Vue.js for Beginners (Part 2)

Last time (in part 1 of this series) we figured out how to add Vue to our index.html with a regular `<script>` tag, and we managed to add our very first reactive property to the page. Today, let's learn how we can change this property with user input.

Our code so far looks like this:

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

    <body>
        <h1>Hello!</h1>
        <div id="app">
          <p>My local property: {{ myLocalProperty }}</p>
        </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'
            }
          });
        </script>
    </body>
</html>
```

Listening to user events

In order to better showcase the reactivity of **Vue**, and to learn how to react to user events we are going to add a button to our app that will change the value of our `myLocalProperty` prop.

Go ahead and first add a new button to our `<div id="app">`.

```
<div id="app">
  <p>My local property: {{ myLocalProperty }}</p>
  <hr>
  <button>Click me</button>
</div>
```

Now, how do we react to this button getting clicked?

If you come from a **jQuery** background your instinct may be to try to do something like this: `$('button').click();`, however, there's a golden rule in Vue. NEVER manipulate the DOM (elements in the page's HTML) directly.

Without going into super intricate details, **Vue** keeps a virtual "copy" of your HTML (in this case our div with the "app" ID) and automagically figures out where and how to update it when properties change.

If you make changes to the DOM directly with JavaScript, you risk losing these changes and unexpected behavior whenever Vue re-renders the content, because it will not be aware of these changes.

Enough theory though, let's move on with the clicking. Add this **event handler** to our button:

```
<button v-on:click="myLocalProperty = 'The button has been clicked'">
  Click me
</button>
```

A couple of things are happening here.

v-on:click="" In **Vue** we have these "directives" that we can add to our HTML content.

A directive simply put is an HTML parameter that **Vue** can understand.

In this particular case, we are telling Vue: *Vue* (v-), *on* the user's click do this: "`myLocalProperty = 'The button has been clicked'`", which is simply an inline declaration to change the value of our property.

If you go ahead and open your `index.html` file now in the browser and click the button, you will see the string that we interpolated earlier inside the {{ }} in our code will react to our button modifying the property.

Alternate syntax

In most places you will most likely not find events being set on the HTML with `v-on:[eventname]` as we have in this example because in Vue we have a very handy shorthand for this type of thing. `@[eventname].`

Let's change our `<button>` click even to to use this shorthand:

```
<button @click="myLocalProperty = 'The button has been clicked'">Click me</button>
```

Methods

In most cases, when a user event like the click of a button is fired, you will need to do a lot more than changing the value of a variable. So let's learn about **methods** (aka, functions).

To continue with our example, let's make the button call a function that will do something really simple, it'll change the value of `myLocalProperty` by appending a random number to a string.

Delete our previous implementation of `@click` and replace it with this:

```
<button @click="buttonClicked">Click me</button>
```

Notice that we're not adding a () after `"buttonClicked".` We can omit these when we are not passing any arguments to our function. For example, `@click="changeName('Marina')".` (More on this later when we look at conditional rendering ☺)

Now that we have our button ready to execute buttonClicked on clicks, we need to actually write this function.

Vue has a special place to write functions that our **Vue instance** can use. This place is inside the { } we passed to our new `Vue({})` line before.

We will create a methods: {} property that will hold an object filled with our functions.

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      myLocalProperty: 'Im a local property value'
    },
    methods: { // 1
      buttonClicked() { // 2
        const newText = 'The new value is: ' + Math.floor( Math.random() * 100); // 3

        this.myLocalProperty = newText; // 4
      }
    }
  });
</script>
```

Let's dissect this:

1. We declare the methods property inside our **Vue** instance. As I mentioned, in here you will put all your instance methods/functions.
2. Inside the methods object { } we declare `buttonClicked(),` which is the function we are trying to call on our `@click` listener. We're not going to use any parameters at this point so empty ().
3. We join the value of the rounded down value `Math.floor` of the result of multiplying the random value of 0-1 by 100 to a string and store it in a constant.
4. We assign the value of our new string to `myLocalProperty.` Now be very careful about this tiny detail 🙃 (lame pun intended). When we assign new values to the properties inside the instance's data property (the one inside data: {}) you **MUST** access it through `this.[prop-name].`

In the context of a method the keyword this refers to the **Vue** instance. Vue will perform some magic behind the scenes so that you can read/write to your properties inside data by doing `this.property = value.`

Now that we have everything set up, reload your index.html file and click on your button. The value of our interpolated {{ }} string on the containing <p> will be updated every time you click the button, for every single time the buttonClicked function is executed. Once again, the magic of Vue's reactivity is coming into play.

Wrapping up

If at this point you're thinking, well, this is really easy then you're on the right track. One of the things I love the most about this framework is its clear syntax and simplicity. It just works. But this should not be confused with thinking that Vue is not powerful.

We're merely scratching the surface of what we can do with Vue so far, but you'll see as we progress through these articles that these tiny building blocks put together will soon make the core of your amazing next app.

Stay tuned for part 3!