WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

FORM INPUTS

FORM INPUT BINDINGS

- You can use the v-model directive to create two-way data bindings on form input, textarea, and select elements.

V-MODEL

```
<input v-model="message" placeholder="edit me">

<p>Message is: {{ message }}</p>
```

v-model will ignore the initial value, checked or selected attributes found on any form elements. It will always treat the Vue instance data as the source of truth.

CHECKBOX

```
new Vue({
    el: '#example-3’,
    data: {
        checkedNames: []
    }
})
```

- Single checkbox, boolean value.
- Multiple checkboxes, bound to the same Array:

```
<div id='example-3’>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames">
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
  <label for="mike">Mike</label>
  <br> <span>Checked names: {{ checkedNames }}</span>
</div>
```

RADIO

```
<input type="radio" id="one" value="One" v-model="picked">
<label for="one">One</label> <br>
<input type="radio" id="two" value="Two" v-model="picked">
<label for="two">Two</label> <br>
<span>Picked: {{ picked }}</span>
```

SELECT

```
<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>

<span>Selected: {{ selected }}</span>
```

MODIFIERS

- . lazy - by default, v-model syncs the input with the data after each input event. You can add the lazy modifier to instead sync after change events

- .number - If you want user input to be automatically typecast as a number, you can add the number modifier to your v-model.

```
<input v-model.number="age" type="number">
```

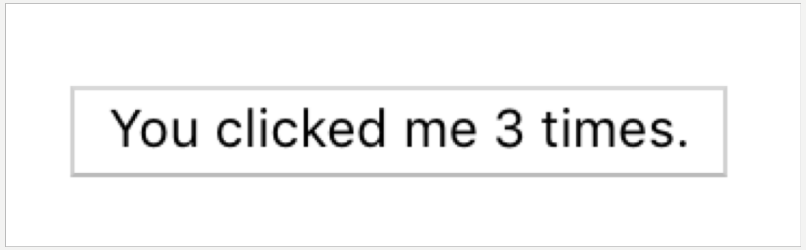
- .trim - If you want whitespace from user input to be trimmed automatically, you can add the trim modifier to your v-model

COMPONENTS

COMPONENTS BASICS

- Components are reusable Vue instances with a name.
- Here's an example of a Vue component:

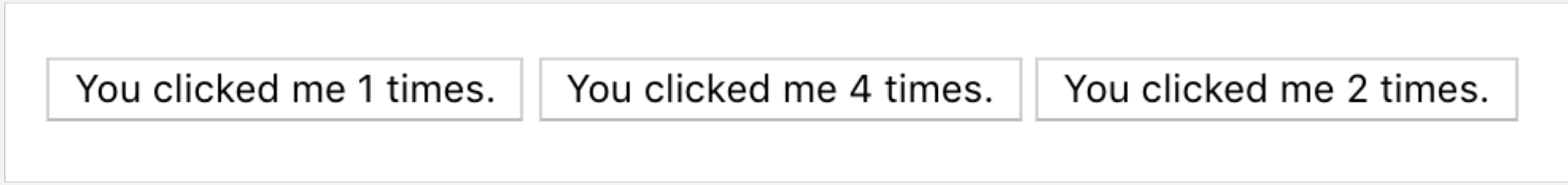
```
// Define a new component called button-counter
Vue.component('button-counter', {
    data: function () {
        return { count: 0 }
    },
    template: '<button v-on:click="count++">You clicked me {{ count }} times.</button>'
})
```



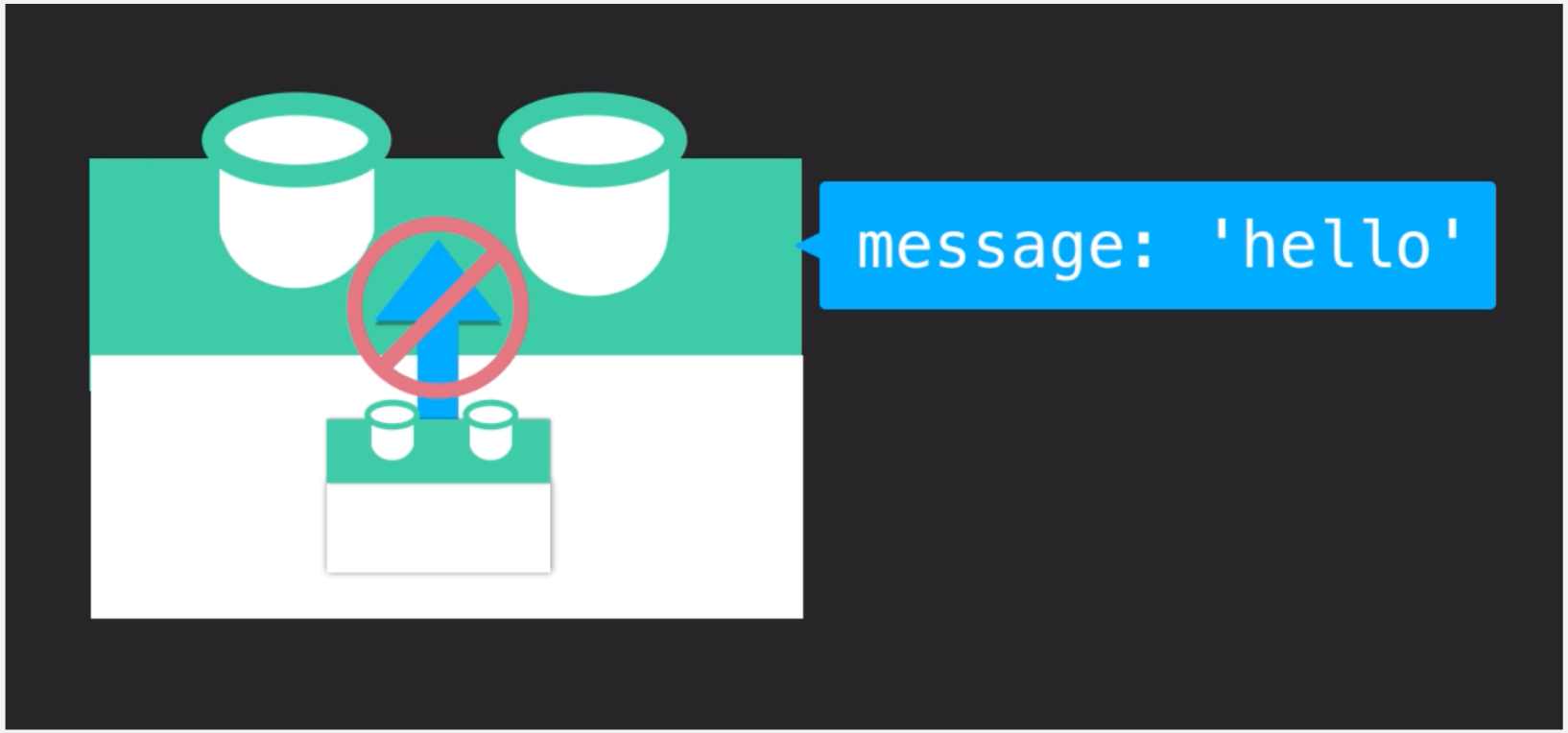
COMPONENTS BASICS

- Since components are reusable Vue instances, they accept the same options as new Vue, such as data, computed, watch, methods, and lifecycle hooks. The only exceptions are a few root-specific options like el.

```
<div id="components-demo">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
```

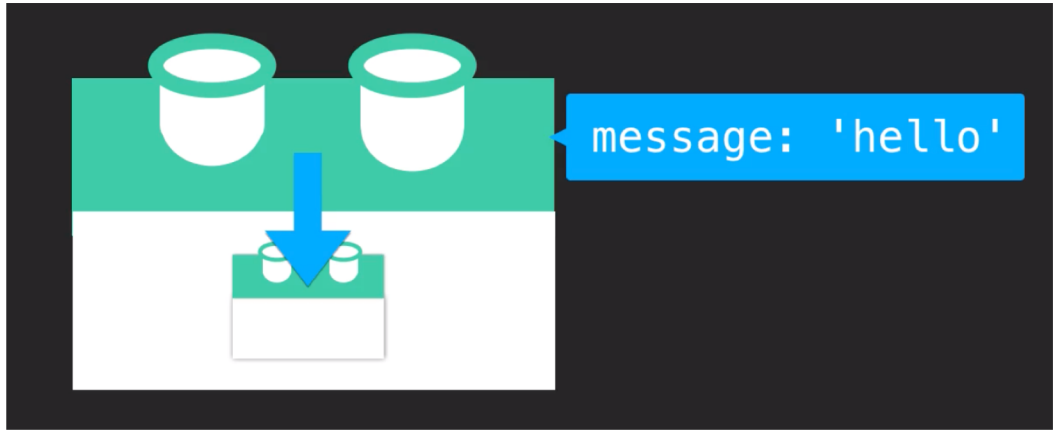


PASSING DATA TO CHILD COMPONENTS WITH PROPS



PASSING DATA TO CHILD COMPONENTS WITH PROPS

- Props are custom attributes you can register on a component.
- When a value is passed to a prop attribute, it becomes a property on that component instance.
- A component can have as many props as you'd like and by default, any value can be passed to any prop.
- You can access this value on the component instance, just like with **data**.



USING PROPS

- JS

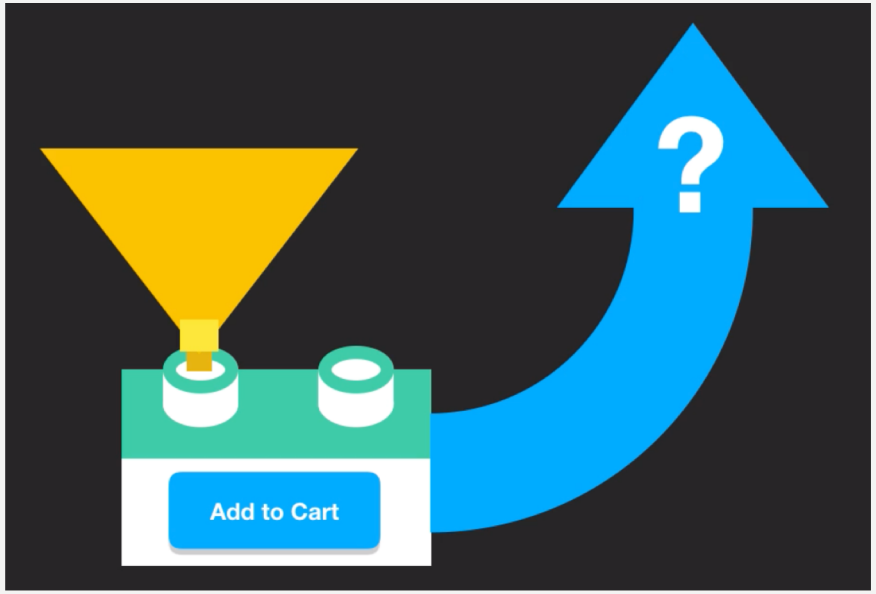
```
Vue.component('blog-post', {
    props: ['title'],
    template: '<h3>{{ title }}</h3>'
})
```

- HTML

```
<blog-post title="My journey with Vue"></blog-post>
<blog-post title="Blogging with Vue"></blog-post>
<blog-post title="Why Vue is so fun"></blog-post>
```

LISTENING TO CHILD COMPONENTS EVENTS

```
<product :premium="true" @add-cart="updateCart()" >
</product>
```




```
$emit('add-cart')
```

EMITTING A VALUE WITH AN EVENT

- It's sometimes useful to emit a specific value with an event.

```
<button v-on:click="$emit('enlarge-text', 0.1)">
  Enlarge text
</button>
```

- When we listen to the event in the parent, we can access the emitted event's value with **$event**

```
<blog-post ... v-on:enlarge-text="postFontSize += $event" >
</blog-post>
```

EMITTING A VALUE WITH AN EVENT

- Or, if the event handler is a method:

```
<blog-post ... v-on:enlarge-text="onEnlargeText" >
</blog-post>

methods: {
    onEnlargeText: function (enlargeAmount) {
        this.postFontSize += enlargeAmount
    }
}
```

USING V-MODEL ON COMPONENTS

- Custom events can also be used to create custom inputs that work with v-model.

```
<input v-model="searchText">
```

- does the same thing as:

```
<custom-input
    v-bind:value="searchText"
    v-on:input="searchText = $event" >
</custom-input>
```

So for your custom component, you need to have:

- A prop named "value" to pass data to the component
- Emit a event named "input" to emit an event with a value from the component