



## Clase 02

# Diseño y Programación Web

## Materia:

Aplicaciones para Dispositivos Móviles

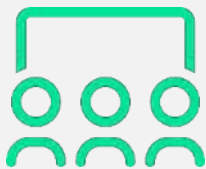
**Docente contenidista:** GARCIA, Mabel

**Revisión:** Coordinación

# Contenido

Condicionales.....	04
Directivas v-if, v-else-if, v-else.....	04
Diferencias entre v-show y v-if. ....	07
Iteraciones .....	08
Directiva v-for para iterar Arrays.....	10
Directiva v-for para iterar con Arrays de Objetos.....	13
Atributo key.....	14
v-for en combinación con v-if.....	16
Iterar estructuras complejas de datos .....	17
Bibliografía .....	22

# Clase 2



iTe damos la bienvenida a la materia  
**Aplicaciones para Dispositivos Móviles!**

**En esta clase vamos a ver los siguientes temas:**

- Directivas Condicionales (v-if, v-else.if, v-else).
- Diferencias entre v-show y v-if.
- Directiva v-for para iterar Arrays.
- Directiva v-for para iterar Arrays de Objetos.
- Iterar un objeto.
- Atributo key.
- v-for en combinación con v-if.
- Iterar estructuras complejas de datos.

# Condicionales

## Directivas v-if, v-else-if, v-else

Las directivas condicionales en Vue.js nos permiten renderizar o mostrar elementos HTML de forma condicional. Esto significa que podemos controlar qué se muestra en la interfaz de usuario según ciertas condiciones.

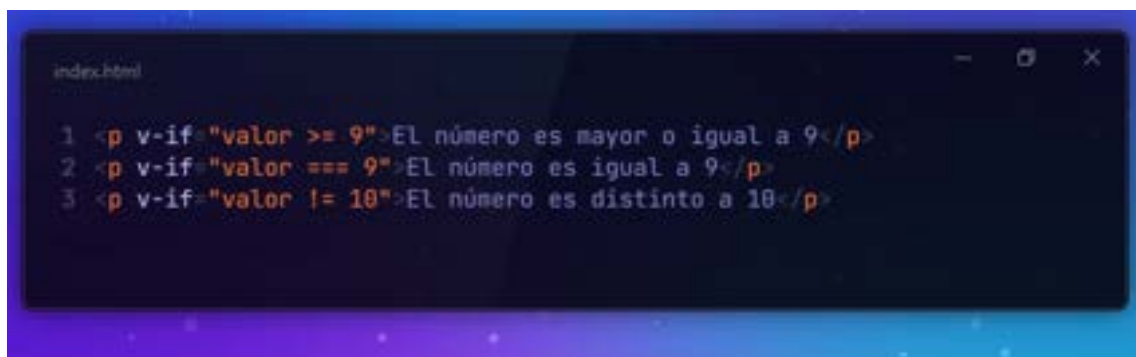
**v-if:** Esta directiva renderiza el elemento sólo si la expresión dada es verdadera. No sólo funciona con valores booleanos true/ false. Puede verificar si una propiedad de datos es igual a una cadena específica, o usar operadores comparativos de javascript.

Por ejemplo, si inicializamos la variable **"valor"** y su contenido es **9**, desde el HTML podemos tener distintas evaluaciones usando operadores javascript.



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       valor: 9
5     }
6   },
7 });
8 app.mount('#contenedor');
```

Podríamos establecer distintos párrafos evaluando esa variable **"valor"**:



```
index.html
1 <p v-if="valor >= 9">El número es mayor o igual a 9</p>
2 <p v-if="valor === 9">El número es igual a 9</p>
3 <p v-if="valor != 10">El número es distinto a 10</p>
```

*¿Dado este escenario qué párrafo se renderizará en el navegador?*

Tal y como ocurre cuando usamos javascript, todas las condiciones son verdaderas, por lo tanto Vue mostrará los 3 párrafos con sus respectivos textos.

**v-else:** Permite hacer un elemento alternativo basado en lo contrario de la declaración **v-if**. No tiene valor y se coloca dentro de la etiqueta de apertura de la etiqueta HTML.

En este caso, tenemos la variable "**mostrar**" inicializada con valor **false** desde nuestra lógica.

```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       mostrar: false,
5       valor: 9
6     }
7   },
8 });
9 app.mount('#contenedor');
```

El párrafo con **v-else** debe seguir directamente al que contiene **v-if**; de lo contrario, la consola arrojará error.

```
index.html
1 <p v-if="mostrar">Texto que se mostrará si la variable tiene valor
  TRUE</p>
2 <p v-else>Texto que se mostrará si la variable tiene valor FALSE</p>
```

Si tuviéramos una estructura intermedia, como por ejemplo otro párrafo, luego del párrafo con **v-if**, la consola nos arrojaría el siguiente error:

```
▲ [Vue warn]: Template compilation error: v-else/v-else-if has no adjacent v-if or v-
else-if.
7 |           <p v-if="mostrar">Texto que se mostrará si la variable tiene valor TRUE</p>
8 |           <p>párrafo intermedio que no debe existir</p>
9 |           <p v-else="">Texto que se mostrará si la variable tiene valor FALSE</p>
  |           ~~~~~
10 |
11 |         </div>
   at <App>
```

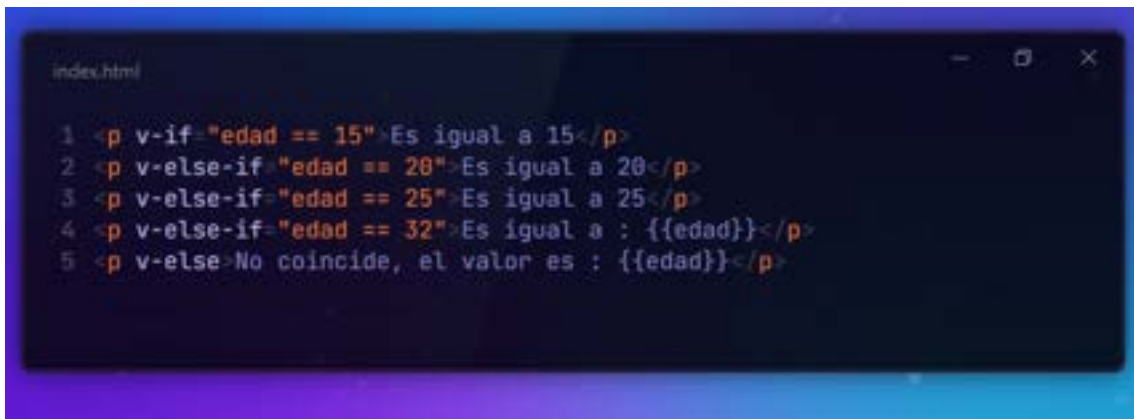
**v-else-if:** Se usa para establecer condiciones adicionales que se evalúan sólo si la condición anterior es falsa. Debe seguir inmediatamente a un elemento **v-if** o **v-else-if**.

Por ejemplo, si tuviéramos desde la lógica la variable **"edad"** con valor **32**, podríamos tener distintas evaluaciones desde nuestro HTML.



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       mostrar: false,
5       valor: 9,
6       edad: 32
7     }
8   },
9 });
10 app.mount('#contenedor');
```

De esta forma tendríamos distintas evaluaciones:

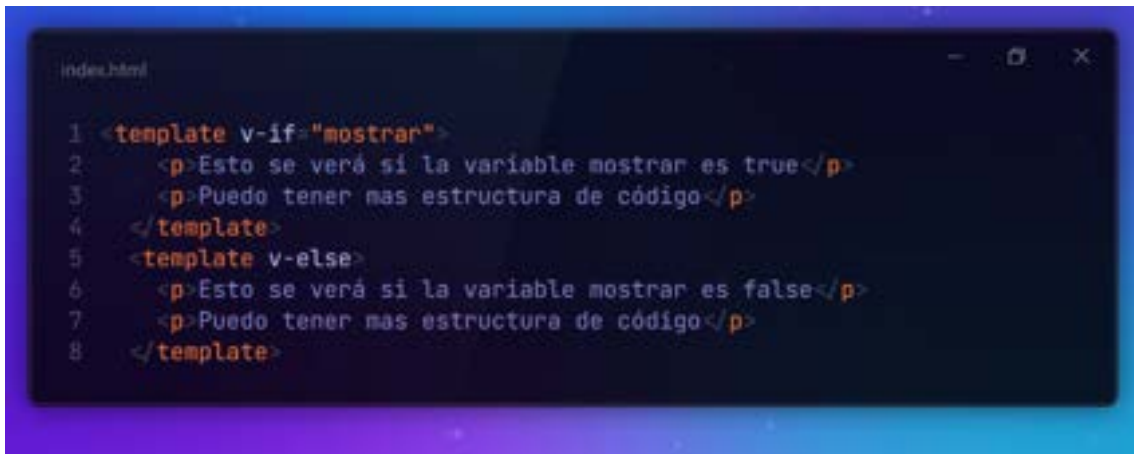


```
index.html
1 <p v-if="edad == 15">Es igual a 15</p>
2 <p v-else-if="edad == 20">Es igual a 20</p>
3 <p v-else-if="edad == 25">Es igual a 25</p>
4 <p v-else-if="edad == 32">Es igual a : {{edad}}</p>
5 <p v-else>No coincide, el valor es : {{edad}}</p>
```

El último **v-else-if** coincidiría con el valor **"32"** de la variable inicializada en la instancia root, por lo tanto, sería el único texto que se renderizaría en el navegador. La **interpolación** a la variable **"edad"**, nos mostraría cuál es su valor.

Vue también nos ofrece la posibilidad de englobar código dentro de un elemento llamado **template**. Como **v-if** es una directiva, tiene que estar adjunta a un solo elemento del HTML. ¿Pero qué pasa si queremos alternar más de un elemento? En ese caso, podemos utilizar **v-if** sobre un elemento **<template>**, que sirve como un envoltorio invisible. El resultado final del renderizado no incluirá el

elemento **<template>**. Hay que tener presente que puede usarse además con las directivas **v-else-if** y **v-else**.

A screenshot of a code editor window titled 'index.html'. The code is written in a dark theme with syntax highlighting. It shows two Vue.js template blocks. The first block is a <template v-if="mostrar"> containing two <p> elements. The second block is a <template v-else> containing two <p> elements. The code is as follows:

```
1 <template v-if="mostrar">
2   <p>Esto se verá si la variable mostrar es true</p>
3   <p>Puedo tener mas estructura de código</p>
4 </template>
5 <template v-else>
6   <p>Esto se verá si la variable mostrar es false</p>
7   <p>Puedo tener mas estructura de código</p>
8 </template>
```

*El uso de template nos va a permitir controlar muchas veces el **aspecto semántico del HTML**, ya que no genera etiquetas adicionales en nuestro código.*

## Diferencias entre v-show y v-if.

**"v-if"** es una representación condicional "real" porque garantiza que los detectores de eventos y los componentes secundarios dentro del bloque condicional **se destruyan y vuelvan a crear** adecuadamente durante los cambios.

En comparación, **"v-show"** es mucho más simple, el elemento siempre se representa independientemente de la condición inicial, con alternancia basada en CSS.

En términos generales, **"v-if"** tiene costos de alternancia más altos, mientras que **"v-show"** tiene costos de renderizado iniciales más altos. Por lo tanto, la documentación oficial nos recomienda usar **"v-show"** si necesitamos alternar algo con mucha frecuencia y elegir **"v-if"** si es poco probable que la condición cambie en tiempo de ejecución.

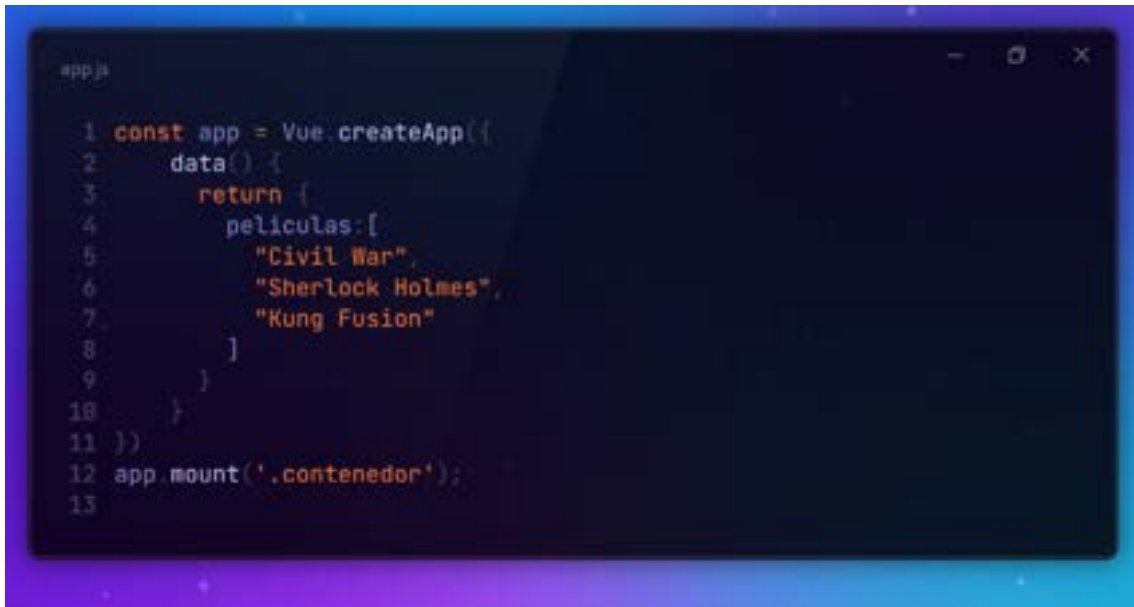


# Iteraciones

## Directiva v-for para iterar Arrays

La directiva **"v-for"** sirve para representar una **lista de elementos** basada en una **matriz**. Esta directiva requiere una sintaxis especial en forma de **"ítem in listadoDeElementos"**, donde **"listadoDeElementos"** será la matriz de datos a iterar e **"ítem"** es un **alias** para el elemento de la matriz sobre el que se itera.

Por ejemplo, en nuestra instancia root tendremos la función **data** que retornará un array llamado **"películas"** con 3 elementos.



```
1 const app = Vue.createApp({
2   data() {
3     return {
4       películas: [
5         "Civil War",
6         "Sherlock Holmes",
7         "Kung Fusion"
8       ]
9     }
10  }
11 })
12 app.mount('.contenedor');
```

En nuestro index.html iteramos con **"v-for"** esa lista de **"películas"** sobre la etiqueta **<li>**. Esto creará un elemento **<li>** por cada iteración del bucle. Dentro de esta etiqueta podemos tener un párrafo, o un span, por ejemplo.

Para que el usuario vea cada uno de los títulos, tenemos que mostrarlos por medio de la **interpolación** y haremos referencia al alias **"ítem"** usado para iterar cada **película**. El alias podría tener cualquier nombre, **"x"**, **"i"**, **"j"**, no es obligatorio que se denomine **ítem**.



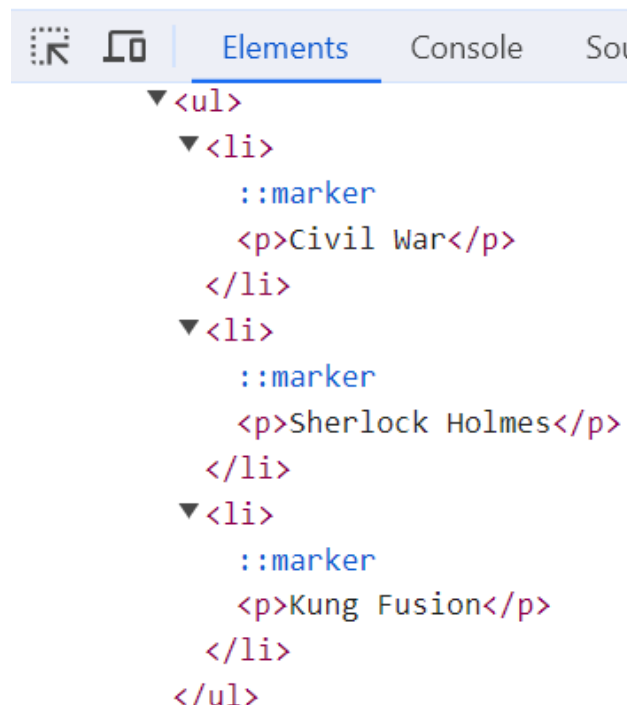
```
index.html
1 <ul>
2   <li v-for="item in peliculas">
3     <p>{{item}}</p>
4   </li>
5 </ul>
```

*Hay que tener presente que la iteración ocurre dentro del <li> y la interpolación debe estar entre la apertura y cierre de esa etiqueta.*

En el navegador veremos las interpolaciones de cada ítem que contiene el **array "películas"**.

## Iterando Arrays

- Civil War
- Sherlock Holmes
- Kung Fusion



Dentro de la consola vemos que se crearon cada uno de los list-items y párrafos acorde a la iteración.

# Directiva v-for para iterar con Arrays de Objetos

Dentro de los bloques “**v-for**” tenemos acceso completo a las propiedades del ámbito principal.

Por cada propiedad del objeto que se quiera mostrar se debe aclarar en la sintaxis del HTML.

En nuestro caso, tenemos un array llamado “**juegos**” y dentro tendrá 3 objetos con las propiedades: **nombre**, **anio**, **portada**, **alt** y **calificación**.

A screenshot of a code editor window titled 'app.js'. The code is written in JavaScript and uses Vue.js. It defines a Vue application with a data object containing an array of three game objects. Each object has properties: nombre, anio, portada, alt, and calificacion. The games are 'Super Mario Bros' (1986, rating 8), 'Mortal Kombat' (1993, rating 9), and 'Street Fighter' (1987, rating 7). The code ends with mounting the app to a container with the id 'contenedor'.

```
1 const app = Vue.createApp({
2   data() {
3     return {
4       juegos: [
5         {
6           nombre: "Super Mario Bros",
7           anio: 1986,
8           portada: "img/super_mario.jpg",
9           alt: "Super Mario Bros Primer juego",
10          calificacion: 8
11        },
12        {
13          nombre: "Mortal Kombat",
14          anio: 1993,
15          portada: "img/mk2.jpg",
16          alt: "Mortal Kombat 2",
17          calificacion: 9
18        },
19        {
20          nombre: "Street Fighter",
21          anio: 1987,
22          portada: "img/sf2.jpg",
23          alt: "Street Figther",
24          calificacion: 7
25        }
26      ],
27    }
28  });
29 app.mount('#contenedor');
```

Pero en nuestra vista sólo mostraremos: el **nombre**, el **año de lanzamiento** y la **imagen de portada** al hacer la iteración.

```
index.html
1 <ul>
2   <li v-for="item in juegos">
3     Nombre = <span>{{item.nombre}}</span>,
4     Lanzamiento = <span>{{item.anio}}</span>,
5     
6   </li>
7 </ul>
```

En el navegador veremos el resultado de las interpolaciones de estas propiedades.



Para mostrar las portadas, recordemos que necesitamos usar la directiva **"v-bind"** para el atributo **"src"** y para el atributo **"alt"** que están inicializados en nuestra lógica.

Hay que prestar atención al **nombre de las propiedades** que solicitamos en las interpolaciones. Si nos equivocamos al tipear o poner mayúsculas donde va minúsculas tendremos error de consola y no se mostrarán las propiedades. Estamos trabajando con **javascript** y es **case sensitive**.

Recordemos que tanto el modelo como la vista deben referenciar a los mismos elementos. Caso contrario, **la consola nos mostrará error**.

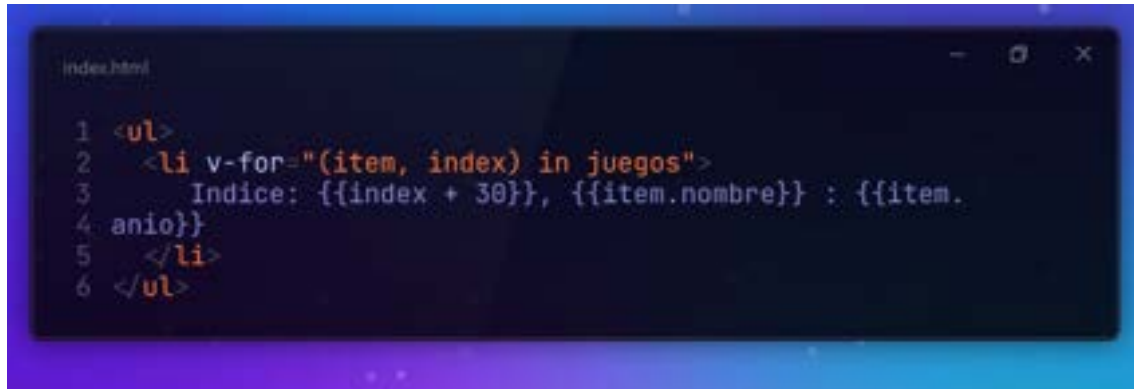
La directiva **"v-for"** admite un segundo argumento opcional para obtener el **índice del elemento actual**.

En tal caso, la sintaxis de la iteración se agrupa entre paréntesis. Al igual que sucede con javascript el primer objeto tendrá el índice 0 y el segundo objeto el índice 1 por defecto.

Si queremos **modificar el valor del índice**, las **interpolaciones pueden procesar operaciones aritméticas**.

Podríamos indicarle que el primer objeto inicie en un número determinado.

Por ejemplo:




```
index.html
1 <ul>
2   <li v-for="(item, index) in juegos">
3     Indice: {{index + 30}}, {{item.nombre}} : {{item.
4     anio}}
5   </li>
6 </ul>
```

En el navegador veremos que la interpolación se procesa e inicia en el número 30.

- Indice: 30, Super Mario Bros : 1986
- Indice: 31, Mortal Kombat : 1993
- Indice: 32, Street Fighter : 1987

Se Puede usar **"of"** como delimitador en lugar de **"in"**, de modo que esté más cerca de la sintaxis de JavaScript para los iteradores:



```
index.html
1 <li v-for="(item, index) of juegos">Indice: {{index + 30}},
  {{item.nombre}} : {{item.anio}}</li>
```

# Iterar un Objeto

También se puede usar la directiva **"v-for"** para iterar a través de las propiedades de un objeto.

El orden de iteración se basará en el resultado de llamar a **Object.keys()** en el objeto. Es decir, que el **orden se basará en cómo fue escrito originalmente el objeto en nuestro código**.

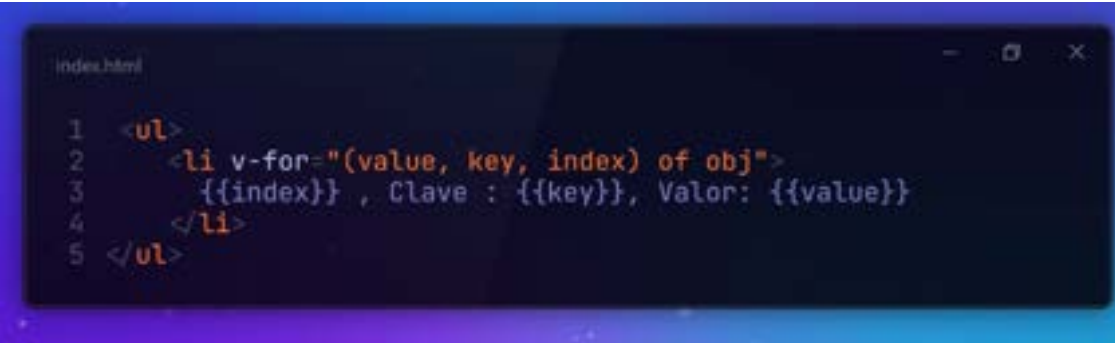
En caso de querer mostrar las **propiedades** de ese objeto, se puede pasar como segundo argumento un **key**. Y si además queremos mostrar el **índice**, éste será el tercer argumento agrupado por paréntesis en el **"v-for"**.

En nuestro archivo **app.js** tenemos el siguiente código:



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       obj: {
5         animal: "gato", raza: "Persa", edad: 4
6       }
7     }
8   }
9 });
10 app.mount('.contenedor');
```

En nuestro **index.html**, tenemos el siguiente código:



```
index.html
1 <ul>
2   <li v-for="(value, key, index) of obj">
3     {{index}} , Clave : {{key}}, Valor: {{value}}
4   </li>
5 </ul>
```

En el navegador mostrará:

- 0 : animal : gato
- 1 : raza : Persa
- 2 : edad : 4

# Atributo key

Desde la documentación de Vue se recomienda siempre el uso de este atributo **"key"** cuando las iteraciones pueden ser complejas, o se usen componentes o elementos del DOM que puedan reordenarse, ser reutilizados o ser modificados al iterar con la directiva **"v-for"**.

El uso de este atributo sirve para aprovechar el rendimiento de Vue y le permite rastrear correctamente los elementos de la iteración.

Este **atributo key debe ser único** y los valores que admite son de tipo **string o numéricos**. Nunca debe ser un objeto.

Para usar este atributo necesitamos de la directiva **"v-bind"** cuando usamos la directiva **"v-for"**.

Por ejemplo, en nuestro archivo app.js tendremos:



```
1 const app = Vue.createApp({
2   data() {
3     return {
4       lista: [
5         {tarea: "Practicar Vue", id: 1, estado: true},
6         {tarea: "Leer los apuntes", id: 2, estado: true},
7         {tarea: "Comprar plantas", id: 3, estado: false},
8         {tarea: "Pintar", id: 4, estado: true},
9       ]
10    }
11  },
12 });
13 app.mount('.contenedor');
```

Desde nuestra **vista**, nuestro archivo **index.html** haremos la iteración agregando el atributo **key** usando la forma abreviada para **"v-bind"** (los dos puntos).

Si bien el objeto tiene varias propiedades, sólo nos interesa mostrarle al usuario los valores de la propiedad **"tarea"**.



```
1 <ul>
2   <li v-for="item in lista" :key="item.id">
3     {{item.tarea}}
4   </li>
5 </ul>
```



En el navegador se mostrarán todas las tareas:

- Practicar Vue
- Leer los apuntes
- Comprar plantas
- Pintar

También podríamos aprovechar el uso de **v-bind** para manipular su estilo visual con CSS en base al **estado true o false de la tarea**.

Usando **operador ternario en Vue con sintaxis de objeto**.

En nuestro HTML podríamos representarlo de la siguiente forma:



```
1 <ul>
2   <li v-for="tarea in lista"
3     :key="tarea.id"
4     :class="{ 'tarea-completada': tarea.estado,
5             'tarea-pendiente': !tarea.estado }">
6     {{ tarea.tarea }}
7   </li>
8 </ul>
```

La sintaxis establece el uso de 2 clases CSS:

**"tarea-completada"**: Si **tarea.estado** es **true** (completada), se aplica esta clase. Esto tacha el texto y le da un color verde.

**"tarea-pendiente"**: Si **tarea.estado** es **false** (pendiente), se aplica esta clase. Esto le da font-weight en bold y un color rojo.

**Operador lógico NOT (!)**: El operador **!** se utiliza para negar una condición. En este caso, **!tarea.estado evalúa a true cuando el estado de la tarea es false, lo que significa que la tarea está pendiente**.

Ambas clases CSS, deben estar desarrolladas en la hoja de estilos CSS.

En el navegador se mostrará así:

- ~~Practicar Vue~~
- ~~Leer los apuntes~~
- **Comprar plantas**
- ~~Pintar~~



## v-for en combinación con v-if

Hay que tener especial cuidado cuando queremos realizar una iteración con **"v-for"** y establecer el uso de **"v-if"** para hacer una evaluación del valor de una propiedad.

Según la documentación de Vue, no se recomienda usar **"v-if"** y **"v-for"** en la misma etiqueta HTML debido a la precedencia implícita.

Cuando coexisten estas directivas en el mismo nodo, **"v-if"** tiene una prioridad más alta que **"v-for"**. Esto significa que la condición de **"v-if"** no tendrá acceso a las variables del ámbito de **"v-for"** y además arrojará error de consola.

Desde el código esto se soluciona muy fácil moviendo la directiva **"v-for"** a una etiqueta **"<template>"**.

Aprovechando el código anterior de la lista, vamos a evaluar que únicamente nos muestre las tareas que tengan en su propiedad el valor **"false"**.

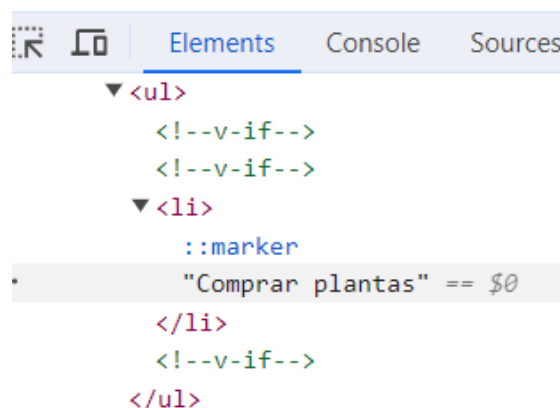
Nuestro index tendrá :



```
1 <ul>
2   <template v-for="item in lista" :key="item.id">
3     <li v-if="!item.estado"> {{ item.tarea }}</li>
4   </template>
5 </ul>
```

Por lo tanto, en el navegador veremos una tarea: **"Comprar plantas"** y en la consola veremos como Vue ha removido los nodos del DOM que tenían el valor del estado en **true**.

- Comprar plantas



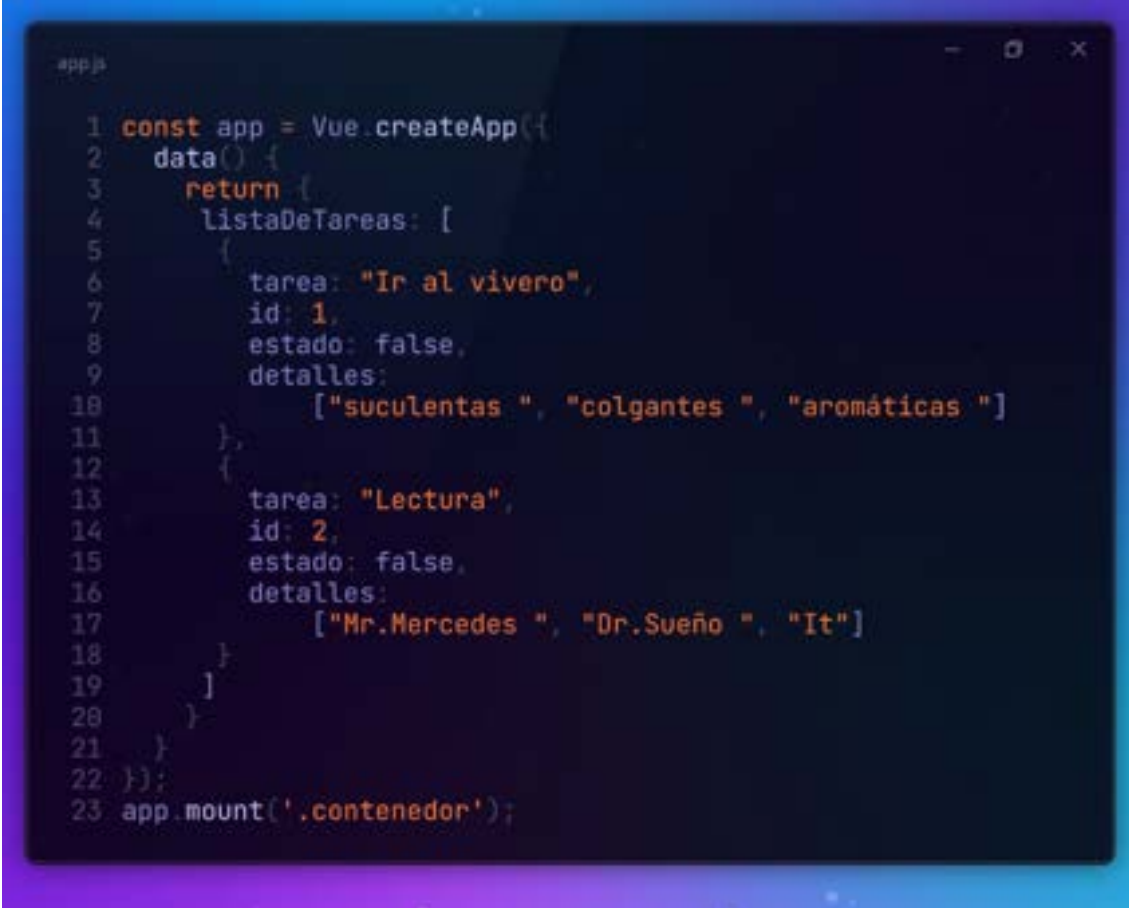
# Iterar estructuras complejas de datos

En varios escenarios nos encontramos con situaciones donde la simplicidad de una estructura de datos básica no es suficiente para representar la complejidad de nuestra información. En estos casos, se vuelve necesario usar estructuras de datos más elaboradas.

Tomemos como ejemplo el código anterior, donde manejamos una lista de tareas que no solo incluyen el **nombre** de la tarea y su **estado**, sino también **detalles** adicionales en **formato de Array**.

Tal y como ocurre en Javascript, tendremos que realizar dos iteraciones para poder acceder a estos detalles.

En nuestra lógica tendremos la siguiente estructura de datos:



```
1 const app = Vue.createApp({
2   data() {
3     return {
4       listaDeTareas: [
5         {
6           tarea: "Ir al vivero",
7           id: 1,
8           estado: false,
9           detalles:
10             ["suculentas ", "colgantes ", "aromáticas "]
11         },
12         {
13           tarea: "Lectura",
14           id: 2,
15           estado: false,
16           detalles:
17             ["Mr.Mercedes ", "Dr.Sueño ", "It"]
18         }
19       ]
20     }
21   }
22 });
23 app.mount('.contenedor');
```

En nuestro index tendremos:

```
index.html
1  <ul>
2  <!-- Itera sobre cada tarea en la listaDeTareas -->
3  <li v-for="item in listaDeTareas" :key="item.id" >
4    {{item.tarea}} :
5
6  <!-- Itera sobre los detalles de la tarea -->
7    <span v-for="x in item.detalles">
8      Detalles: {{x}} -
9    </span>
10 </li>
11 </ul>
```

Para ir repasando muchos de los conceptos que vimos en esta clase, recordemos lo que está haciendo cada elemento empleado:

- La directiva **v-for** se utiliza para iterar sobre cada elemento en **listaDeTareas**.
- La propiedad **:key="item.id"** se utiliza para proporcionar una clave única a cada elemento de la lista para mejorar la eficiencia de Vue en el seguimiento de los cambios.
- Se muestra el nombre de la tarea con la interpolación **{{ item.tarea }}**.
- Se usa otra iteración **v-for="x in item.detalles"** para recorrer los detalles de cada tarea.
- Cada detalle se muestra con el alias **{{ x }}**

Como resultado el navegador mostrará:

- Ir al vivero : Detalles: suculentas - Detalles: colgantes - Detalles: aromáticas -
- Lectura : Detalles: Mr.Mercedes - Detalles: Dr.Sueño - Detalles: It -

En el caso de que estos detalles tuvieran, por ejemplo, una **estructura de objetos** dentro de un elemento, el proceso es muy similar.

Debemos generar 2 iteraciones para poder mostrar la información de forma adecuada.

En nuestra lógica tendríamos:

```
#PP2#
1  tareasConElementos:
2  [
3  {
4    tarea: "Ir al vivero",
5    id: 1,
6    estado: false,
7    elementos: [
8      {
9        nombre: "suculentas",
10       cantidad: 4,
11       tamaño: "pequeño",
12     },
13     {
14       nombre: "colgantes",
15       cantidad: 10,
16       tamaño: "mediano",
17     },
18   ]
19 },
20 {
21   tarea: "Comprar estantes",
22   id: 2,
23   estado: false,
24   elementos: [
25     {
26       nombre: "Estanteria de madera",
27       cantidad: 2,
28       tamaño: "grande",
29     },
30     {
31       nombre: "Soportes metálicos",
32       cantidad: 4,
33       tamaño: "mediano",
34     },
35   ]
36 },
37 ]
```

En nuestro archivo HTML se representaría de la siguiente forma:

```
index.html
1 <ul>
2   <li v-for="item in tareasConElementos" :key="item.id" >
3     {{item.tarea}}
4     <span v-for="x in item.elementos">
5       {{x.nombre}} : {{x.cantidad}} {{x.tamano}}
6     </span>
7   </li>
8 </ul>
```

La única gran diferencia radica en que, al emplear el segundo “**v-for**” en “**item.elementos**”, nuestro alias para el iterador debe ir seguido de **cada propiedad del objeto** que deseamos visualizar.

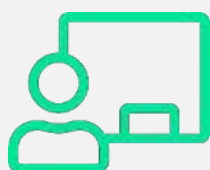
*¡Ya estamos listos para trabajar con condicionales e iteraciones!*

*¡Practiquemos lo aprendido!*



Hemos llegado así al final de esta clase en la que vimos:

- Directivas Condicionales (v-if, v-else.if, v-else).
- Diferencias entre v-show y v-if.
- Directiva v-for para iterar Arrays.
- Directiva v-for para iterar Arrays de Objetos.
- Iterar un objeto.
- Atributo key.
- v-for en combinación con v-if.
- Iterar estructuras complejas de datos.



Te esperamos en la **clase en vivo** de esta semana.  
No olvides realizar el **desafío semanal**.

**¡Hasta la próxima clase!**

# Bibliografía

Vue.js, Documentación Oficial. (s.f.). Condicionales. Recuperado de <https://vuejs.org/guide/essentials/conditional.html>

Vue.js, Documentación Oficial. (s.f.). Iteraciones. Recuperado de <https://vuejs.org/guide/essentials/list.html>