Vue Fundamentals

Module – Data and bindings

Peter Kassenaar –
info@kassenaar.com

For visuals – add Bootstrap

- There are (a lot!) of other Vue-optimized UI-frameworks
  - We can cover them later
  - We now just add default Bootstrap to get some basic styling

https://getbootstrap.com/

Adding Bootstrap to a Vue project

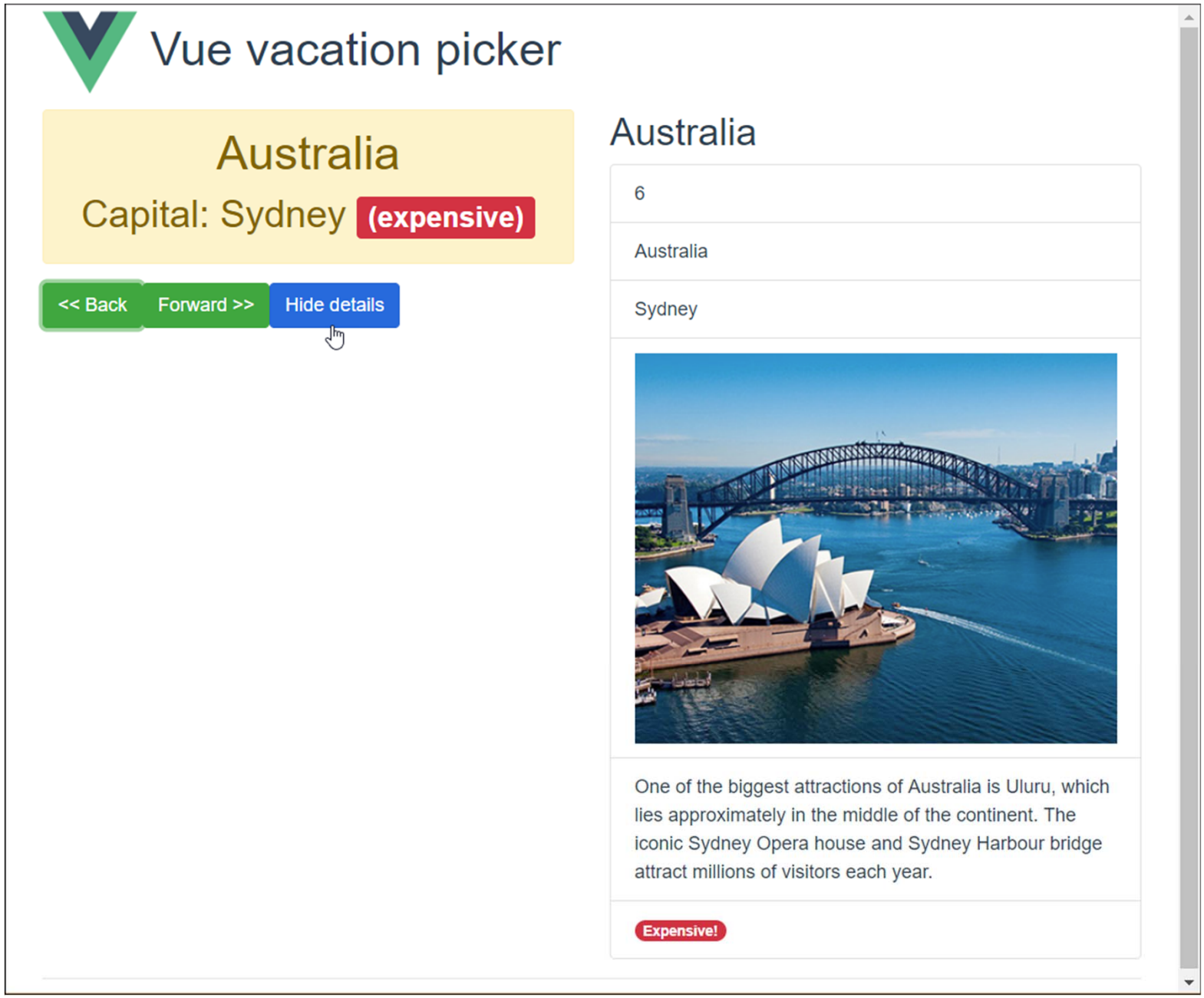
```
npm install bootstrap [--save]
```

- `Bootstrap` for basic styling and components
- `JQuery` and `Popper.js` for components functionality (Bootstrap 4)
- NO NEED TO INSTALL jQuery and popper in Bootstrap 5+

In `main.js`, add:

```
// Bootstrap styling
import 'bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
```

What are we going to build



Data binding

- Vue knows four ways of data binding:
  - Simple Data binding Expressions: `{{ … }}`
  - Attribute binding with `v-bind:`
  - Event binding with `v-on:`
  - Two-way binding with `v-model`

Building an app: VactionPicker

- We’re going to build a simple vacation picker.
  - Data comes from a separate `.js`-file (but will come from a real db in the future!)
- Requirements
  1. User can cycle through different destination countries
  2. User can show/hide details for each destination
  3. User can add trips to countries to a shopping cart
  4. Shopping cart can be send to the backend for processing

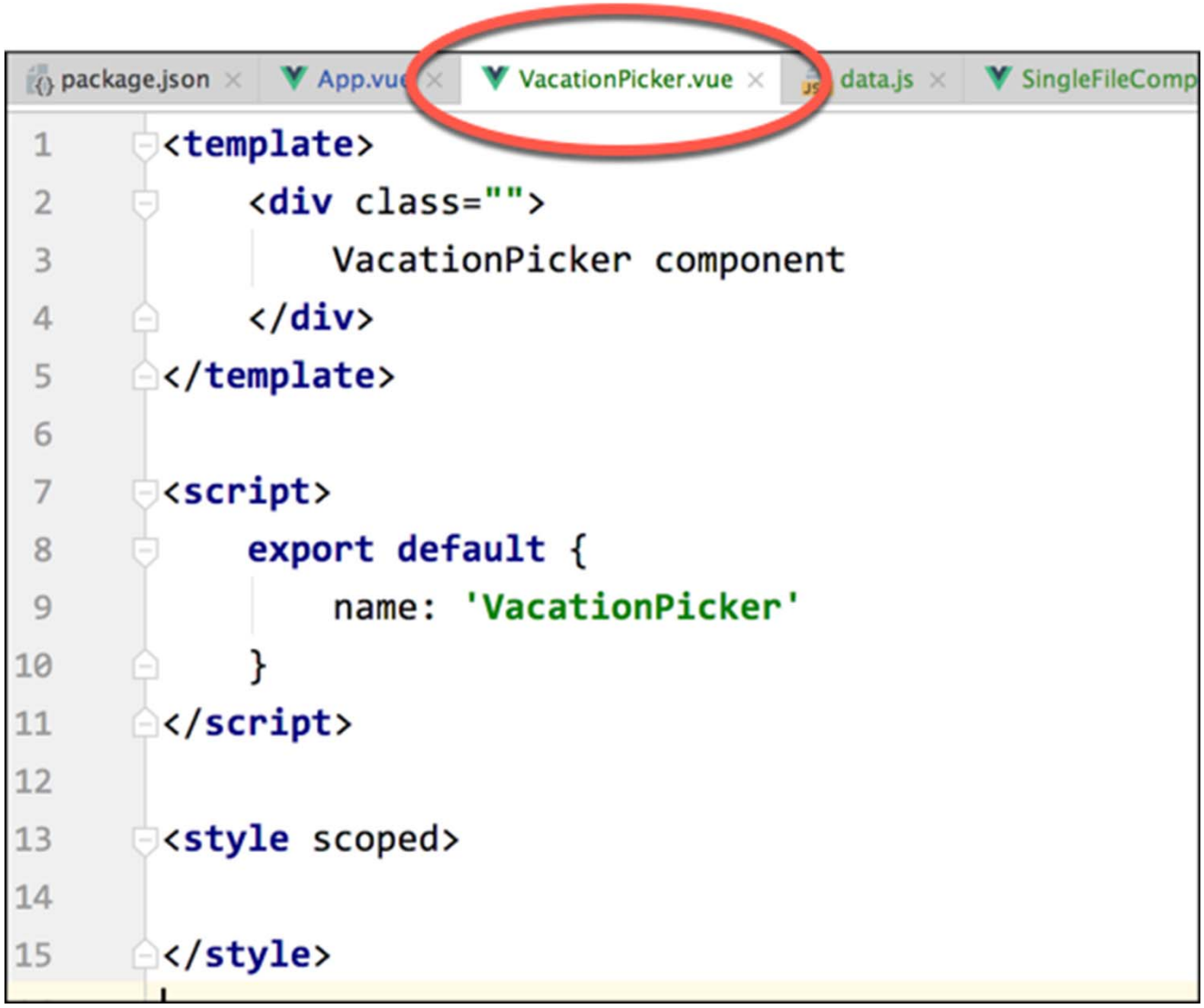
Creating the data file

```
// data.js - holding an array of country/capital data.
// This of course will come from a real db in the future.
const data = {
   countries: [
      {
         id: 1,
         name: 'USA',
         capital: 'Washington',
         cost: 1250,
         details: 'United States are among the most visited country in the world.',
         img: 'washington.jpg'
      },
      {
         id: 2,
         name: 'Netherlands',
         capital: 'Amsterdam',
         cost: 795,
         details: 'The capital of the Netherlands, Amsterdam, is over 1000 years old.',
         img: 'amsterdam.jpg'
      },
      { …
}
```

Our example: a simple JavaScript object, holding an array with countries and some (fake) data, don't forget to `export` it!

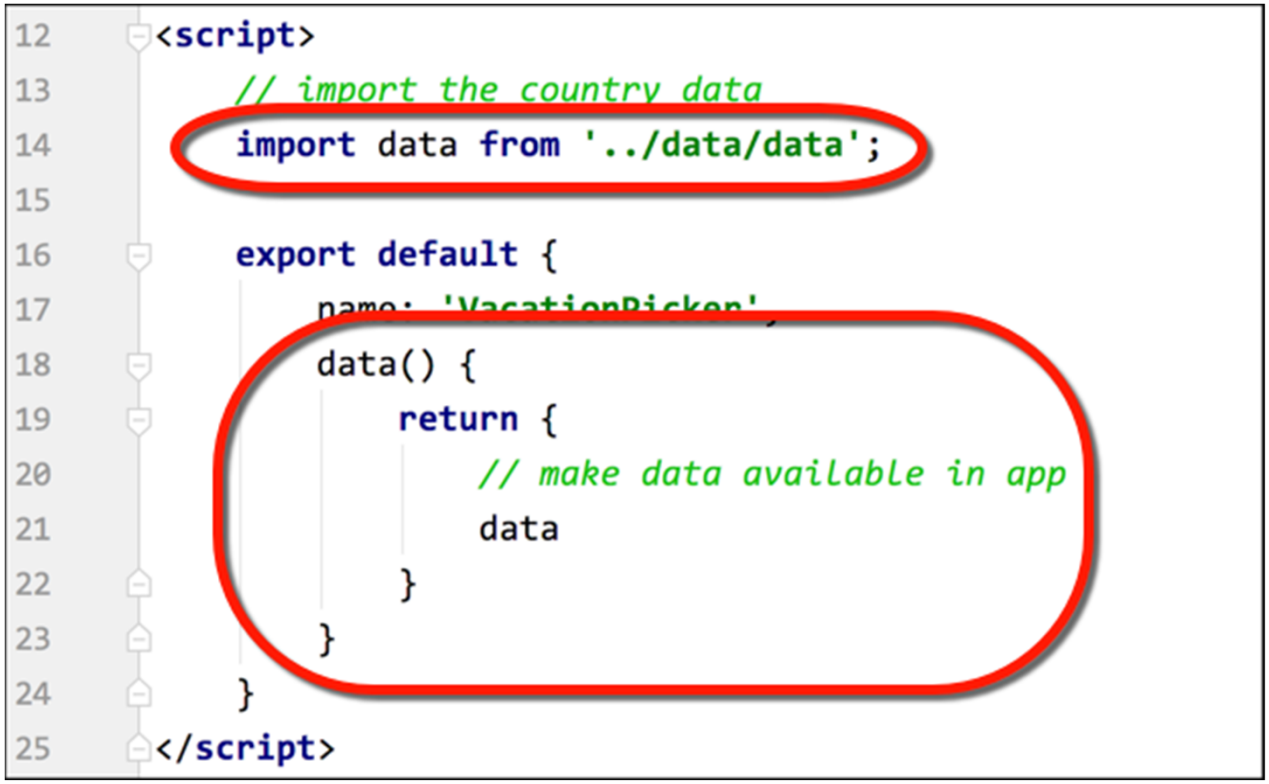
Importing the data

Create a new component, `VacationPicker.vue` with default content.



Importing data

- Use default `import` statement for the `data.js` file
- Data is made available via a `data:{…}` property on the component



Best practice for data

*"**Component data must be a function.***

When using the `data` property on a component (i.e. anywhere except on `new Vue`), the value must be a function that returns an object."

https://vuejs.org/v2/style-guide/#Component-data-essential

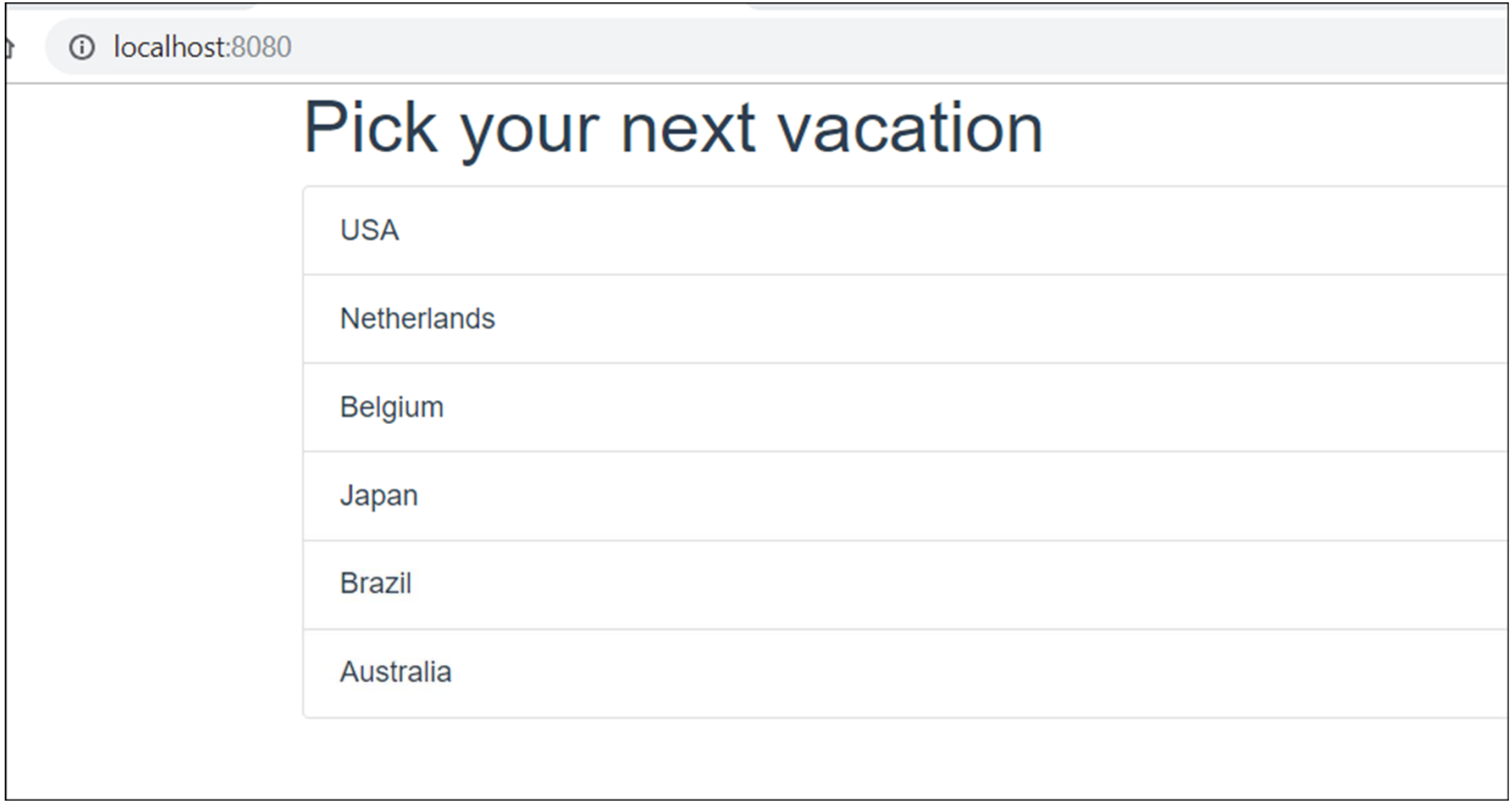
Binding to the data

Use the `v-for` directive to render a list of items, based on an array

```
<ul class="list-group">
    <li class="list-group-item" v-for="country in data.countries">
       {{country.name}}
    </li>
</ul>
```

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

Result

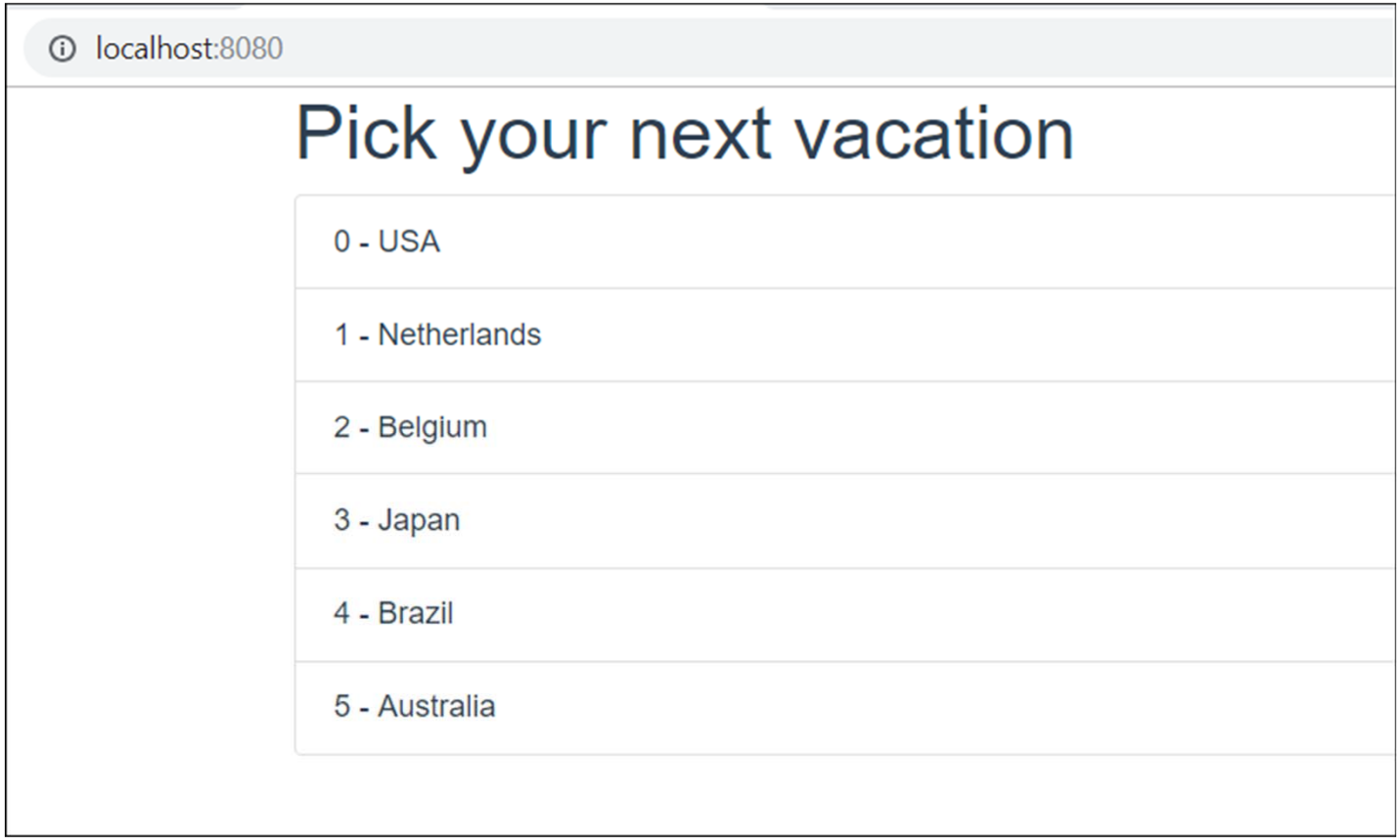


(assumes you updated `App.vue` to include the `VacationPicker` component and update/remove some styles)

Data binding expression

- Inside `v-for` blocks we have full access to parent scope properties.
  - Use the data binding expression `{{ … }}` to bind to properties
- `v-for` also supports an optional second argument for the `index` of the current item.

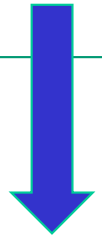
```
<li v-for="(country, index) in data.countries">
   {{ index }} - {{country.name}}
</li>
```



More data binding

- You can bind to any property exposed by the `data` function:

```
data() {
   return {
      // make data available in app
      data,
      header: 'Pick your next vacation'
   }
}
```



```
<h1>{{ header }}</h1>
```

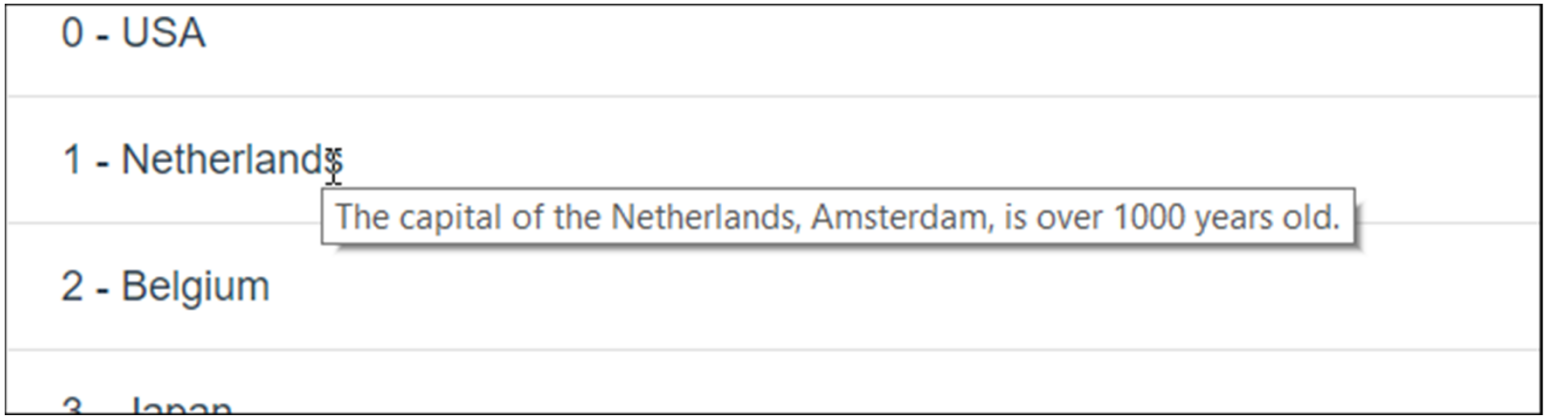
Using v-bind

Creating dynamic attributes on HTML- and custom elements

Used more often v-bind:

- Add `v-bind:` as prefix to an HTML-property to set it's content
- If you want to add an `id` and `title` attribute for example:

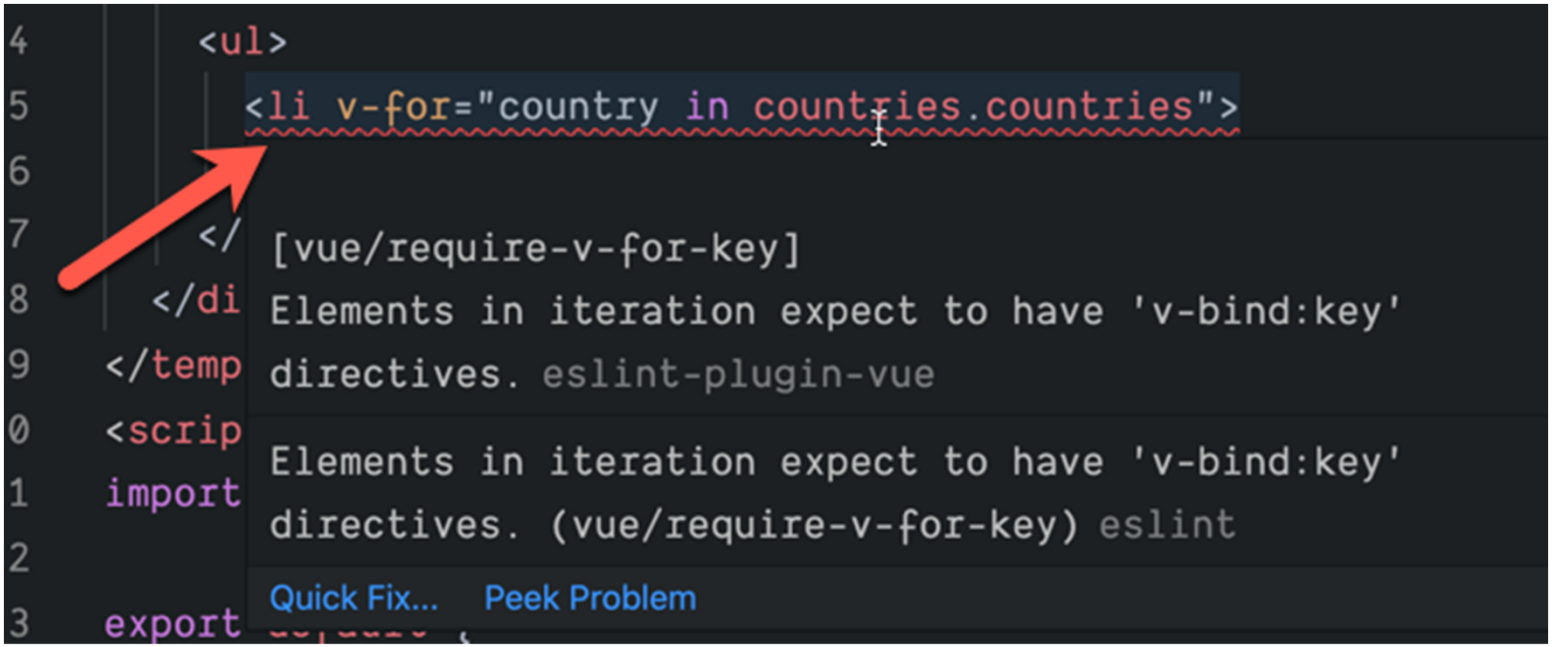
```
<span v-bind:id="country.id"
      v-bind:title="country.details">
    {{ index}} - {{country.name}}
</span>
```



Lists: using the v-bind:key property

```
    <ul>
      <li v-for="country in countries.countries">
        {{ country.id}} - {{ country.name }}</li>
    </ul>
</div>
```

The editor is complaining it is missing a `v-bind:key` directive here



Why the key attribute?

- Create a dynamic key attribute for each element in a repeated collection
- Vue needs to be able (as other frameworks) to uniquely identify each element in a list
  - Shadow DOM! Or Virtual DOM
- Solution: use the `v-bind:key` attribute

```
<li class="list-group-item"
    v-for="country in data.countries"
    v-bind:key="country.id">
```

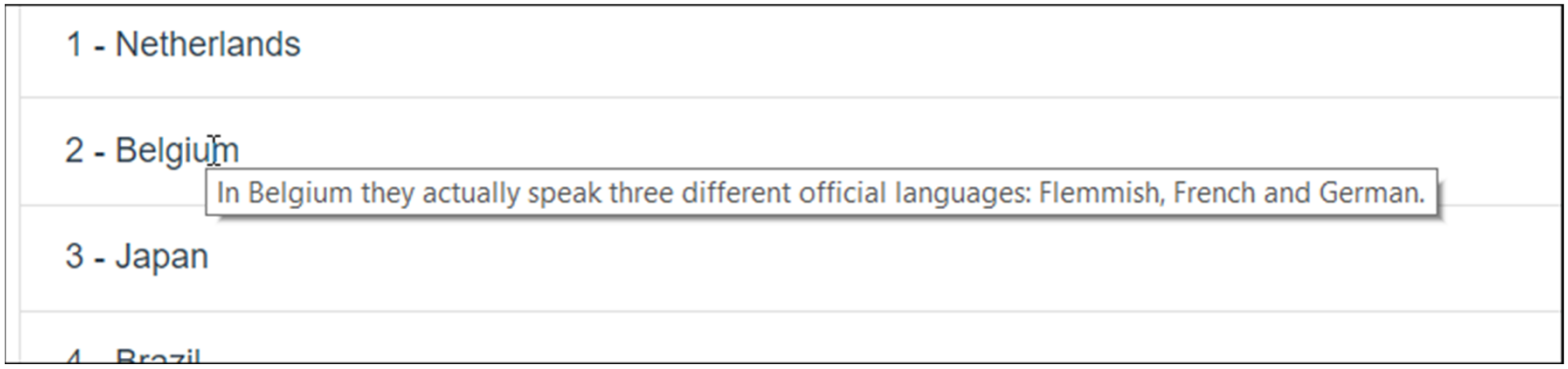
Summary

The `v-bind:` notation can be used on any attribute and it is used every time you need the attribute to be dynamic (e.g. adjusted from within code)

Using shorthand notation :

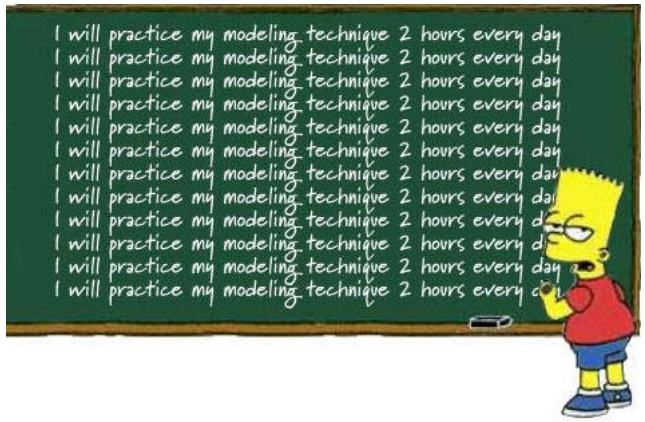
Same effect: using just : as a shorthand notation for v-bind:

```
<span :id="country.id"
      :title="country.details">
    {{ index}} - {{country.name}}
</span>
```



Workshop

- Add a data file to your application.
  - You can use `countryData.js` as an example, or build your own data file
  - Use for instance `CustomerData.js` as a blueprint and add your own data
- Import the data file to your application
- Show its contents in a new component, or inside `VacationPicker`, using `v-for`
- Create a `v-bind:key` binding for your data
- Example: `../103-databinding`
- Optional: Experiment with `v-text` and `v-html`



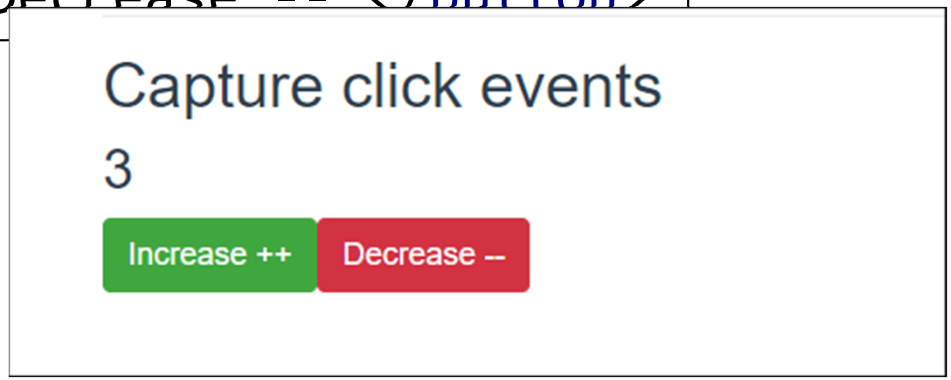
Using event binding

Handling events from the user interface

Binding events with v-on

- You can bind to DOM-events using the `v-on:` syntax
  - Events like `click`, `blur`, `focus`, `mouseover`, etc
  - https://developer.mozilla.org/en-US/docs/Web/Events
  - (add the `counter` variable to the `data` property)

```
<h3>{{ counter }}</h3>
<button v-on:click="counter++">Increase ++ </button>
<button v-on:click="counter--">Decrease -- </button>
```

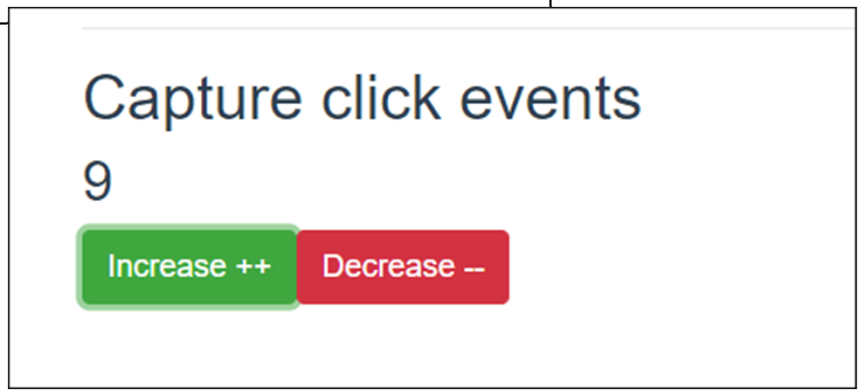


Or, use the shorthand notation @

```
<h3>{{ counter }}</h3>

<button @click="counter++">Increase ++ </button>

<button @click="counter--">Decrease -- </button>
```



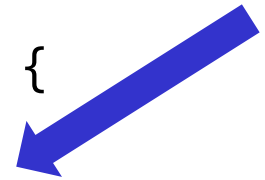
More likely, you'll be calling a function

- Functions for components are called `methods`
- They are defined on the component script block
- Methods can accept parameters, like any other function
- Refer to the data of the component using `this.<propertyName>`

```
    <h3>{{ counter }}</h3>
    <button @click="increase()">Increase ++ </button>
    <button @click="decrease()">Decrease -- </button>
    …
</template>

<script>
export default {
   …,
   methods:{
      increase(){
         this.counter++;
      },
      decrease(){
          this.counter--;
      }
   }
  …

</script>
```

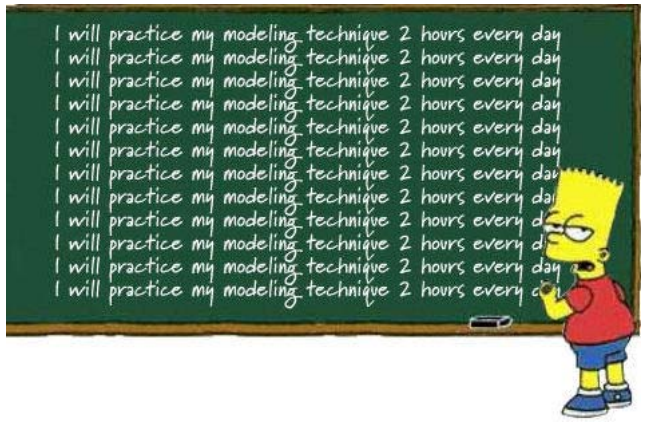


```
methods:{
   // long-hand notation:
   increase: function(){
      this.counter++;
   },
   decrease: function(){
      …
   }
}
```

Vue Style guide: “use shorthand function notation”

Workshop

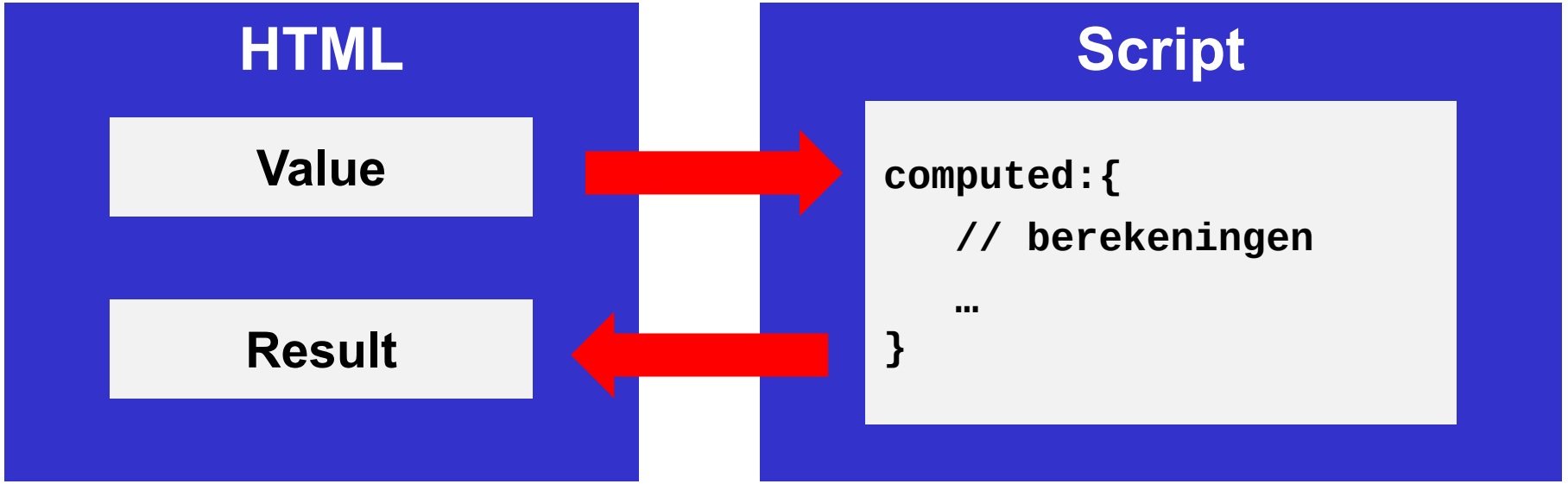
- Add a button to the page that toggles the `disabled` attribute of another button
  - Tip: create a dynamic `:disabled="isDisabled"` property
- When a specific country in the list is clicked, show it's name and capital in a JavaScript `alert()` box (or use `console.log`)
- Log the name of the current country and it's index/id to the console on a mouseover
- Use `v-on:` and @-notation
- Example: `../104-eventbinding`



Computed properties

Binding to more complex properties and update them on changing

Architecture of computed properties



Selecting a specific country

- We want to select a `country` (or `product`, or `employee`, or whatever) when a user clicks on them
- Create a `computed{…}` object on the component that composes and returns the requested data
  - Here: we're setting a property that holds the currently selected index
  - Update the computed property when the index changes
- Computed properties avoid having to do complex computations inside the HTML-view!
- A computed property is only evaluated once its dependencies change

1. In the view

```
<li @click="selectCountry(country)"
    class="list-group-item" v-for="country in data.countries">
```

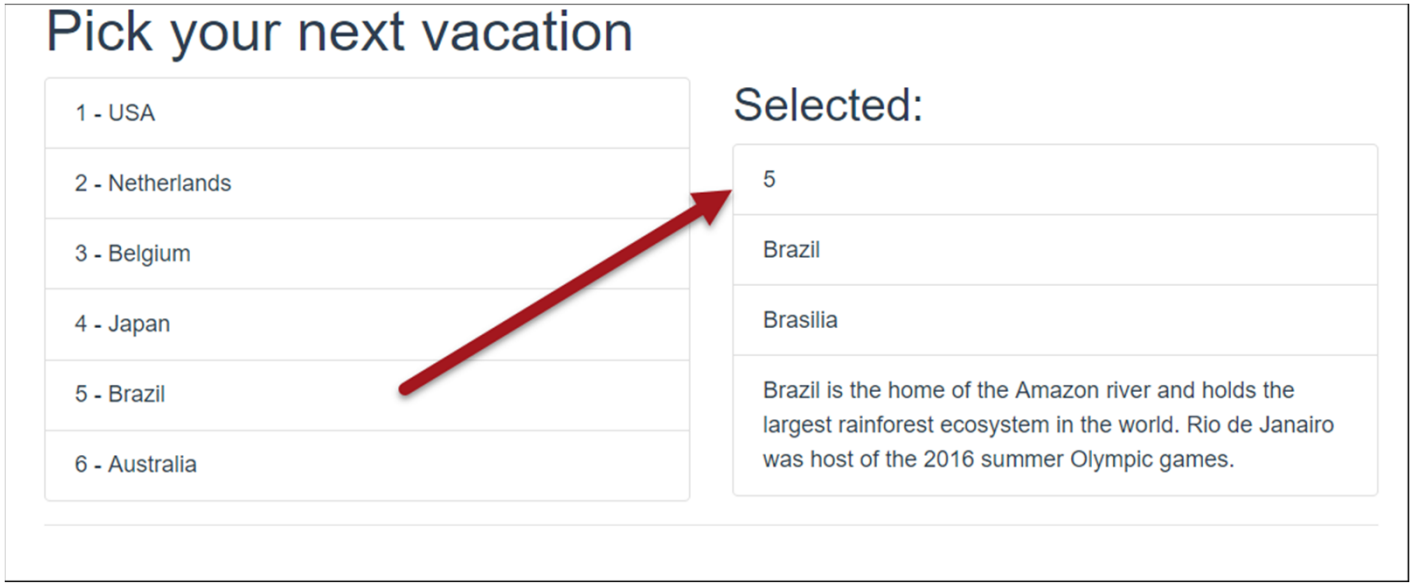
2. On the component

```
methods: {
   selectCountry(country) {
      this.selectedCountryIndex = this.data.countries.indexOf(country);
   }
},
computed: {
   selectedCountry() {
      return {
         // use the spread operator to return all properties
         ...this.data.countries[this.selectedCountryIndex]
      }
   }
```

3. On the component again

```
<h2>Selected:</h2>
<ul class="list-group">
    <li class="list-group-item">{{ selectedCountry.id}}</li>
    <li class="list-group-item">{{ selectedCountry.name}}</li>
    <li class="list-group-item">{{ selectedCountry.capital}}</li>
    <li class="list-group-item">{{ selectedCountry.details}}</li>
</ul>
```

4. Result



Computed properties or methods?

“Instead of a computed property, we can define the same function as a method instead. For the end result, the two approaches are indeed exactly the same. However, the difference is that **computed properties are cached based on their dependencies.** A computed property will only re-evaluate when some of its dependencies have changed. ”

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

`v-if` and more computed properties

- Show a badge when a destination is expensive
- Use `v-if` to render a DOM element if a certain condition is `true`
  - You can use `v-else` and `v-else-if` to render elements (though used less often)

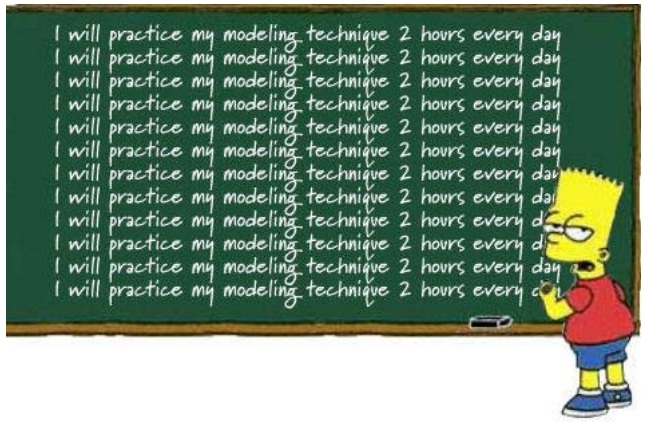
```
<li class="list-group-item" v-if="isExpensive">
    <span class="badge badge-danger badge-pill">Expensive!</span>
</li>
```

```
isExpensive() {
   return data.countries[this.selectedCountryIndex].cost > 4000;
}
```

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

Workshop

- Create a detail view for your own app.
  - If an element/data is clicked, show details in the UI
- Create a computed property that show a destination as ‘on sale’ if the cost is less than 1000.
- Replace the text `Selected` with the actual name of the selected country
- Use `v-if` where appropriate
- Example: `../105-computed-properties`



`v-if` vs. `v-show`

- There is also a `v-show` directive in Vue
  - `v-show` – or `:hidden` - the element is rendered in the DOM, but hidden afterwards
  - `v-if` – the element is not rendered at all-in the DOM
- Documentation:

"Generally speaking, v-if has higher toggle costs while v-show has higher initial render costs.

So prefer v-show if you need to toggle something very often, and prefer v-if if the condition is unlikely to change at runtime."

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

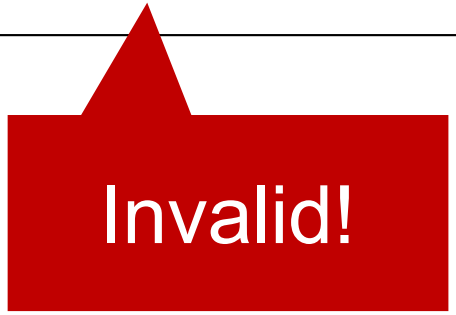
Binding to images

We need a bit of WebPack magic here...

Dynamically binding to images

- Vue can not simply interpolate the name of an external resource and re-use it for binding
  - For instance, the `src` attribute of an image.
- So this is invalid:

```
<li class="list-group-item">
    <img :src="./assets/{{selectedCountry.img}}">
</li>
```



WebPack to the rescue

- Because WebPack builds JavaScript strings of everything, it needs to be able to determine the location of the requested file
- Create a method that returns a string with the correct location:

```
<li class="list-group-item">
    <img :src="getImgUrl(selectedCountry.img)"
         :alt="selectedCountry.img"
         class="img-fluid" >
</li>
```

```
methods: {
  …,
  getImgUrl(img){
     console.log(img);
     return require('../assets/countries/' + img);
  }
…
```

Pick your next vacation

1 - USA
2 - Netherlands
3 - Belgium
4 - Japan
5 - Brazil
6 - Australia

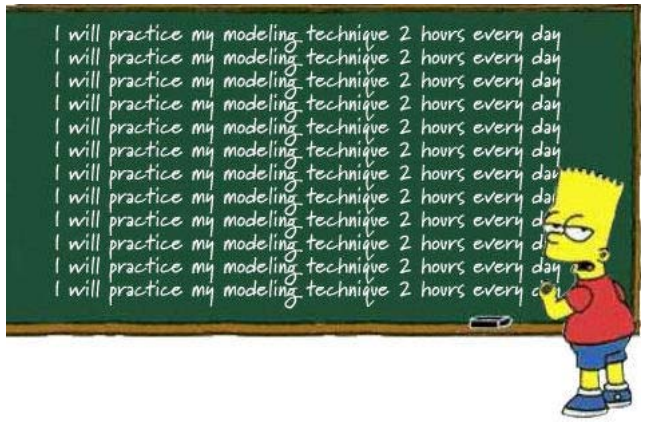
Selected:

1
USA
Washington

United States are among the most visited country in the world.

Workshop

- Static:
  - Create an array of images in your app, – has to be a complete URL!
  - Render them in a `v-for` loop. This can be done at any time.
  - See for instance https://www.youtube.com/watch?v=B8rVIxQm8Cs.
- Dynamic
  - Add/use an image as property of an object (like with the countries)
  - Render them conditionally/dynamically in the UI, like in the previous slides
  - Use the WebPack `require()` function.
- Example: `../106-image-binding`

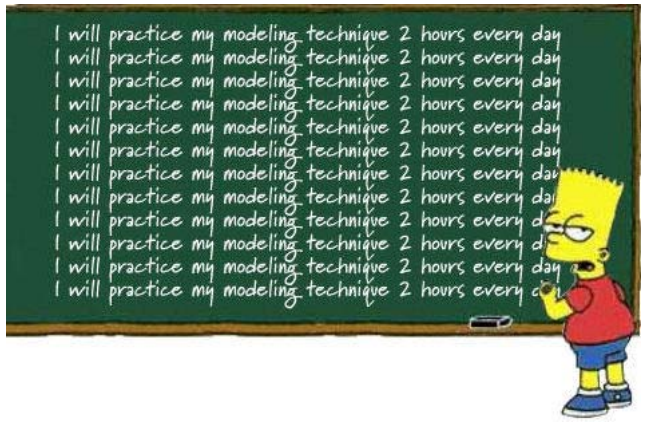


Checkpoint

- You can import and use third party libraries
- You know the most important data binding concepts of Vue
- `{{ .. }}` for simple data binding expressions
- `v-bind:` or simply `:` for attribute binding
- `v-on:` or simply `@` for event binding
- `v-if` for conditional rendering
- Computed properties that update on a triggered change
- Some WebPack stuff to render images in the UI

Final Workshop - 1

- Create the app shown in the beginning of the slides
  - One country is visible at a time
  - Its name and capital are shown
  - There are two buttons to cycle through the countries
  - User can hide/show details with a button
  - There are badges if visiting the country is Expensive, or On Sale
- Example: `../110-data-binding-wrap-up`
  - But first try it yourself



Final Workshop – 2

- Create the same app as before, but in TypeScript
  - Start from scratch, choose `TypeScript` as an option
  - Create your components as TypeScript classes
  - Create properties, methods, computed properties (getters) and load the data.
- No Example available