Vue.js for Beginners (Part 4)

Welcome back!

Last time we took at conditional rendering with v-if and v-show. This time we will learn how to loop through arrays and objects and create an element for each one of the items in them. We will also apply some of the concepts we have learned before.

v-for

`v-for` is one of the fundamental directives of *Vue.js*, and once you understand how it works the extension of what you can build inside your apps will grow exponentially.

`v-for` is, simply put, a for loop. If you don't yet know what this means, a for loop is a piece of code that gets executed one time per each element in a group - which in turn is usually an `Array` or an `Object`.

We're going to start with an empty slate today so that everything we do has a clear purpose. Here's a copy of our base `index.html` file for you to copy and paste into your editor.

```
<html>

<head>
  <title>Vue 101</title>
</head>

<body>
  <div id="app">

  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

  <script>
    const app = new Vue({
      el: '#app',
      data: {

      },
      methods: {

      }
    });
  </script>
</body>

</html>
```

Let's start by creating a simple list, an array, that we can loop to output its content. We will create a property inside our data object, called *games*. Feel free to change the titles to your own personal favorites 🙃

```
data: {
  games: [
    'Super Mario 64',
    'The Legend of Zelda Ocarina of Time',
    'Secret of Mana',
    'Super Metroid'
  ]
},
```

Awesome! Now that we have our array set up, let's create a sad and simple `<ul>` element where will display it. For the sake of example, let's keep it simple for now.

```
<div id="app">
  <ul>
    <li>Game title here</li>
  </ul>
</div>
```

Ok, looking good! Now we have to tell *Vue* that we want to output as many `<li>` elements inside the `<ul>` as needed to loop through our whole array.

In other languages, and even in vanilla JavaScript, you may be used to doing something that looks similar to this:

```
<?php foreach ($game in $games): ?>
  <li><?php echo $game; ?></li>
<?php endforeach; ?>
```

Where the loop *encloses* the element(s) it's going to output or print out.

In Vue we declare our v-for directive on TOP of the element we want to loop. Make these changes to your `<li>` and we'll dissect them after.

```
<ul>
  <li v-for="game in games">{{ game }}</li>
</ul>
```

Let's take a look.

1. `v-for` was added directly to the `<li>`, not the `<ul>` as we saw earlier. This reads: "For each game in my games array, please make a new `<li>` inside these `<ul>` tags.
2. Note that `games` is the property that we added earlier with the array inside our `data`, so we have to use this variable name.
3. The variable `game` (singular) is defined by us, we could use item, game, title or whatever we feel like. But be sure to understand that this `*game*` in `games` is what you will be using as a variable inside your loop.
4. Finally, inside our `<li>` tag we're outputting the contents of our game variable, so while the loop is running for each of our games, this will output the string into the `<li>`.

Run your app inside the browser, and you should see your list of items being outputted to the screen.

Taking it up a notch

So far, so good? `v-for` is actually a very simple concept, and this example is super boring. So how about we make things a bit more complicated, by making our array include some objects, and also applying some `v-ifs` inside our list?

First things first, let's update our `games` property with some more interesting data.

```
data: {
  games: [
    { name: 'Super Mario 64', console: 'Nintendo 64', rating: 4 },
    { name: 'The Legend of Zelda Ocarina of Time', console: 'Nintendo 64', rating: 5 },
    { name: 'Secret of Mana', console: 'Super Nintendo', rating: 4 },
    { name: 'Fallout 76', console: 'Multiple', rating: 1 },
    { name: 'Super Metroid', console: 'Super Nintendo', rating: 6 }
  ]
},
```

As always feel free to use your own favorite titles. PS. Super Metroid's rating of 6 is not a typo, it's just THAT good - and I'm biased. 😆 Also, Bethesda, you should be ashamed. *cough* Anyways.

If you run your app this point it will not particularly break, but it will just output the objects in a string format, which is not pretty. In fact, we're going to scratch our `<ul>` approach completely, and use a `<div>` to output our information. (Don't worry, it'll still be ugly).

Update your whole `<div id="app">`:

```
<div id="app">
  <div v-for="game in games">
    <h1>{{ game.name }} - <small>{{ game.console }}</small></h1>

    <span v-for="star in game.rating">❤️</span>

    <div v-if="game.rating > 5">Wow, this game must be <b>REALLY</b> good</div>
  </div>
</div>
```

WOAH. Ok, maybe not, but don't worry, you already know everything you need to understand what's happening here.

1. `div v-for="game in games"` Same old, we're going to loop our games array prop and store each game in the game variable.
2. `h1`. Ok, so `game` is an object, which in turn holds its own properties, name, console and rating. Inside the `<h1>` we're going to output the game's *name: `game.name`. And the console: `game.console`. As you can see now, `v-for` is not limited to outputting just a single element as we saw before

with the li, but you can actually output as much HTML as you need.

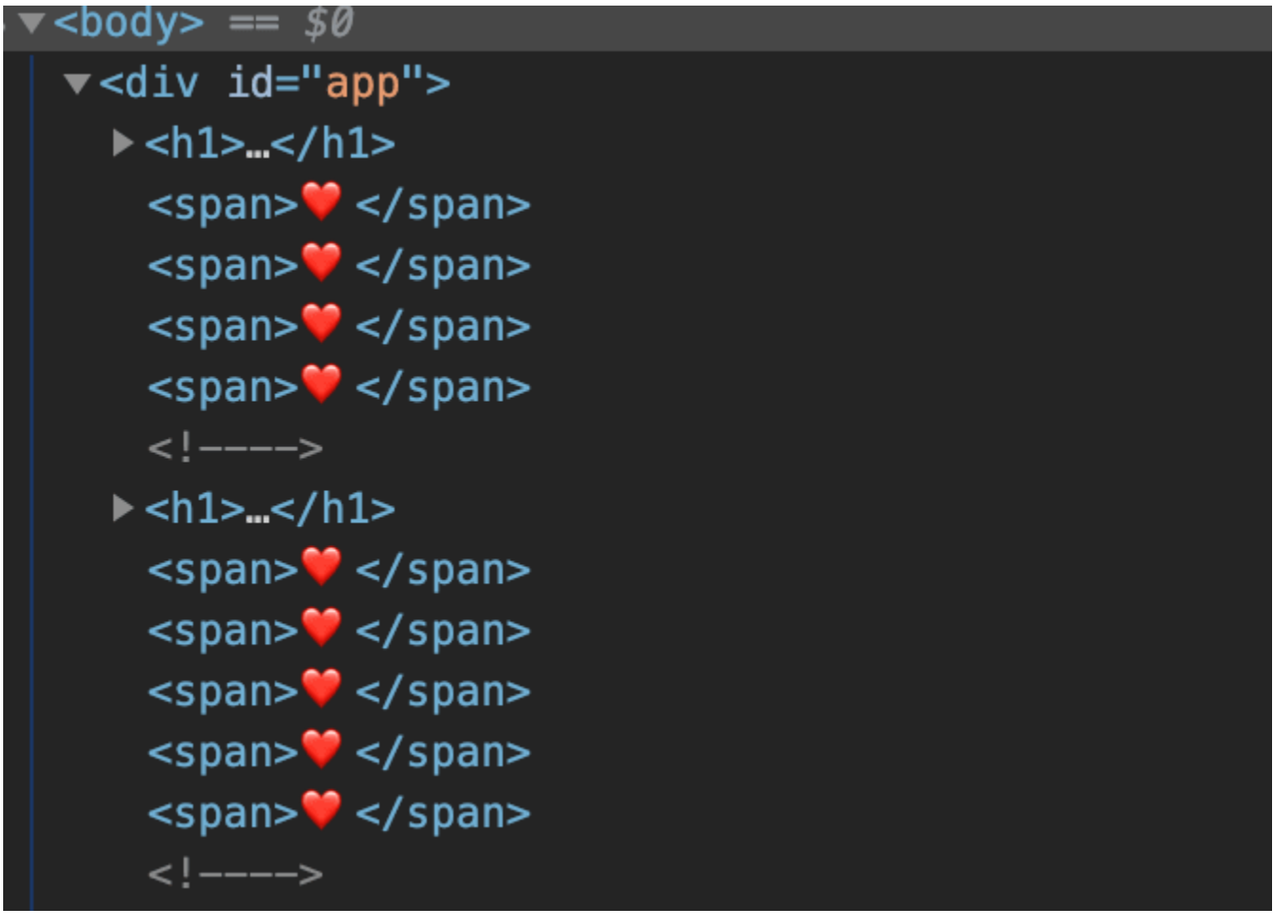
3. The nested `v-for`. So inside the span element we actually have a nested `v-for` loop (which is TOTALLY ok to do), except it's a little different, we're not looping an array or an object. I didn't LIE to you, maybe just withheld some information - like for example, you can actually loop a numeric value (in this case `game.rating` and the loop will count up from 1 till it reaches the value of the rating. Simple?
4. Finally, `v-if`. We are going to output a `<div>` tag inside our loop *IF* the condition is met, so if and only if the current game's rating is greater than 5. Take a guess which?

Go ahead and run this again in your browser and behold the awesomeness of not bothering with CSS.

What if I don't need a wrapping DIV?

If at any point you find yourself making a bunch of `<div>` elements simply to wrap up your `v-for` loops, there's a special HTML tag you can use to help your case.
`<template></template>`

If you, for example, remove the wrapping `<div>` and change it for `<template>` take a look at your developer console and you'll see that the `<h1>` and `<span>` elements are not wrapped by anything.



`<template>` is special, because Vue will treat it as a wrapper element but it won't be rendered into the HTML when we execute it, so you can safely use it to wrap a bunch of other elements logically for the loop without affecting your markup.

The :key attribute

One final thing that I purposely left for the end. The `:key` attribute.

When you are looping through elements with `v-for` *Vue.js* has NO clue how to track your elements for reactivity, because it can't "tell apart" one object from the other. What this means for you is that since *Vue* can't do this, it will re-render the WHOLE section of the page that is being created by this loop. In our case it's a very small section and the performance hit would probably be minimal, but it's something that you should keep in mind - and just do it for best practice.

Now, how do we use it?

`:key` expects some string it'll use to "name" or "track" the element, so we need to give it a unique identifier. In the case of our `games` it's simple, we can do:

```
<div v-for="game in games" :key="game.name">
```

I'm pretty certain that we're not going to have the same game twice in this list, so this is pretty safe. An id if you have data coming from a database is also ideal to use here.

If you are curious about the intricacies of :key you can always take a look at the documentation. Key's docs

In fact, now that you have gotten this far I can't stress enough the importance of getting acquainted with the documentation. *Vue.js* 's docs are impressively good, and very very clear with code examples, the documentation team does a fantastic job to keep them updated and clear - big shout out to all of them.

Final code

Here's the final code, just in case.

```
<html>

<head>
  <title>Vue 101</title>
</head>

<body>
<div id="app">
  <div v-for="game in games" :key="game.name">
    <h1>{{ game.name }} - <small>{{ game.console }}</small></h1>

    <span v-for="star in game.rating">❤️</span>

    <div v-if="game.rating > 5">Wow, this game must be <b>REALLY</b> good</div>
  </div>
</div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

  <script>
    const app = new Vue({
      el: '#app',
        data: {
          games: [
            { name: 'Super Mario 64', console: 'Nintendo 64', rating: 4 },
            { name: 'The Legend of Zelda Ocarina of Time', console: 'Nintendo 64', rating: 5 },
            { name: 'Secret of Mana', console: 'Super Nintendo', rating: 4 },
            { name: 'Fallout 76', console: 'Multiple', rating: 1 },
            { name: 'Super Metroid', console: 'Super Nintendo', rating: 6 }
          ]
        }
    });
  </script>
</body>

</html>
```

Challenge

This time you get a challenge if you wish to accept it. Add a `@click` listener to the `<span>` which outputs the game's rating, and increase the ranking by 1 with each click for that UNIQUE game. You already know everything you need to achieve this ☺.

Tip: You'll need to pass the game you're looping to the click function, and inspect it. Good luck!