



# Renderizado condicional

A continuación vamos a ver las directivas condicionales, empezamos trabajando con v-if. Vamos a hacer que aparezca un mensaje si el motor está encendido.

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <p v-if="engineStarted">Feliz viaje</p>
5
6
7
8   <button @click="engineStarted = true"> Arrancar el motor</button>
9   <button @click="engineStarted = false"> Parar el motor</button>
10  <pre>{{ $data }}</pre>
11 </div>
12
13
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     engineStarted: false,
5   },
6 });
7
```

Arrancar el motor Parar el motor  
{  
 "engineStarted": false  
}

Se evalúa la variable y mostrará el mensaje o no. Pero si lo que queremos es añadir varios elementos no es viable realizarlo. Para ello utilizaremos un tag que se llama template.

```
<template v-if="engineStarted">
  <h1>Buen viaje</h1>
  <h2>Buen viaje</h2>
</template>
```

Utilizando la estructura de programación if, ahora podemos ampliarlo con el else.

```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <p v-if="engineStarted">Feliz viaje</p>
5   <hr>
6   <p v-if="fuelAmount > 250">Tu depósito está lleno</p>
7   <p v-else-if="fuelAmount < 90">Tu depósito está casi vacío</p>
8   <p v-else>Tu depósito está a la mitad</p>
9
10
11
12   <button @click="fuelAmount = 300"> 300 l</button>
13   <button @click="fuelAmount = 150"> 150 l</button>
14   <button @click="fuelAmount = 40"> 40 l</button>
15   <button @click="engineStarted = true"> Arranca</button>
16   <button @click="engineStarted = false"> Para</button>
17   <pre>{{ $data }}</pre>
18 </div>
19
```

```
CSS
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 .roundCorners {
21   border-radius: 10px;
22 }
23
24 button {
25   height: 100px;
26   width: 100px;
27 }
```

```
JS
1 new Vue({
2   el: '#app',
3   data: {
4     engineStarted: false,
5     fuelAmount: '300',
6   },
7 });
8
```

depósito está casi vacío

300 l	150 l	40 l	Arranca	Para
-------	-------	------	---------	------

engineStarted: false,  
fuelAmount: 40

# Show

Ahora vamos a ver esta directiva, funciona como v-if, evalúa una propiedad y muestra u oculta el contenido, no elimina el contenido del DOM sino que lo hace con la propiedad de css display:none.

```
<p v-show="fuelAmount > 250">Tu depósito está lleno</p>
```

Con v-if cuando lo muestra habría que volver a renderizar, mientras que con v-show no, es más un tema de rendimiento que de funcionalidad porque las dos directivas sirven para lo mismo.

# Claves

Cuando utilizamos el renderizado condicional, es mucho más efectivo, pero vamos a ver la siguiente situación:

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4
5   <template v-if="switchInputs === 'A'">
6     <input placeholder="Añade un texto"/>
7   </template>
8   <template v-else>
9     <input placeholder="Añade un email"/>
10  </template>
11
12
13
14  <button @click="switchInputs = 'A'">Mostrar A</button>
15  <button @click="switchInputs = 'B'">Mostrar B</button>
16  <pre>{{data}}</pre>
17 </div>
18
19
20
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     switchInputs: 'A',
5   },
6 });
7
```

Añade un texto    Mostrar A    Mostrar B

"switchInputs": "A"

Cuando yo alterne entre botones veré que la variable cambia de valor, aunque yo escriba algo en la caja de texto y parezca que no, pero la variable sí que cambia.

Añadiendo las keys ahora re-renderizará los campos cada vez que cambie de uno a otro input y no reutilizará el que hay.

```

<template v-if="switchInputs === 'A'">
  <input placeholder="Añade un texto" key="campoTexto"/>
</template>
<template v-else>
  <input placeholder="Añade un email" key="email"/>
</template>

```

## Interactuando con el DOM

La directiva v-for nos va a servir para hacer recorridos como si fuera un bucle for de programación.

```

HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4
5 <h1> Lista de la compra</h1>
6 <ul>
7 <li v-for="product in shopList">{{product}}</li>
8 </ul>
9 </div>
10
11
JS
1 new Vue({
2   el: '#app',
3   data: {
4     shopList: ['Manzanas', 'Peras', 'Sandías'],
5   },
6 });
7

```

### Lista de la compra

Manzanas  
Peras  
Sandías

Por cada elemento del array nos los va a mostrar en un <li>. Si ahora quisiera recorrer objetos.

```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4
5 <h1> Lista de la compra</h1>
6 <ul>
7 <li v-for="product in shopList">{{product}}</li>
8 </ul>
9 <hr>
10 <ul>
11 <li v-for="misuper in superMarkets"> {{misuper.name}}
12 </li>
13 </ul>
14 <hr>
15 {{data}}
16 </div>
17
18

CSS

JS
1 new Vue({
2   el: '#app',
3   data: {
4     shopList: ['Manzanas','Peras','Sandias'],
5     superMarkets: [
6       {name: 'Lidl',streets: ' Calle numero 1'},
7       {name: 'Carrefour',streets: ' Calle numero 2'},
8       {name: 'Amazon',streets: ' Calle numero 3'}
9     ]
10  },
11  });
12
```

## Lista de la compra

ananas  
ras  
ndías  
dl  
rrefour  
nazon

shopList: [ "Manzanas", "Peras", "Sandías" ], "superMarkets": [ { "name": "Lidl", "streets": " Calle numero 1" }, { "name": "Carrefour", "streets": " Calle numero 2" }, { "name": "Amazon", "streets": " Calle n

Ahora si queremos iterar sobre diferentes elementos utilizaremos el template para mejorar la visualización.

```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4
5 <h1> Lista de la compra</h1>
6 <ul>
7 <li v-for="product in shopList">{{product}}</li>
8 </ul>
9 <hr>
10 <ul>
11 <template v-for="misuper in superMarkets">
12 <li> <h1>{{misuper.name}}</h1><p>{{misuper.streets}}
13 </li>
14 </template>
15 </ul>
16 <hr>
17 {{data}}
18 </div>
19

CSS

JS
1 new Vue({
2   el: '#app',
3   data: {
4     shopList: ['Manzanas','Peras','Sandias'],
5     superMarkets: [
6       {name: 'Lidl',streets: ' Calle numero 1'},
7       {name: 'Carrefour',streets: ' Calle numero 2'},
8       {name: 'Amazon',streets: ' Calle numero 3'}
9     ]
10  },
11  });
12
```

Ahora vamos a aprender a sacar el índice y la key de cada uno de ellos.

```
1
2
3
4
5 <h1> Lista de la compra</h1>
6
7
8 <ul>
9 <li v-for="(misuper, index) in superMarkets">
10   {{index}} - {{misuper.name}}</li>
11 </ul>
12
13
14 </div>
15
16
17
18
```

Este índice empieza desde cero, si quisiera que empezase en el uno no tendría que añadir +1. `{index+1}`. Para acceder a la clave ésa siempre tiene que ser el segundo parámetro. Es hacer un for dentro de for, recorrer el objeto que hay en cada posición de los supermercados.

```
1- new Vue({
2-   el: '#app',
3-   data: {
4-     shopList: ['Manzanas','Peras','Sandias'],
5-     superMarkets: [
6-       {name: 'Lidl',streets: ' Calle numero 1'},
7-       {name: 'Carrefour',streets: ' Calle numero 2'},
8-       {name: 'Amazon',streets: ' Calle numero 3'}
9-     ]
10-  },
11- });
```

```
2- <div id="app">
3-   <h1> Lista de la compra</h1>
4-   <ul>
5-     <li v-for="(misuper, index) in superMarkets">
6-       <div v-for="(value,key,index) in misuper">
7-         {{value}} / {{key}} - {{index}}
8-       </div>
9-     </li>
10-   </ul>
11- </div>
```

### Lista de la compra

- Lidl / name - 0  
Calle numero 1 / streets - 1
- Carrefour / name - 0  
Calle numero 2 / streets - 1
- Amazon / name - 0  
Calle numero 3 / streets - 1

## Iterar números

```
<span v-for="number in 10">{{number}}</span>
```

Vue nos permite hacer acciones sobre los arrays, mediante push añadiré un equipo de fútbol al array. Realmente no interactúa con el DOM, éste se vuelve a recargar. Con arrays sencillos funciona bien, pero con más complejos puede acarrear ciertos problemas, para ello tendríamos que añadir el key.

Ahora quisiera devolver aquellos que tengan de pobla < 200. Para ello crearemos una computed property que será la que recorreremos en el v-for y dentro de ella meteremos la condición.

```
HTML
2
3 - <div id="app">
4
5 - <ul>
6 -   <li v-for="team in filterCities">
7     {{team.name}} [{{team.id}}] [{{team.pobla}}]
8     <input type="text">
9   </li>
10 </ul>
11 </div>
12
13
CSS
JS
5 - soccerTeams: [
6   {id: '1', name: 'Sevilla', pobla: '500'},
7   {id: '2', name: 'Granada', pobla: '100'},
8   {id: '3', name: 'Levante', pobla: '150'},
9   {id: '4', name: 'Valladolid', pobla: '300'},
10 ]
11 },
12 computed: {
13   filterCities: function() {
14     return this.soccerTeams.filter(function(team) {
15       return team.pobla < 200;
16     });
17   }
18 }
19
20 });
21
```

- Granada [2] 100
- Levante [3] 150

# Ejercicio TO-DO

Vamos a realizar una lista de tareas, donde añadir tareas, verlas, eliminarlas ...

Lo primero es realizar una cabecera para ello partiremos de un proyecto base llamado vue1.zip. A partir de aquí empezamos con la cabecera y en primer lugar con la fecha.

Para esto añadimos este código para seguir una nomenclatura común:

```
<div id="app">
  <main class="vue-Todo">    <section class="vue-Todo_Header">
    <p class="vue-Todo_Header">
      {{todoDate}}
    </p>
  </section>
  <p class="vue-Todo_HeaderTaskCounter">
    Tienes {{taskList.length}} tareas
  </p>
</main>
</div>
```

Y en el js:

```
var setDate = new Date(),
    locale = "es-ES",
    dateOptions = {weekday: 'long', month: 'long', day:'numeric'}
    readableDate = setDate.toLocaleDateString(locale, dateOptions);

new Vue({
  el: '#app',
  data: {
    text:'Hola Mundo',
    todoDate: readableDate,
  },
});
```

Con esto lo que hacemos es coger la fecha y formatearla para luego mostrarla en la cabecera.

Después del main añadiremos

```
</main>
  <pre>{{ $data }}</pre>
</div>
```

para ver los datos y nuestro array que acabamos de añadir en el js:

```
taskList:[
  'Hacer la compra',
  'Pasear al perro',
  'Llevar el coche al taller',
  'Comprar ropa',
],
```

Y en la hoja de estilos:

```
.vue-Todo {
  background: white;
  border-radius: 5px;
```



```

overflow: hidden;
margin: 0 auto;
width: 60%;
min-width: 400px;
max-width: 600px;
border: 1px black solid;
}
.vue-Todo_HeaderTaskCounter {
border-radius: 5px;
background: grey;
color: white;
width: 150px;
padding: 5px;
}

```

Con esto ya tenemos la fecha y el número de tareas, ahora debería mostrar un aspecto muy parecido a este:



```

{
  "todoDate": "miércoles, 8 de abril",
  "taskList": [
    "Hacer la compra",
    "Pasear al perro",
    "Llevar el coche al taller",
    "Comprar ropa"
  ]
}

```

Ahora añadiremos un formulario y en él un botón que empleará una clase del css de materialize.

```

<button class="btn-floating orange waves-effect waves-light"
  <i class="material-icons">add</i>
</button>

```

Con el siguiente css:

```
.vue-Todo_Header {
  position: relative;
  background: url('https://source.unsplash.com/collection/158643/') center center no-repeat;
  border-radius: 5px 5px 0 0;
  padding: 1em;
  min-height: 200px;
}

.vue-Todo_Header .btn-floating {
  position: absolute;
  bottom: -20px;
  right: 20px;
}

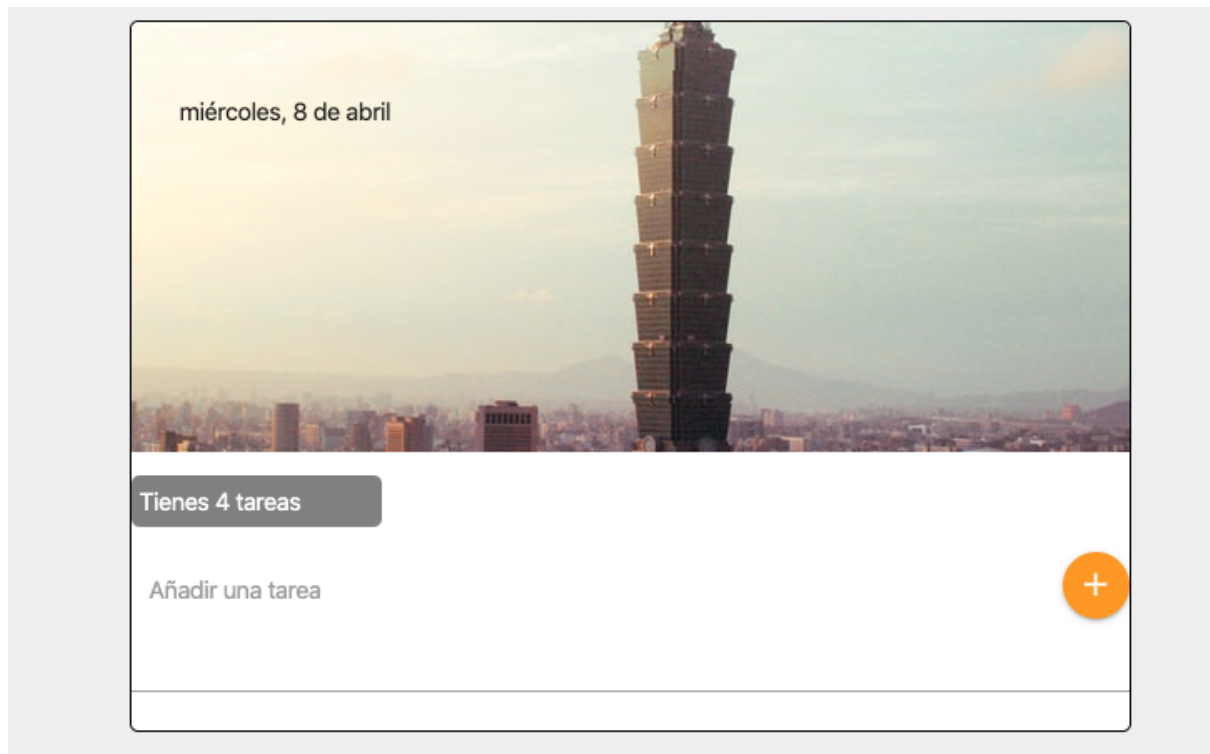
button {
  position: relative;
  float: right;
}

.vue-Todo_HeaderTaskCounter {
  border-radius: 5px;
  background: grey;
  color: white;
  width: 150px;
  padding: 5px;
}
```

Y el html añadido al form:

```
<form>
  <button class="btn-floating orange waves-effect waves-light">
    <i class="material-icons">add</i>
  </button>
  <div class="vue-Todo_AddTaskInput input-field col s6">
    <input
      id="input_task"
      type="text"
      class="validate"
      v-model="taskItem"
    >
    <label for="input_task">Añadir una tarea</label>
  </div>
</form>
```

Se debería mostrar más o menos así:

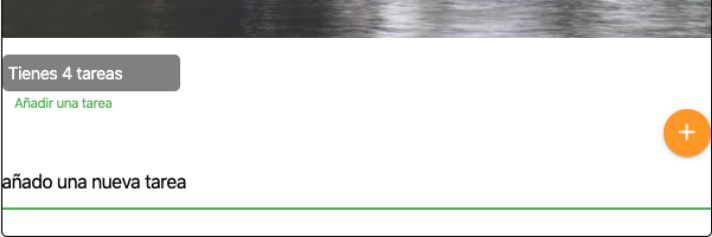


Ahora si aprieto al botón, éste no hace nada sólo recarga la página. Para evitar esta recarga de la página incluimos: `<form @submit.prevent>` y al botón `@click="addTask"`

Y en el js:

```
'Comprar ropa',
],
taskItem:"",
},
methods:{
  addTask: function(){
    this.taskList.push(this.taskItem);
  }
}
});
```

Así lo que logramos es que cuando escribamos en el input el taskItem se va rellenando, y cuando le demos al pulsar el botón éste se añade a la lista, para eso tenemos \$data para visualizar como se van actualizando los datos.



```
{
  "todoDate": "miércoles, 8 de abril",
  "taskList": [
    "Hacer la compra",
    "Pasear al perro",
    "Llevar el coche al taller",
    "Comprar ropa"
  ],
  "taskItem": "añado una nueva tarea"
}
```

Ahora falta tener en cuenta que no hayan elementos vacíos.

A continuación veremos como hacer un título editable:

```
<ul class="collection with-header">
  <li class="collection-header">
    <input v-if="isEditing"
      v-bind:value="todoTitle"
    >
    <h3 v-
      @click="isEditing = true"
    >{{todotitle}}</h3>
  </li>
```

Y en el JS:

```
todotitle: 'Mi lista de Tareas',
isEditing: false,
```

Con este código accederemos al título del js y lo podremos hacer editable haciendo click en él.



Bien, ahora sólo faltaría mostrar la lista de tareas, para ello añadiremos un v-for:

```

<li v-for="task in taskList" class="collection-item">
  <p class="collection-item_text">
    {{task}}
  </p>
</li>

```

Y ahora sólo falta eliminarlos, para ello en el botón eliminar crearemos el evento click y le pasaremos el índice a eliminar.

```

<button
  class="vue-Todo_Button-remove"
  @click="removeTodo(index)"
  >
  Eliminar</button>
</li>

```

Y en el js crearemos un método que elimine mediante el método splice el que tenemos seleccionado.

```

removeTodo: function (index) {
  this.taskList.splice(index,1);
}

```

Ahora sólo faltaría un mensajito indicando cuando no hay tareas y que nos redirige a crear una nueva.

```

<div
  v-if="taskListEmpty"
  class="vue-Todo-empty"
  @click="addTaskFocus"
  >
  <p>No hay tareas:<a> Añade una nueva</a></p>
</div>

```

y en el js

```

addTaskFocus: function(){

```

```
this.$refs.taskInput.focus();  
}
```