 = Vue.createApp({
    data(){
        return{
            items: [],
            styleobj: {
                width: "30%",
                padding: "12px 20px",
                margin: "8px 0",
                boxSizing: "border-box"
            }
        }
    },
    methods: {
        showinputvalue(event) {
            this.items.push(event.target.value);
        }
    },
}).mount("#ejemplo11");
```

```
<div id="ejemplo11">
    <h3>Ingrese nombre de fruta y presione Enter</h3>
    <input type="text" v-on:keyup.enter="showinputvalue" v-bind:style="styleobj" placeholder="Ingrese nombre de fruta" />
    <h1 v-if="items.length>0">Frutas:</h1>
    <ul>
        <li v-for="(value, index) in items">{{index+1}} - {{value}}</li>
    </ul>
</div>
```