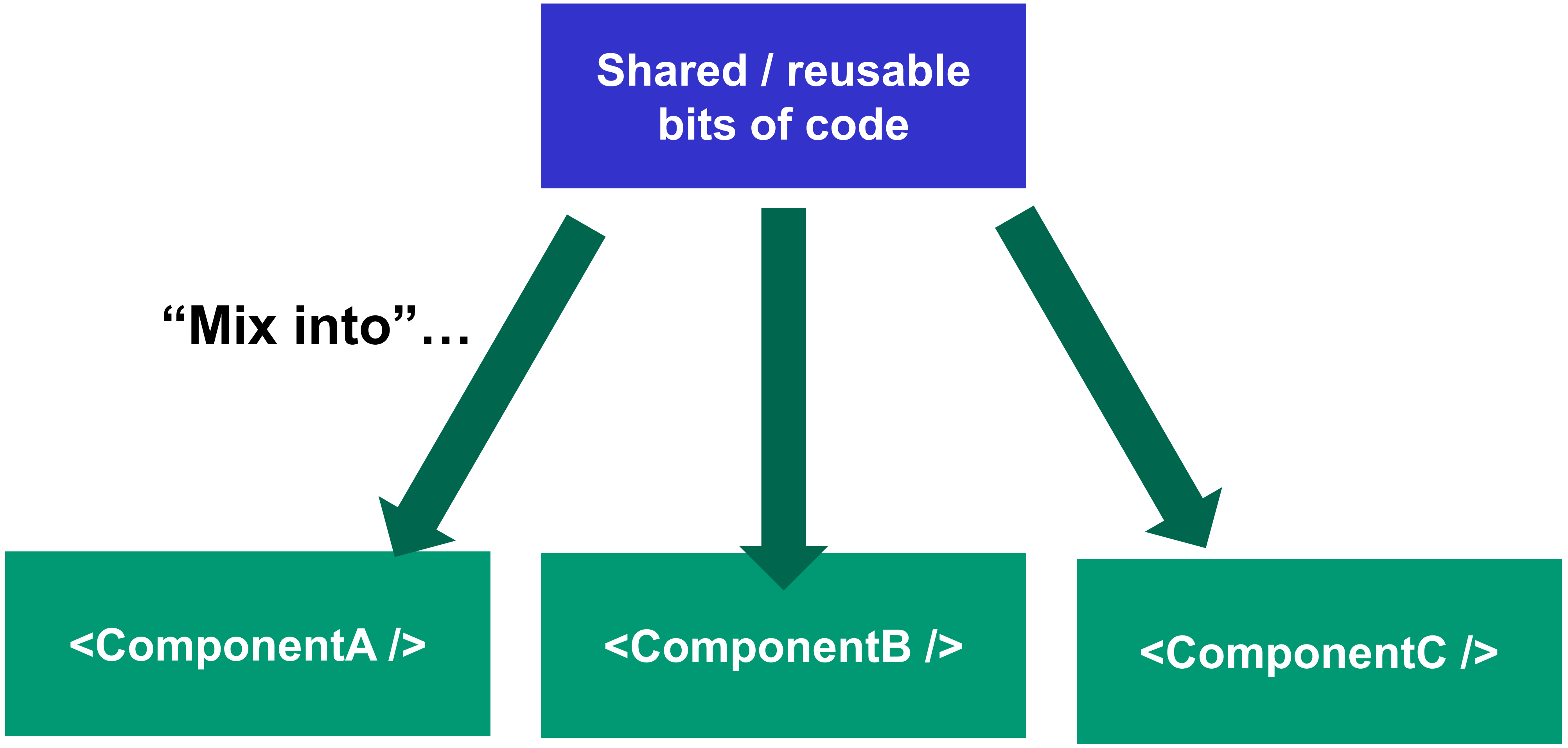
Vue Fundamentals - 03
Style binding & in depth components

Peter Kassenaar –
info@kassenaar.com

Using mixins

Reuse functionality across components

Mixin architecture



“Mixins are a flexible way to distribute reusable functionalities for Vue components. A mixin object can contain any component options. When a component uses a mixin, all options in the mixin will be “mixed” into the component’s own options.”

https://vuejs.org/v2/guide/mixins.html

Mixins

- Mixins are a way to share functionality across components
- Useful if you find yourself duplicating code in multiple components
- Usually stored in a separate file

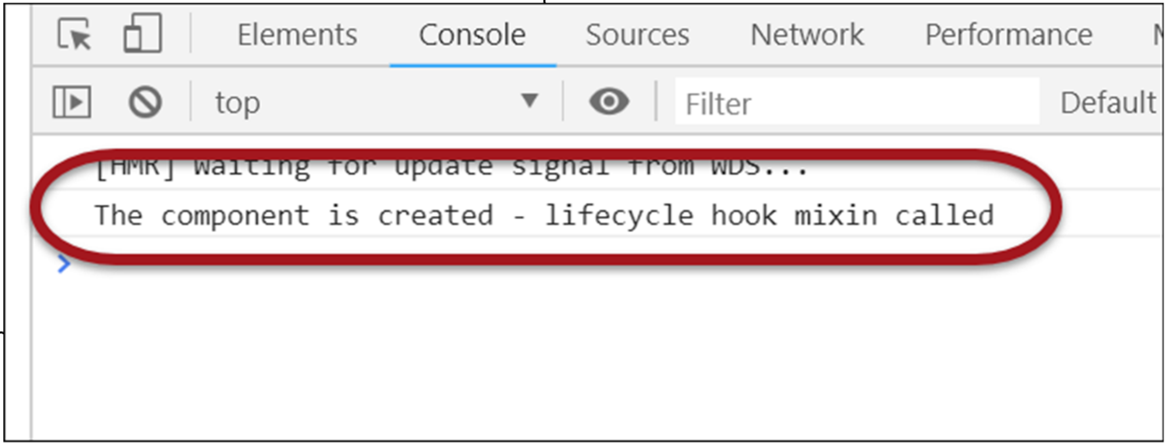
```
// mixins.js - export default mixins in this application.
// They can be loaded into every component that needs them.
export default {
   // Using the 'created' lifecycle hook in a mixin
   created(){
      console.log(‘Component created - lifecycle hook mixin called');
   },
}
```

https://vuejs.org/v2/guide/mixins.html

Using a mixin

- Import the `mixin.js` file in the component
- Add it to the mixins property of the component.
- It is used as before
- NOT just lifecycle hooks! All kinds of functionality & data you want to share

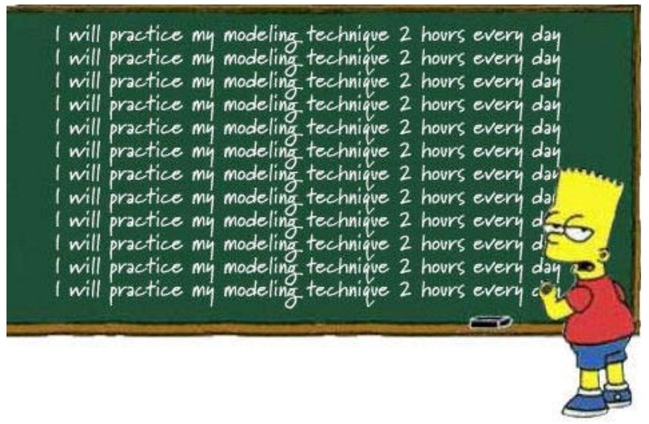
```
<script>
    …
   // import the mixin
   import createdHookMixin from '../mixins/mixins.js'

   export default {
      name: "VacationList",
      …,
      mixins:[createdHookMixin]
   }
</script>
```



```
[HMR] Waiting for update signal from WDS...
The component is created - lifecycle hook mixin called
```

Workshop - mixins

- Create a mixin for the `data.js` – file
- Use it in the component.
  - See if the component can still access the data
  - This way you can share data over multiple components
- General example: `../160-mixins`



Component lifecycle hooks

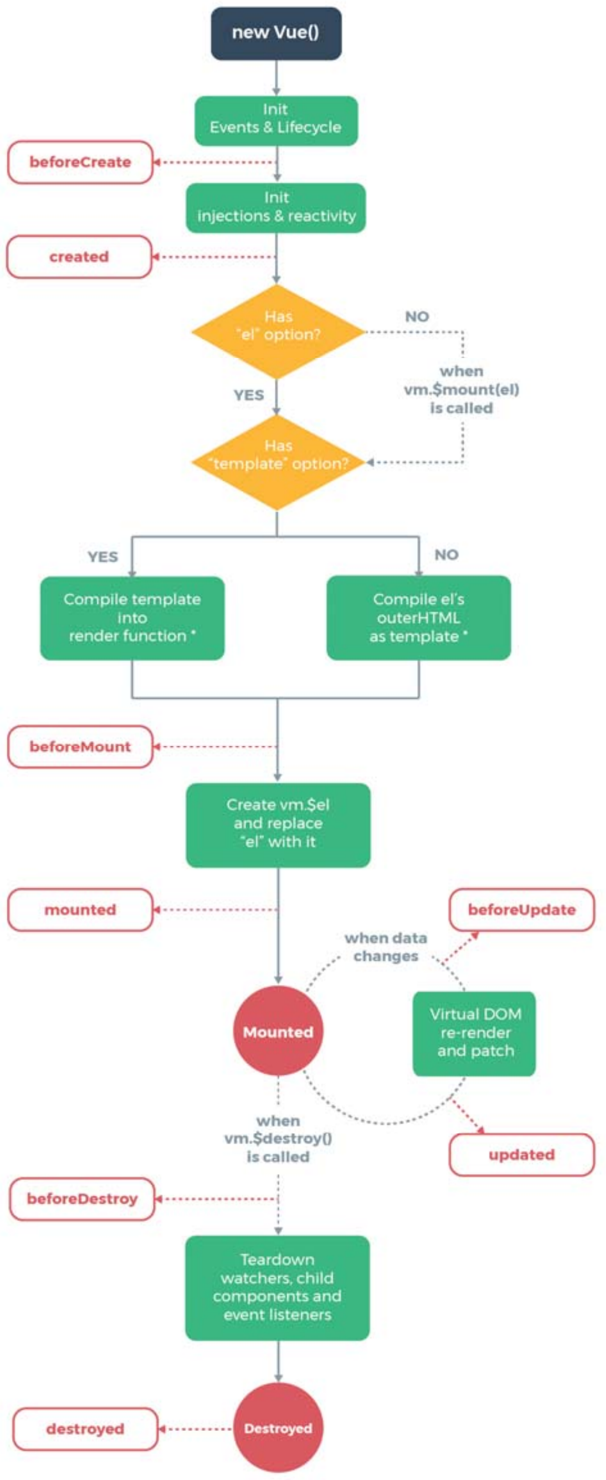
Tapping into the lifecycle of created components

Lifecycle hooks

- Perform an action automatically when a specific lifecycle event occurs

“Each Vue instance goes through a series of initialization steps when it’s created - for example, it needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes.”

https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks


new Vue()
Init Events & Lifecycle
beforeCreate
Init injections & reactivity
created
Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

Official lifecycle diagram

The Red squares are the lifecycle hook methods.

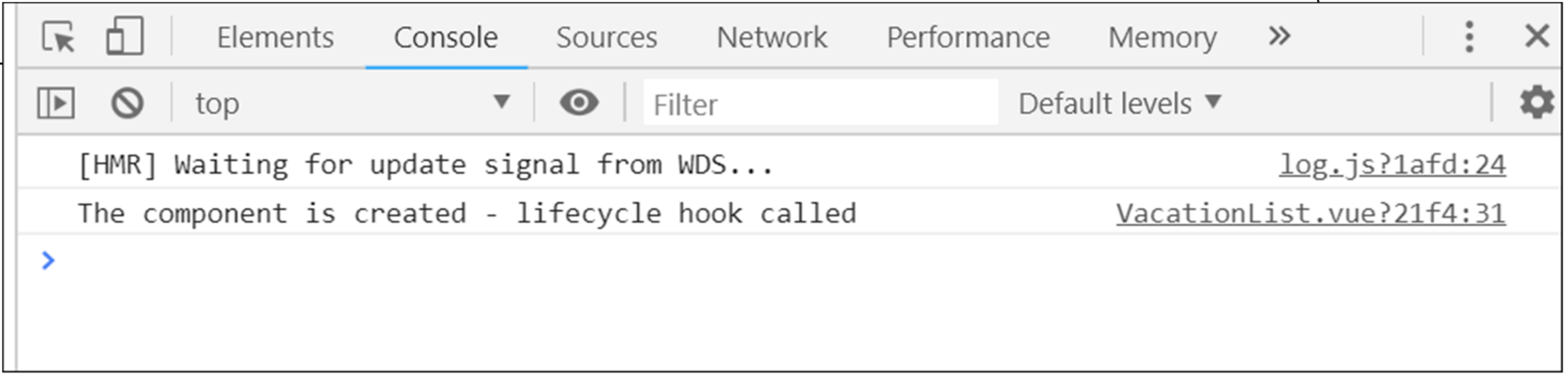
Most used:

- `created`
- `updated`
- `destroyed`

https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks

Using the created hook

```
export default {
   name: "VacationList",
   data() {
      return {
         header: 'List of destinations',
      }
   },
    // Using the 'created' lifecycle hook.
   created(){
      console.log('The component is created - lifecycle hook called');
      // update the header
        this.header = 'The component is created';
   },
   ...
}
```



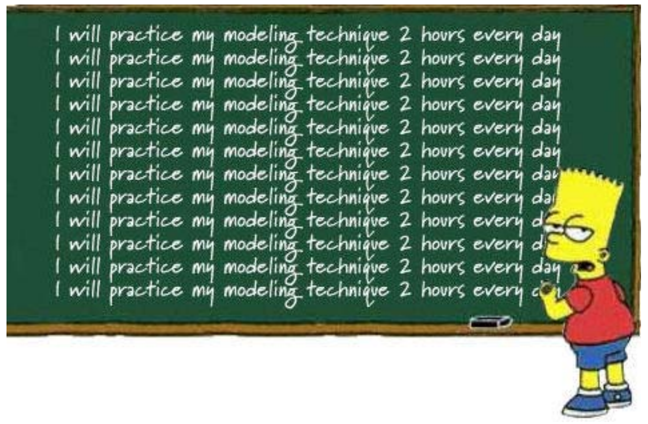
```
[HMR] Waiting for update signal from WDS...              log.js?1afd:24
The component is created - lifecycle hook called         VacationList.vue?21f4:31
```

Usage of lifecycle hooks

- Typical usage
  - `created` – initialisation of variables, call API’s for fetching data etc.
  - `mounted` – if you want to access or modify the DOM.
  - `updated` – when the component receives new data from the outside (props)
  - `destroyed` – to destroy or garbage collect stuff that is not removed automatically

Workshop

- Create a new component.
- Give it some data that you bind in the UI.
- Use a lifecycle hook `created` to log to the console that the component is created.
- Update the data in the `created` lifecycle hook. Verify that it is shown correctly in the UI.
- Read the documentation on some other lifecycle hooks, for instance https://alligator.io/vuejs/component-lifecycle/
- Example: `…/150-lifecycle-hooks`



Global styles and scoped styles

With default styles, CSS is globally available.

For instance, see `App.vue`:

```
<style>
    #app {
        font-family: 'Avenir', Helvetica, Arial, sans-serif;
        color: #2c3e50;
    }
</style>
```

This is also true for components!

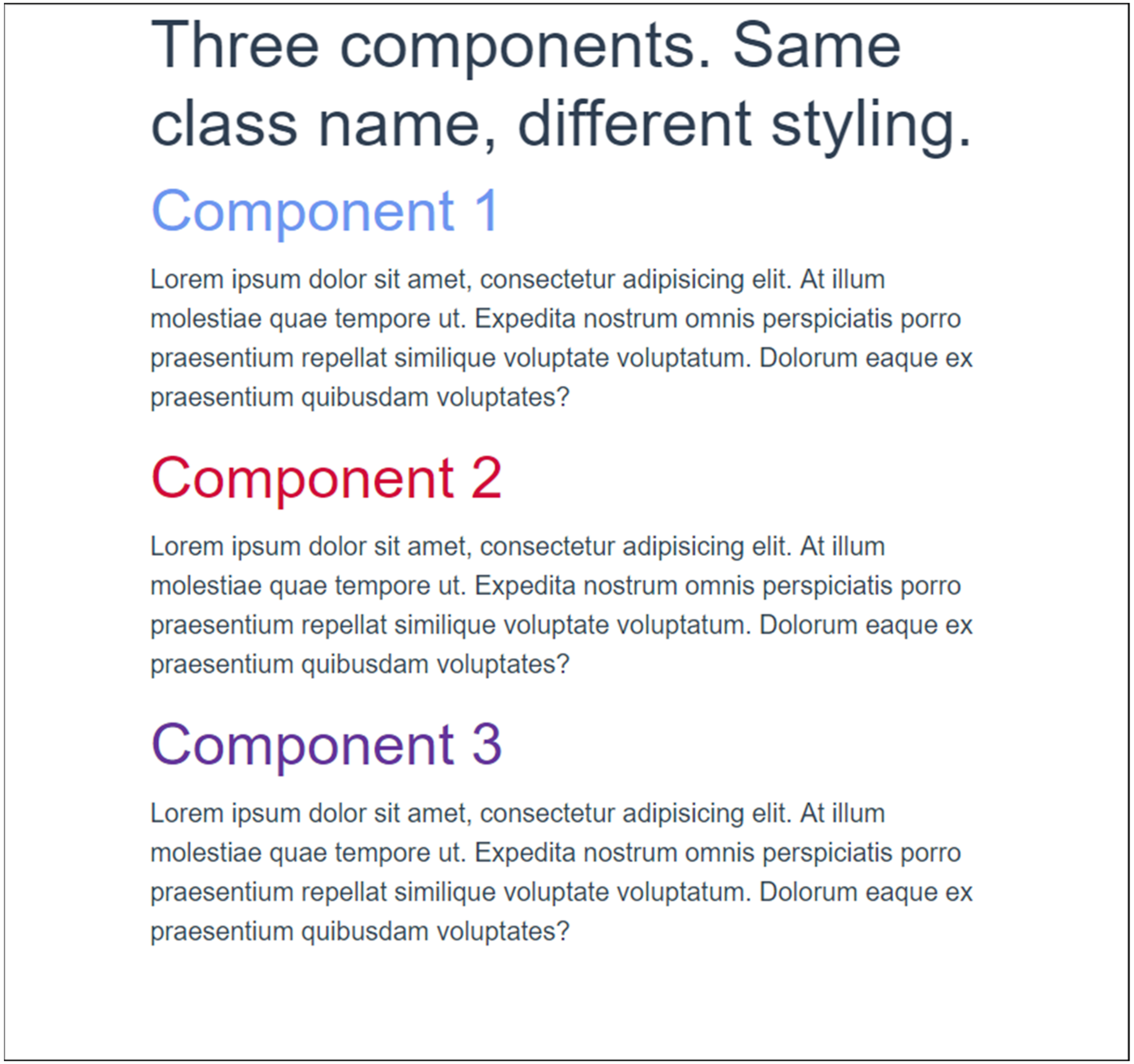
Using scoped styles

- To avoid naming collisions, it is best to add the scoped attribute to a style block inside a component
- Different components now can reuse the same classname without clashes.

```
<template>
    <div>
        <h2 class="heading">Component 1</h2>
        …
    </div>
</template>

<script>
   export default {
      name: "ComponentOne",
   }
</script>

<style scoped>
    .heading {
        font-size: 36px;
        color: cornflowerblue;
    }
</style>
```

```
<h2 class="heading">Component 2</h2>
<style scoped>
    .heading {
        font-size: 36px;
        color: crimson;
    }
</style>
```

```
<h2 class="heading">Component 3</h2>
<style scoped>
    .heading {
        font-size: 48px;
        color: rebeccapurple;
    }
</style>
```



Three components. Same class name, different styling.

Component 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

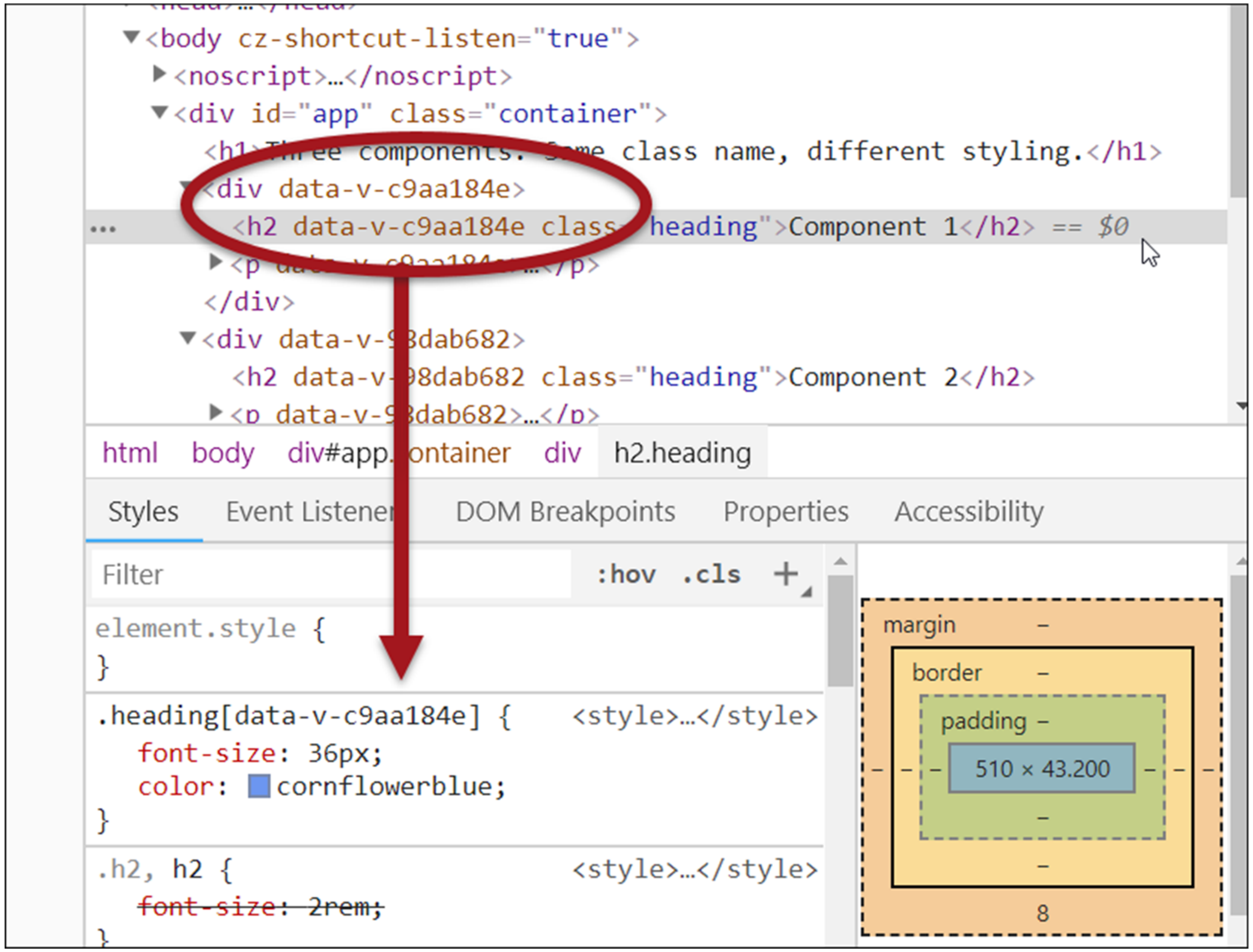
Component 2

Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

Component 3

Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

Vue adds (semi random) hashes to elements



```
  <head>…</head>
▼<body cz-shortcut-listen="true">
  ▶<noscript>…</noscript>
  ▼<div id="app" class="container">
      <h1>Th ee components. S me class name, different styling.</h1>
    ▼<div data-v-c9aa184e>
        <h2 data-v-c9aa184e class="heading">Component 1</h2> == $0
      ▶<p data-v-c9aa184e>…</p>
      </div>
    ▼<div data-v-98dab682>
        <h2 data-v-98dab682 class="heading">Component 2</h2>
      ▶<p data-v-98dab682>…</p>
```

```
html  body  div#app.container  div  h2.heading
```

```
element.style {
}
.heading[data-v-c9aa184e] {        <style>…</style>
   font-size: 36px;
   color: cornflowerblue;
}
.h2, h2 {                          <style>…</style>
   ~~font-size: 2rem;~~
}
```

General rules on styling

- Do not use global styles in components
- Only the top level component (`App.vue`) should have global styles
- You *can* use a generic CSS-framework like Bootstrap, Foundation, Vuetify, etc.

Conditionally applying styles

- Bind to the style attribut like so:
  - `v-bind:style=“{ …some-style…}”` or just
  - `:style=“{…some-style…}”`
  - For instance `:style=“{ border: ‘2px solid black’”}`
  - These are actually just CSS styles and notation!
- If your CSS-style has a hyphen in them, a special notation is needed:
  - `:style="{['background-color']: 'lightBlue’}”`
  - or use camelCase notation:
  - `:style="{backgroundColor: 'lightBlue'}"`

Making the style conditional

- For instance: we only want the style to be applied if the cost of a trip is less than 1000
- We can just bind to the HTML `:style` property
- For the value: use a computed property, or method.
- Let the computed property or method return a valid CSS style object

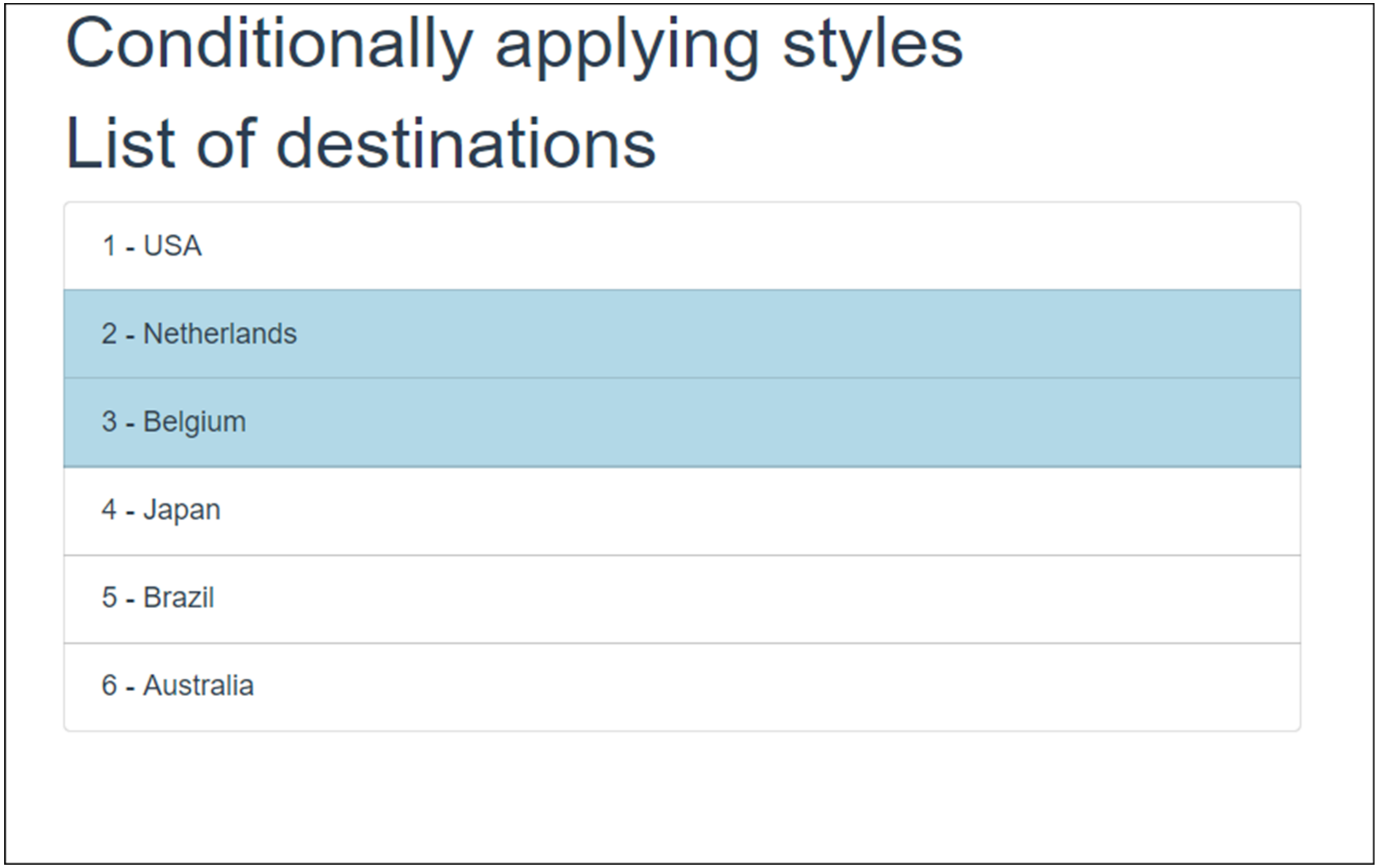
```
:style="{backgroundColor: 'lightBlue'}"
```

This works, but it is not conditional

This example: using a method

```
<li class="list-group-item"
    :style="highlightBackground(index)"
    v-for="(country, index) in data.countries" :key="country.id">
        {{ country.id }} - {{country.name}}
</li>
```

```
methods:{
   highlightBackground(index){
      return {
         backgroundColor:
               this.data.countries[index].cost < 1000 ?
                   'lightBlue' :
                   'transparent'
      }
    }
}
```



Conditionally applying styles

List of destinations

1 - USA

2 - Netherlands

3 - Belgium

4 - Japan

5 - Brazil

6 - Australia

Using v-model

Two-way databinding with Vue

Using v-model to select changes

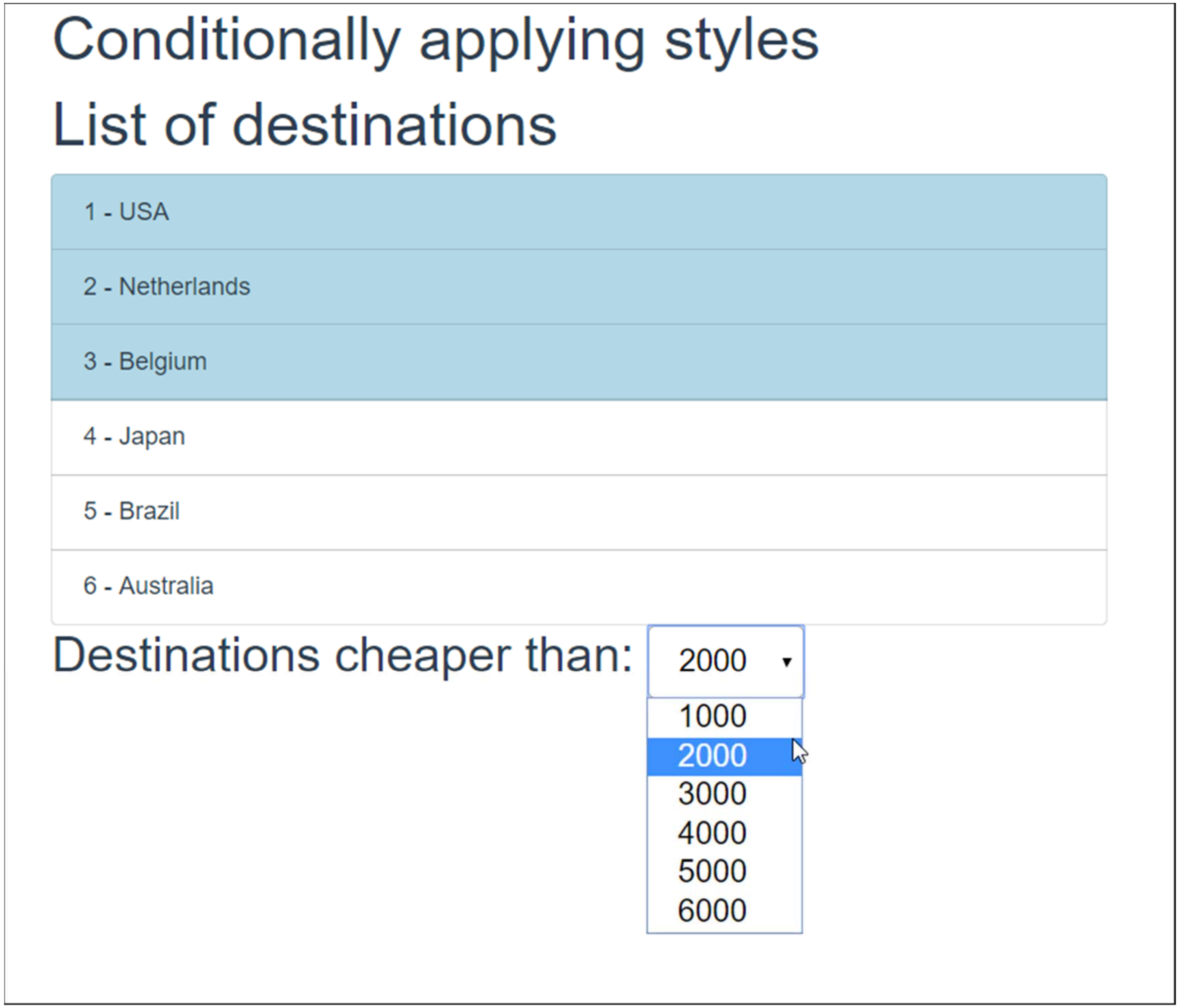
“You can use the `v-model` directive to create two-way data bindings on form input, textarea, and select elements. It automatically picks the correct way to update the element based on the input type.”

https://vuejs.org/v2/guide/forms.html

Using v-model on a selection list

```html
<h2>Destinations cheaper than:
    <select class="form-control-lg" v-model="selectedCost">
        <option value="1000">1000</option>
        <option value="2000">2000</option>
        <option value="3000">3000</option>
        <option value="4000">4000</option>
        <option value="5000">5000</option>
        <option value="6000">6000</option>
    </select>
</h2>
```

```javascript
data() {
   return {
      …
      selectedCost: 1000
   }
},
methods: {
   highlightBackground(index) {
      return {
         backgroundColor:
            this.data.countries[index].cost < this.selectedCost ?
               'lightBlue' :
               'transparent'
      }
   }
}
```



Conditionally applying styles

List of destinations

1 - USA

2 - Netherlands

3 - Belgium

4 - Japan

5 - Brazil

6 - Australia

Destinations cheaper than: 2000

Conditionally applying classes

- Most of the times it is better to use CSS classes instead of inline styles
- Class binding is an object where the keys are the name of the class you want to toggle.
- You set the value to a boolean expression that should evaluate to `true` or `false`
  - If `true`, the class is applied
  - If `false`, the class is removed from the element
  - Of course this is all dynamic

Same functionality – with class binding

Create a CSS class:

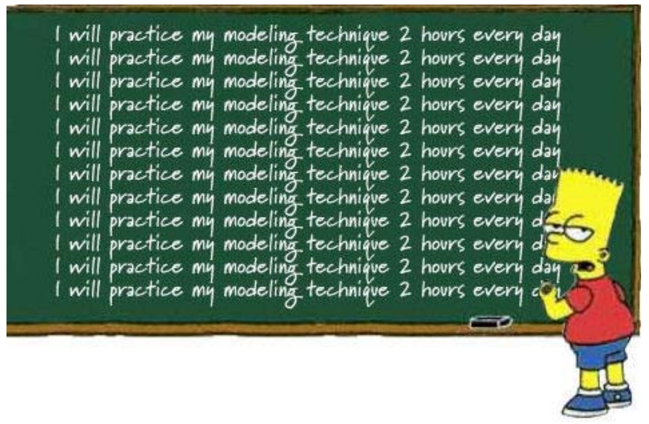
```
<style scoped>
    .lightblueBackground {
        background-color: lightblue;
    }
</style>
```

Apply the class conditionally in HTML:

```
:class="{'lightblueBackground': country.cost < selectedCost }"
```

Workshop

- Create a component with a `<button>` and a `<div>`
- if the button is clicked, the class of the div is toggled
  - First – use conditionale styles
  - Second – use conditional classes
- Add a `<div>`. If you hover the mouse over the div, toggle a class to highlight it
- Example: `140…/…/ConditionalClass.vue`



Using and creating Filters

Formatting UI elements using a pipe/filter

What are filters?

“Vue.js allows you to define filters that can be used to apply common text formatting. (...) Filters should be appended to the end of the JavaScript expression, denoted by the “pipe” symbol”

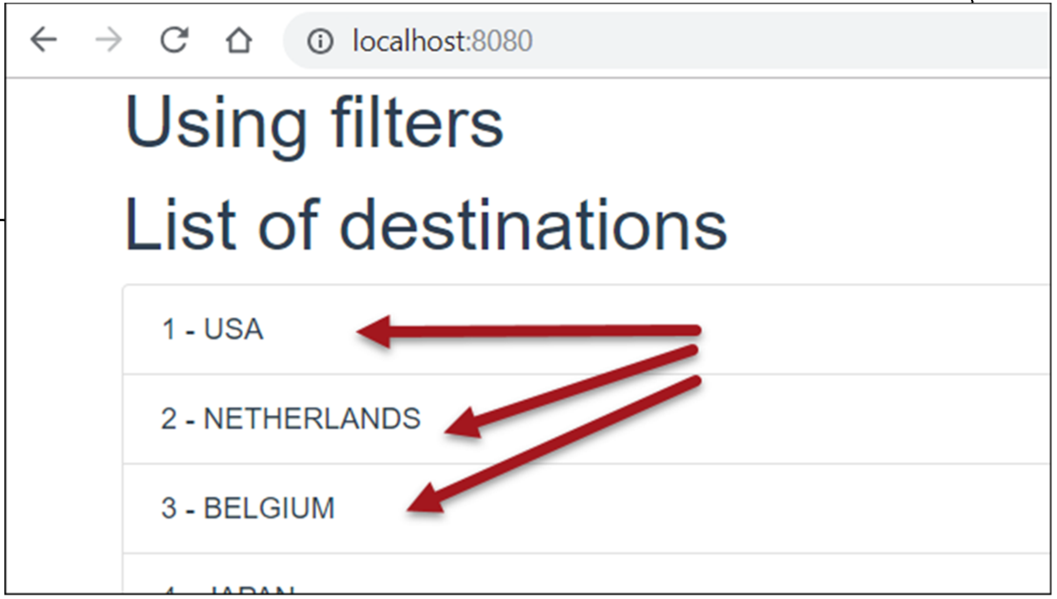
Using a filter

- Inside data binding expressions:
  - `{{country.name | uppercase }}`
- Inside `v-bind:` or `:`-expressions
  - `<div :id="rawId | formatId">…</div>`
- Filters can be declared
  - Locally to a component
  - Globally before creating a Vue instance
- Filters may be chained
  - `{{ message | filterA | filterB }}`

Creating a filter

- Vue doesn’t come with default filters.
- You always have to create them yourself

```
// creating a local filter, called 'uppercase'. This is used in the UI
filters:{
   uppercase(value){
      if(!value){
         return;
        }
      return value.toUpperCase();
    }
}
```


localhost:8080

Using filters

List of destinations

1 - USA

2 - NETHERLANDS

3 - BELGIUM

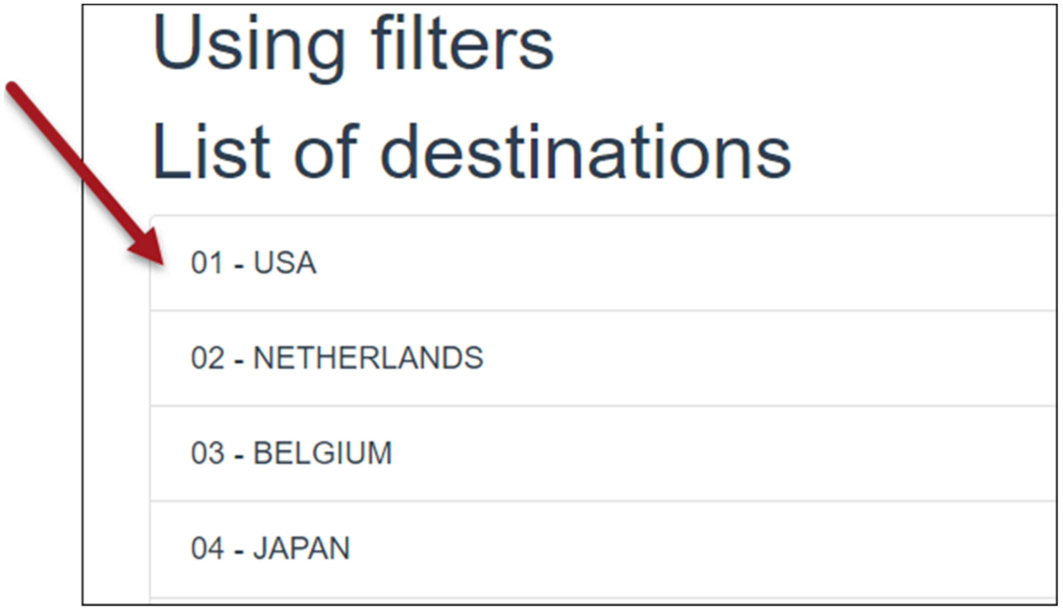
https://vuejs.org/v2/guide/filters.html

Creating a global filter

Create global filter in `main.js`, or in separate file and import in `main.js`

```
// Defining a global filter, before creating the Vue instance
// This assigns a leading zero if id < 10
Vue.filter('formatId', function (value) {
  if(!value)return;
  return value >= 10 ? value : '0' + value
});
```

```
{{ country.id | formatId }}
```


Using filters

List of destinations

01 - USA

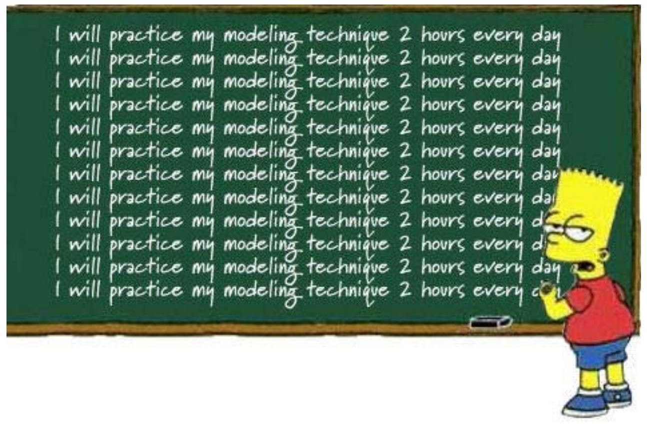
02 - NETHERLANDS

03 - BELGIUM

04 - JAPAN

Workshop - filters

- Create a local filter in a component.
- It should reverse the input given.
  - i.e. if the inputstring is `Hello World`, it should print `olleH dlroW`
  - Notation can be like `{{ inputString | reverse }}`
  - Search the internet for reversing strings in JavaScript!
- Test the filter and move it to a global filter.
  - Import it in the component and check if it still works
- General example on using filters:
  - `../170-filters`.


I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day

Checkpoint

- You know the difference between global styles and scoped styles
- You know how to apply styles and classes conditionally
- You know about mixins and filters and when/how to apply them