Vue.js

Framework overview for People in a hurry

You can download this presentation as PDF

Why Vue?

Key benefits of using Vue.js

- Progressive (incrementally adoptable) - you don’t have to go full Vue.js
- Small core library focused on UI - everything else handled by 3rd party (routing, global state)
- Intuitive template syntax - feels like just writing HTML
- Single File Components - template, JavaScript & CSS all in one file

Vue instance

A taste of Vue.js

Interpolation, `{{ message }}` would output the content of message data property

Event handling v-on:click would call changeMessage() method when element is clicked

```
<div id="app">
  <p>{{ message }}</p>
  <button v-on:click="changeMessage">Say Bye!</button>
  <p v-if="changed">Message changed</p>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    message: 'Hi!',
    changed: false
  },
  methods: {
    changeMessage() {
      this.message = 'Bye!';
    }
  },
  watch: {
    message: function () {
      this.changed = true;
    }
  },
  created() {
    fetch('http://google.com');
  }
})
</script>
```

Conditional rendering - `<p>` would only be rendered if `changed` is true

Vue is bound to this element

Reactive data

Vue would automatically re-render if any of this „reactive“ data changes

Methods are used to group logic

Methods

Watchers

When `message` data property changes, this function would be called

Lifecycle methods are called during Vue life - eg. `created()` when Vue is created

„Lifycycle“ methods

Templating

```
<div id="app">
  <span>Message: {{ msg }}</span>

  <span v-once>This will never change: {{ msg }}</span>

  <p>Outputting raw HTML: <span v-html="rawHtml"></span></p>

  <div v-bind:disabled="disabled"></div>

  {{ number + 1 }}
  {{ isOk ? 'YES' : 'NO' }}
  {{ msg.split('').reverse().join('') }}

  <p v-if="show">You will see this text</p>

  <a v-bind:href="url">A link with dynamic URL</a>
  <a v-on:click="doIt">Click this to run doIt() method</a>
  <a v-on:[eventName]="doIt">The event will be evaluted dynamically</a>
</div>
```

Mustache interpolation syntax

The `{{ msg }}` would be rendered only once, with initial value of `msg`

Expressions are fine in mustache syntax, control structures (`if`, `for`) are not

Mustache syntax cannot be used to set attribute value

To output raw HTML, use `v-html` directive

A `Directive`, special attribute starting with `v-` prefix. Reactively applies changes to the DOM

Use `v-bind` directive instead. `v-bind:disabled` would set `disabled` to the value of `disabled` data property

`v-bind`, dynamically set HTML attributes

`v-on`, listen to the DOM event (`onClick` in this case)

`v-on:[eventName]` will listen to event specified inside eventName data (new in Vue 2.6)

Templating cd…

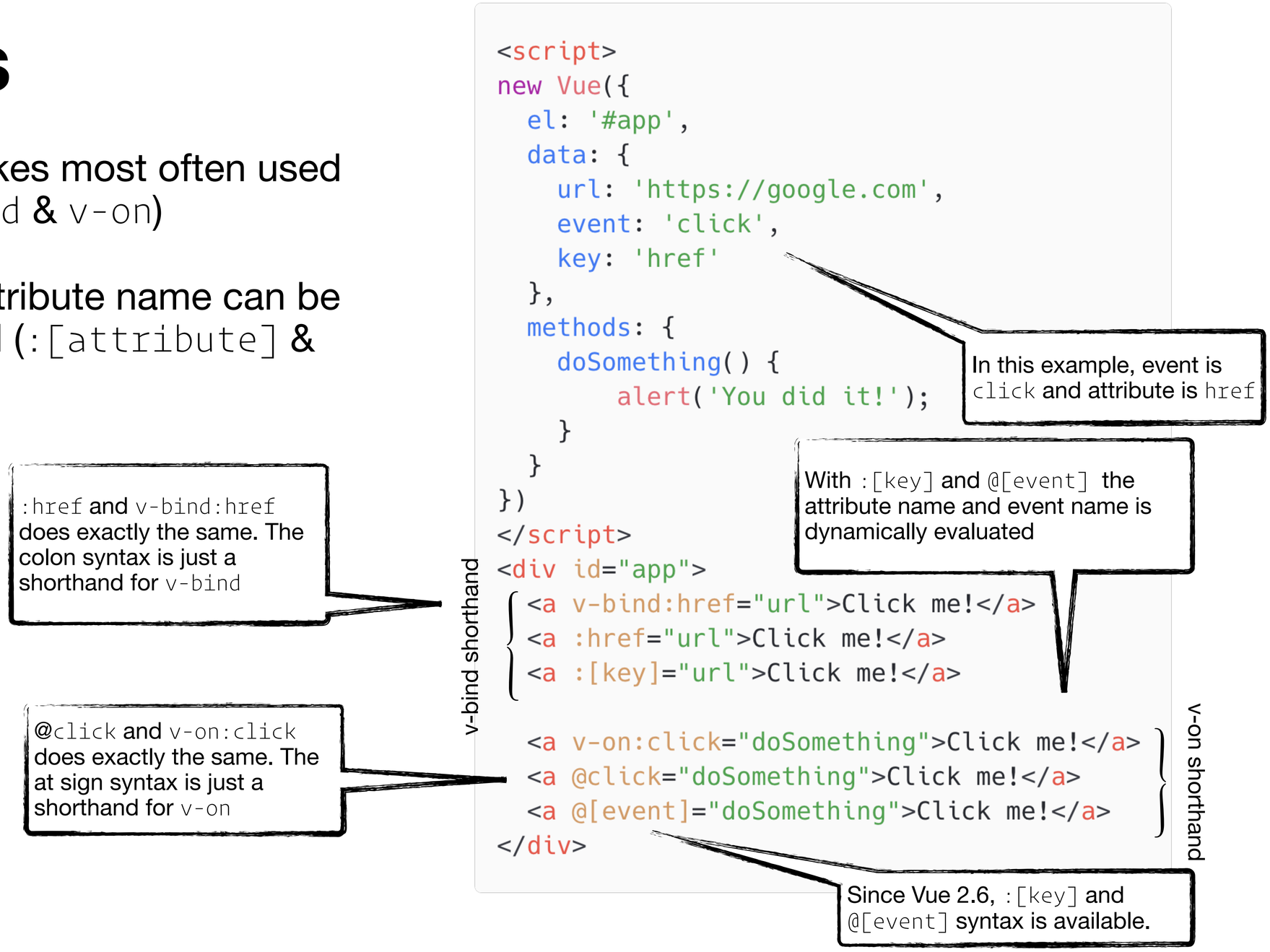
The Vue instance with reactive data and methods for the last page example

Reactive data

```
<script>
 new Vue({
  el: '#app',
  data: {
    number: 2,
    msg: "I'll be back!",
    rawHtml: '<span style="color:red;">This is red!</span>',
    disabled: false,
    isOk: true,
    show: true,
    url: 'https://google.com',
    eventName: 'click'
  },
  methods: {
    doIt() {
      alert('You did it!');
    }
  }
})
</script>
```

Shorthands

- Shorthand syntax makes most often used directives brief (`v-bind` & `v-on`)
- Both event name & attribute name can be dynamically evaluated (`:[attribute]` & `@[event]`)



Computed Properties

- They replace in-template expressions
- Can be read just like a normal data property
- Computed property result is cached, method result is not

```
<script>
let vm = new Vue({
  el: '#app',
  data: {
    firstName: 'Piotr',
    surname: 'Jura',
  },
  computed: {
    fullName() {
      return this.firstName + ' ' + this.surname;
    }
  }
})
</script>
<div id="app">
  <p>Hello {{ fullName }}</p>
</div>
```

When these change, the `fullName` computed property value will be re-evaluated

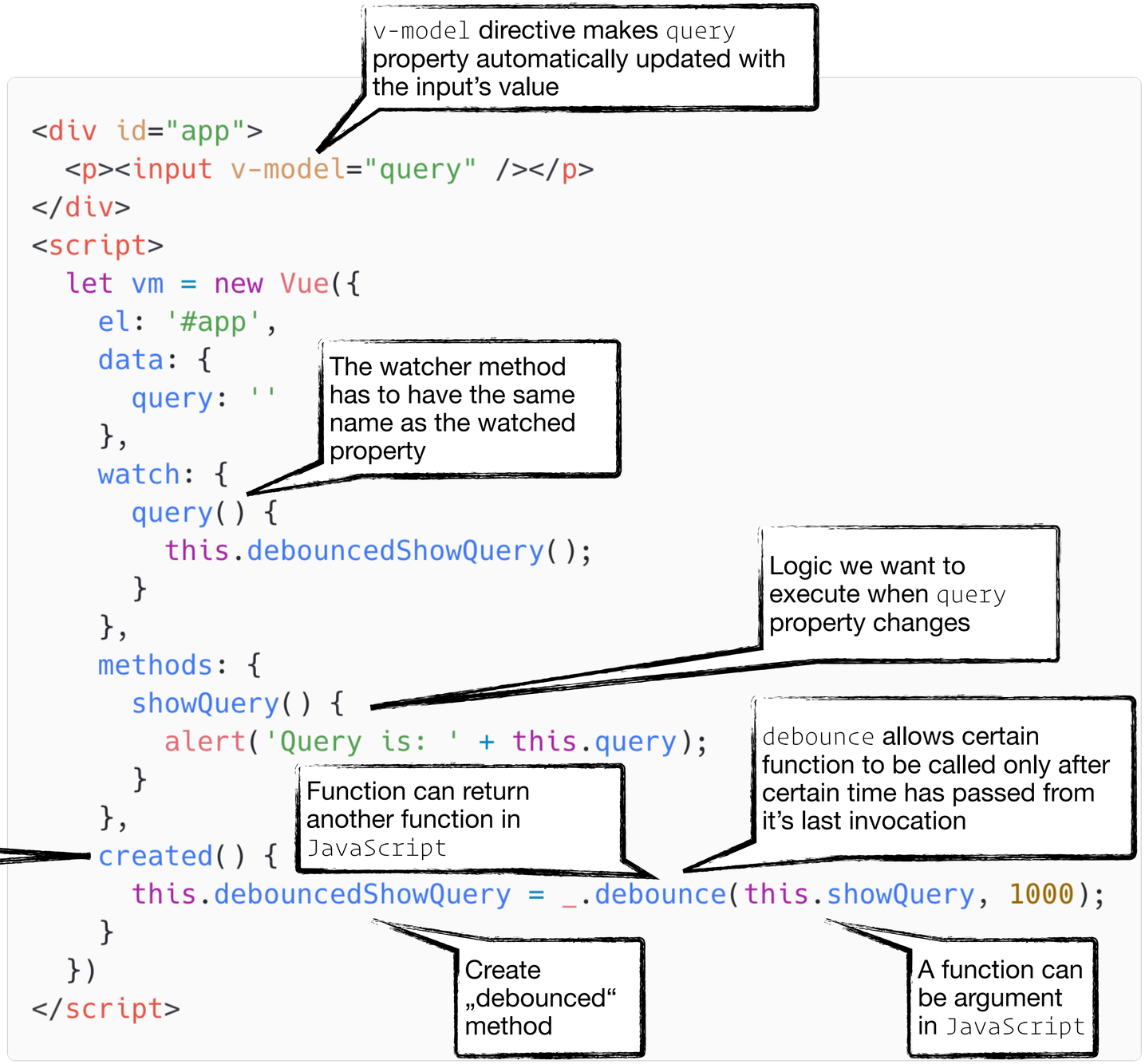
You can now use `{{ fullName }}` as data property

Computed properties are added under „computed“ property

Computed property is a function, which result depends on data properties

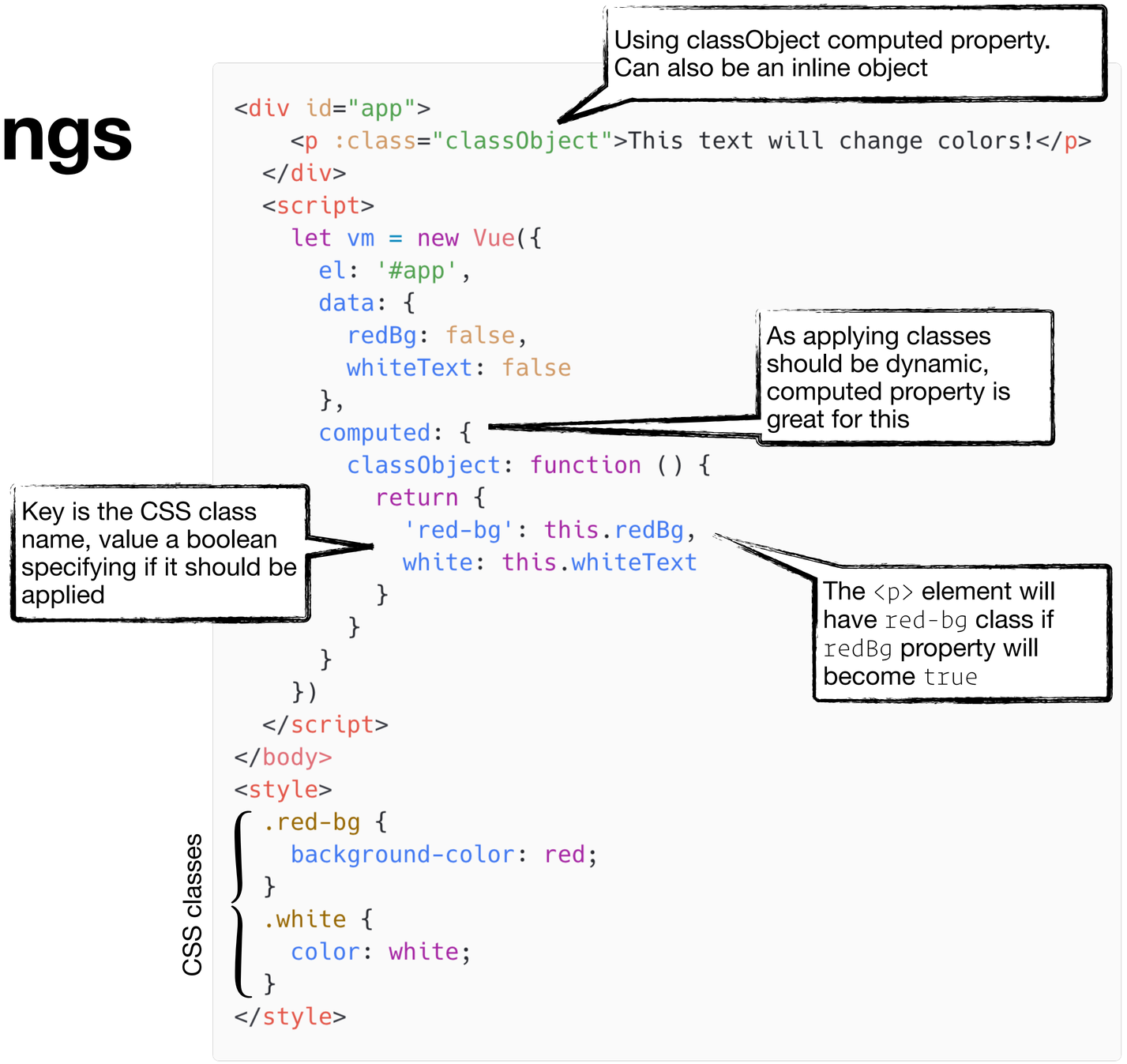
Watchers

- Function called when property value changes
- Usually computed property is a better option
- Best used when performing asynchronous/expensive operation



Class & Style Bindings

- Special handling of `v-bind` combined with `class` attribute
- Much easier than string concatenation




```
<div id="app">
    <p :class="classObject">This text will change colors!</p>
  </div>
  <script>
    let vm = new Vue({
      el: '#app',
      data: {
        redBg: false,
        whiteText: false
      },
      computed: {
        classObject: function () {
          return {
            'red-bg': this.redBg,
            white: this.whiteText
          }
        }
      }
    })
  </script>
</body>
<style>
  .red-bg {
    background-color: red;
  }
  .white {
    color: white;
  }
</style>
```

Class & Style Bindings - Array Syntax

```
<div id="app">
  <p :class="[redBg, whiteText]">Colors!</p>
</div>
<script>
  let vm = new Vue({
    el: '#app',
    data: {
      redBg: 'red-bg',
      whiteText: 'white-text'
    }
  })
</script>
<style>
  .red-bg {
    background-color: red;
  }
  .white-text {
    color: white;
  }
</style>
```

Array syntax lists CSS classes to apply

Data properties contain CSS class names

That's the HTML output

```
<div id="app">
  <p class="red-bg white-text">Colors!</p>
</div>
```

Conditional Rendering

- `v-if` directive allows conditional block rendering
- Block will be render only if expression in `v-if` evaluates to `true`

```
<div id="app">
  <p>Please guess the number (1 or 2):
    <input v-model="guess" /></p>
  <p v-if="guessedOk">You gussed right!</p>
  <p v-else-if="guess == null">Type the number!</p>
  <p v-else>Sorry try again!</p>
</div>
<script>
  let vm = new Vue({
    el: '#app',
    data: {
      guess: null,
      answer: Math.floor(Math.random() * 2) + 1
    },
    computed: {
      guessedOk: function () {
        return this.guess == this.answer;
      }
    }
  })
</script>
```

Typing the answer modifies the `guess` data property (using `v-model`)

Only one `<p>` element would be rendered

Initially guess is null, so second `<p>` element is rendered first

After typing the answer the `guess` is not null, so second `<p>` will not be rendered

We are left with first or third `<p>` element. The answer is either correct, or not - meaning `guessedOk` returns `true` or `false`

This is how you generate a random number in JavaScript. The result will always be 1 or 2

Renderling lists

- `v-for` can render list of items from `Array`
- `v-for` can't be used on single element together with `v-if`

```
<div id="app">
  <ul>
    <li v-for="pizza in pizzas" :key="pizza.id">
      {{ pizza.name }} ({{ pizza.price }}€)
    </li>
  </ul>
</div>
<script>
  let vm = new Vue({
    el: '#app',
    data: {
      pizzas: [
        { id: 1, name: 'Margherita', price: 5 },
        { id: 2, name: 'Marinara', price: 6 },
        { id: 3, name: 'Capricciosa', price: 7.5 }
      ]
    }
  })
</script>
```

`v-for` will render `<li>` element for every item in `pizzas` array

`pizza` variable is the current element of the `pizzas` array

With `v-for` you need to give every element a `key` attribute that has to be unique across this array

`v-for` can't be used together with `v-if` on the same element. You have to wrap it with another HTML element

You can use anything for the :`key`, as long as it's unique across the array. In this case `name` & `price` would work too

If an item gets added to `pizzas` array, Vue will automatically re-render

Form input binding

- `v-model` is used to bind a reactive property to form input
- `v-model` will ignore initial `value`, `checked` or `selected` attributes

`v-model` makes 2-way binding to the property. Input change would update property, property change would update input

```
<div id="app">
  <form>
    Name: <input v-model="name">
    Decription: <textarea v-model="desc"></textarea>
    Accept:</label> <input type="checkbox" v-model="accept">
    Yes/No:
      <input type="radio" value="Yes" v-model="yesNo"> Yes
      <input type="radio" value="No" v-model="yesNo"> No
  </form>
</div>
<script>
  let vm = new Vue({
    el: '#app',
    data: {
      name: 'Piotr',
      desc: 'Udemy Instructor',
      accept: false,
      yesNo: 'Yes'
    }
  })
</script>
```

`v-model` with checkbox means property set to `true` if checked, `false` if unchecked

`v-model` with radio will set the property to selected input's value attribute

Event Handling

- `v-on` directive allows listening to the DOM events

`v-on` can also trigger a method by name - the default argument to that method is `event`

Eg. `self` modifier would only trigger `click` event if the element itself was clicked (by default events from child elements are propagated to parent elements!)

There are plenty of other event modifiers, check them out here: https://vuejs.org/v2/guide/events.html#Event-Modifiers

`stop` modifier would on the other hand stop propagating `click` event to elements parents

```
<div id="app">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button has been clicked {{ counter }} times.</p>
  <p v-if="counter % 5 == 0 && counter > 0">Go on, click 5 more!</p>
  <p><button v-on:click="sayHi" id="btn1">HI!</button></p>
  <p>
    <button v-on:click="say('Hello there!', $event)" id="btn2">Hello!</button>
  </p>
  <div v-on:click.self="say('Top')">
    Top
    <div v-on:click="say('Second')">Second
      <div v-on:click.stop="say('Deep')">Deep</div>
    </div>
  </div>
</div>
<script>
  var vm = new Vue({
    el: '#app',
    data: {
      counter: 0
    },
    methods: {
      sayHi(event) {
        alert('Hi there!');
        alert(event.target.tagName);
      },
      say(message, event) {
        alert(message);
        alert(event.target.id);
      }
    }
  })
</script>
```

`v-on` handler can be inline or a method, remember `v-on:click` is equivalent to `@click`

Methods can be called inline, to pass the relevant event object, use magic `$event` variable

You can use an event modifier after the dot, which will change how the event is handled