Vue.js for Beginners (Part 3)

Welcome back! Last time we went over listening to our very first user events and methods to react to these events. Today you will learn about **directives** and conditional rendering.

A simple if-else

One of the most important tools under the belt of any programmer regardless of framework is conditional rendering. The ability to show or hide parts of your app depending on a condition or value is a great place to start learning about this, and also about Vue **directives**.

We will continue building upon our previous example. In case you lost it or are just catching up, here's what we have so far:

```
<html>
    <head>
        <title>Vue 101</title>
    </head>

    <body>
        <h1>Hello!</h1>
        <div id="app">
          <p>My local property: {{ myLocalProperty }}</p>
          <hr>
          <button @click="buttonClicked">Click me</button>
        </div>

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

        <script>
          const app = new Vue({
            el: '#app',
            data: {
              myLocalProperty: 'Im a local property value'
            },
            methods: {
              buttonClicked() {
                const newText = 'The new value is: ' + Math.floor( Math.random() * 100 );

                this.myLocalProperty = newText;
              }
            }
          });
        </script>
    </body>
</html>
```

So far we've managed to output our local properties into our app, and also listen to the clicks of a user on a simple button.

Let's take it a step further and learn about our conditional rendering.

Let's change our button clicks so that they generate a random number just as we have been doing, but instead of outputting a concatenated string, we will toggle the display of a couple of `<p>` elements with the results.

This will require some refactoring, so first let's change our `buttonClicked` method to only calculate this new number, and we will store it on a new property called `randomNumber`.

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      myLocalProperty: 'Im a local property value',
      randomNumber: 0 // 1
    },
    methods: {
      buttonClicked() {
        this.randomNumber = Math.floor(Math.random() * 100); // 2
      }
    }
  });
</script>
```

Let's take a quick look.

We've added a new local property `randomNumber`, and the default value will be 0.

We deleted the old code, and instead of using the random value on the previous string we will just store it provisionally in our `randomNumber` prop.

We want to show/hide content depending on the result of our `randomNumber` calculation, so let's have two new `<p>` elements. One will show only when `randomNumber` is greater or equal to 50. The other will show if it is less than 50.

```
<div id="app">
  <p>My local property: {{ myLocalProperty }}</p>
  <hr>
  <button @click="buttonClicked">Click me</button>
  <hr>
  <!-- 1 -->
  <p v-if="randomNumber >= 50">randomNumber is >= 50!</p>

  <!-- 2 -->
  <p v-else>Sorry, randomNumber is only <b>{{ randomNumber }}</b></p>
</div>
```

We've added a `<hr>` for clary and separation, and then our two `<p>` elements.

Let's look at each in detail.

First, `v-if="randomNumber >= 50"`. So, `v-if` is a Vue **directive**. Don't get too caught on the definition of the term, it only means that it is a "special" value that we can place inside HTML elements that Vue will know how read and interpret. In fact, you've already used **directives** before. Remember `v-on:click` and `@click`? Those are directives too!

Theory aside, `v-if` tells Vue to only show this element if the condition we declare inside of it is true. In this case, "Vue: only show this <p> element IF and only IF `randomNumber` is greater than or equal that 50".

Second, whenever you have a v-if directive, you can have an else case. But heads up, `v-else` **ONLY** works on an element that directly follows the one that holds the `v-if` (or a third option `v-else-if`). As you'd expect from any if - else statement, the element with `v-else` will get rendered on any other case that is not true for the first. Either/or.

Go ahead and reload your index.html and click the button a few times. You'll see that the `<p>` tags get rendered reactively depending on the value of `randomNumber`.

v-if and v-show

If you're curious to open your dev tools while you click around, you will notice a VERY important thing. `v-if is not a display: block/hidden` css switch toggle, it actually renders or destroys elements whenever the value of our conditional changes. If you want to have a visibility toggle **directive**, go ahead and try switching that first `v-if` for `v-show` and see what happens!

You may notice is that the block that has the `v-else` **declarative** is not showing anymore. This is because `v-show` is a lone-ranger and will only work by itself. So what is the benefit of using `v-show`?

There is a performance cost that you may want to consider when using `v-if` because Vue has to go and re-render the DOM (don't worry it's very smart about which parts it needs to

add/remove) but this is a more extensive task than applying/ removing css `display` properties.

Bottom line: If you're going to toggle a small/medium part of the app a few times only, like a menu bar for example, `v-if` will usually do the trick. But if you're going to be switching around tabbed screens for example, or huge chunks of your page then `v-show` may be cheaper in terms of performance because your markup is not getting re-written every time.

(P.S. before we continue, set back the directive to `v-if` or else you'll get console errors because of the `v-else` below it is unpaired.)

Conclusion

With the fundamentals that we've already covered: setup, events, properties and conditional rendering you now have the building blocks to start creating some really fun and reactive applications. However, this is just barely scratching the surface of the power of Vue and it only gets more fun and interesting from here.

Stay tuned for part 4!