Vue.js Exercises

Setup

Unless otherwise specified, the following setup can be assumed for all exercises.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Single-page Application</title>
    <script src="vue.js"></script>
  </head>
  <body>
    <div id="app"></div>
    <script>
      const app = Vue.createApp({
        //...
      });
      const vm = app.mount('#app');
    </script>
  </body>
</html>
```

Exercises

1. Create web page that counts from 0 to infinity. Create a vue instance **`vm`** with
   - a **`counter`** property in its data option, which is initially set to `0`,
   - a template that *interpolates* the **`counter`**, and
   - use the following piece of code to increase the counter.

   ```
   setInterval(() => vm.counter++, 100);
   ```

   0

   Figure 1: Resulting web page at its initial state.

   30

   Figure 2: Resulting web page after 3 seconds.

2. Create a vue instance that has a template with
   - two **`<input>`** controls *bound* to variables **`a`** and **`b`**, respectively, and
   - after these two controls, an interpolation of the addition of **`a`** and **`b`**.

   23 1 24

   Figure 3: Example of the addition of `23` and `1`.

   Note: the term *bound*, as employed here, means that there is a data bindings between the variable and the form control. It may either be an one-way or a two-way binding (with **`v-bind`** or **`v-model`**, respectively).

   Hint: use **`parseFloat`**.

3. Create a vue instance with a single **`<button>`** that disappears when clicked.

Figure 4: Initial state.

Figure 5: After clicking the button.

4. Create a vue instance with an empty **`<input>`** text box. The text box clears itself when its text length reaches 5 characters (or surpasses that number).

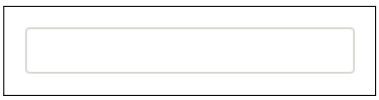

Figure 6: Initial state, and after writing five characters.

Figure 7: After writing four characters.

5. Create a vue instance with an empty **`<input>`** text box. The text box turns red when keys are pressed, and restores its original color upon key release.

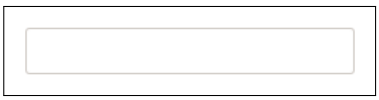

Figure 8: Initial state.

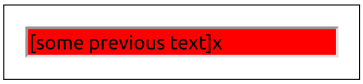

Figure 9: After pressing (but not releasing) the ‘x’ key.

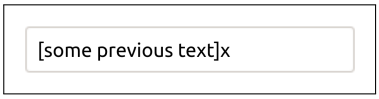

Figure 10: After releasing the ‘x’ key.

Hint: **`v-on:keydown`**.

6. Using the following template, create an instance that changes the ‘redness’ of the **`AM I RED?`** text according to the value in the range slider. Hide the **`YES!`** text when redness is under 70%.

```
<div>
  <div style="color: hsl(0,??%,50%)">AM I RED?</div>
  <input type="range" min="0" max="100">
  <div>YES!</div>
</div>
```

Figure 11: When 'redness' is `0`.

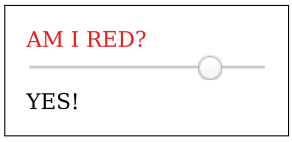

Figure 12: When 'redness' is `70`.

7. Create a vue instance with:

   - the properties **a**, **b**, **c** and **d** in its data option (initially set to `false`), and
   - a template with an `<input type=checkbox>` bound to **a**, followed by the interpolation of the four variables (**a**, **b**, **c** and **d**).

   Create a watch function for the variable **a** that sets **b** equal to **a**. Similarly, create a watch function for the variable **b** that sets **c** equal to **b**, and a watch function for the variable **c** that sets **d** equal to **c**.

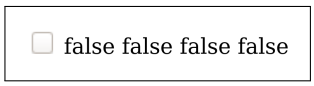

Figure 13: Initial state.

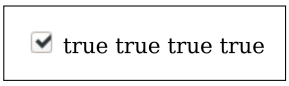

Figure 14: After switching the checkbox.

8. Create a vue instance that displays the following phone book as shown in the accompanying figure.

```
[
  { name: 'Jaime Sommers', phone: '311-555-2368' },
  { name: 'Ghostbusters', phone: '555-2368' },
  { name: 'Mr. Plow', phone: '636-555-3226' },
  { name: 'Gene Parmesan: Private Eye', phone: '555-0113' },
  { name: 'The A-Team', phone: '555-6162' },
]
```

Employ the following CCS style.

```
table { border-collapse: collapse; }
table th,td { border: 1px solid black; }
```

Name	Phone number
Jaime Sommers	311-555-2368
Ghostbusters	555-2368
Mr. Plow	636-555-3226
Gene Parmesan: Private Eye	555-0113
The A-Team	555-6162

Figure 15: How the phone book must be rendered.

Hints: `v-for` and the following html code.

```
<table><tr><th>Name</th><th>Phone number</th></tr>
<tr><td>{{item.name}}</td><td></td></tr>
</table>
```

9. Suppose this is a 'sempahore':

```
<div style="display: inline-block; width:30px;">
  <div style="height: 30px; background-color: indianRed"></div>
  <div style="height: 30px; background-color: khaki"></div>
  <div style="height: 30px; background-color: seagreen"></div>
</div>
```

Create a web page that:

- renders the semaphore in a vue template,
- has a **`state`** variable, which is an integer representing which light is on, and
- has a **`<button>`** that switches the semaphore **`state`**.

A value of `0` for **`state`** denotes a green light, a value of `1` denotes a yellow light, and a value of `2` denotes a red light. The initial state is `0`.

Use the following css colors to represent when lights are on: **`red`**, **`yellow`**, and **`lawngreen`**; and use the following to represent when lights are off: **`indianRed`**, **`khaki`**, and **`seagreen`**.

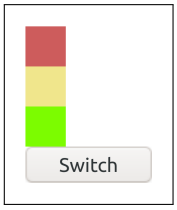

Figure 16: Initial state, and after 3 button clicks.

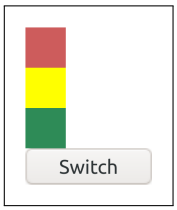

Figure 17: After 1 button click.

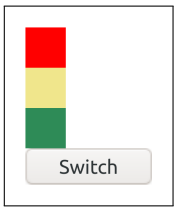

Figure 18: After 2 button clicks.

10. Extend the previous 'phone book' exercise by adding delete buttons. Add a third column with an individual delete button for each entry.

Name	Phone number	
Jaime Sommers	311-555-2368	Delete
Ghostbusters	555-2368	Delete
Mr. Plow	636-555-3226	Delete
Gene Parmesan: Private Eye	555-0113	Delete
The A-Team	555-6162	Delete

Figure 19: Initial state.

Name	Phone number	
Jaime Sommers	311-555-2368	Delete
Ghostbusters	555-2368	Delete
Gene Parmesan: Private Eye	555-0113	Delete
The A-Team	555-6162	Delete

Figure 20: After deleting the 'Mr. Plow' entry.

Hints:

- `list.splice`,
- `v-for="(item,index) in list"`.

11. Create the component `<words-to-list>` that transforms words into list items. Words given through the `words` attribute are transformed into `<li>` elements inside an `<ul>`.

    For example, `<words-to-list words="w1 w2 w3"></words-to-list>` is transformed into `<ul><li>w1</li><li>w2</li><li>w3</li></ul>`.

    Hint: `string.split(' ')`.

    Usage example:

```
Vue.createApp({
  template: `<words-to-list words="Lorem ipsum dolor sit amet">
    </words-to-list>`,
}).mount('#app');
```

- Lorem
- ipsum
- dolor
- sit
- amet

Figure 21: Result when `words` is `"Lorem ipsum dolor sit amet"`.

12. Create the component `<card>`, which is used to render user information. It is used as follows:

```
Vue.createApp({
  data: function() {
      return {
        person: {
            name: 'My Name',
            picture: `
```

```
                AAAAfFcSJAAAADUlEQVR42mM82Mz1HwAFqgJP3gasfwAAAABJRU5ErkJggg==`,
              email: 'me@somerandomdomain.com',
              phone: '+00 00 000 0000',
          }
      }
    },
    template: `<div style="display:flex;">
                <card v-bind:personal-data="person"></card>
               </div>`,
}).mount('#app');
```

For the previous data, the component template has to yield this final result:

```
<div class="card">
  <div>
    <img src="
          AAAAfFcSJAAAADUlEQVR42mM82Mz1HwAFqgJP3gasfwAAAABJRU5ErkJggg==">
  </div>
  <div><h1>My Name</h1></div>
  <div>me@somerandomdomain.com</div>
  <div>+00 00 000 0000</div>
</div>
```

Employ the following css style:

```
.card { font-family: Roboto; text-align: center; background: #ffbcbc;
  box-shadow: 6px 6px 8px #888; margin: 15px;}
.card div {padding: 10px; }
.card img { width: 100px; }
```

Figure 22: The **<card>** component, as employed in the previous example.

13. Create a **<switch-button>** component as follows:

    - The rendered component has to resemble the following html snippet.

```
<div style="border:solid;display:inline-block">
  <button>ON</button>
```

```
    <button disabled>OFF</button>
  </div>
```

- When the 'ON' button is clicked, the **on** event is dispatched, the 'ON' button is disabled, and the 'OFF' button is enabled.
- Similarly, when the 'OFF' button is clicked, the **off** event is dispatched, the 'OFF' button is disabled, and the 'ON' button is enabled.

Usage example:

```
Vue.createApp({
  data: () => ({ state: null }),
  template: `<div><switch-button
      v-on:on="state='just turned on'"
      v-on:off="state='just turned off'"
    ></switch-button> {{state}}
  </div>`,
}).mount('#app');
```

Figure 23: Initial state.

Figure 24: After clicking on the 'ON' button.

Figure 25: After clicking on the 'OFF' button.

14. Create a **<color-selector>** component as follows:

    - The rendered component has to resemble the following html snippet.

```
<div style="border:solid; display:flex;">
  <div style="background-color:#000; width:110px; height:110px;"></div>
  <div style="display:flex; flex-direction:column; padding:10px;">
    <div>R: <input type="range" min=0 max=255> red value</div>
    <div>G: <input type="range" min=0 max=255> green value</div>
    <div>B: <input type="range" min=0 max=255> blue value</div>
  </div>
</div>
```

    - When a new color is selected, the component has to emit a **color** event with the selected color value (in css format).

Usage example:

```
Vue.createApp({
  data: () => ({ color: null }),
  template: `<div style="border:solid red; display:flex;">
```

```
      <color-selector v-on:color="color = $event"></color-selector>
      <div v-bind:style="'color:' + color">TEXT</div>
    </div>`,
}).mount('#app');
```

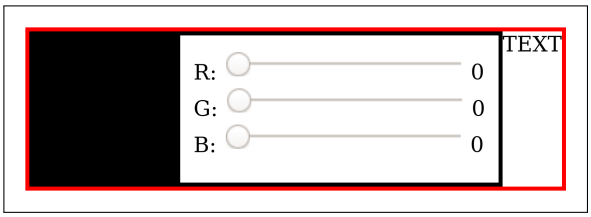

Figure 26: Initial state.

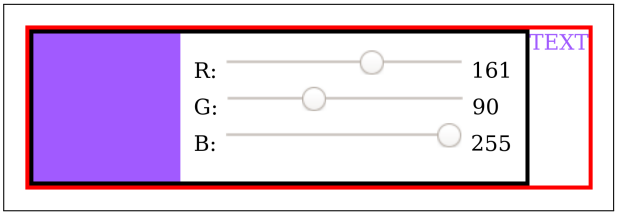

Figure 27: Result after some color selection.

15. Create a **<magic-input>** component that works like a regular input text box (**<input type=text>**), except that it turns upper case letters into lower case letters and viceversa. The component has to support **v-model**, and should be used as follows:

```
<magic-input v-model="modelVariable"></magic-input>
```

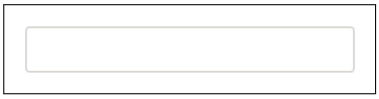

Figure 28: How an empty **<magic-input>** text box should look like.

Note that the **<magic-input>** has to display exactly what the user writes, and only change the text case in its model variable. For example, after typing the text **"Hola"** in **<magic-input v-model="modelVariable">**, the text box has to display the text **"Hola"** and the value of **modelVariable** has to be **"hOLA"**.

Similarly, when **modelVariable** is set to **"Test"**, the text box has to display **"tEST"**, as in the following example.

```
const vm = Vue.createApp({
  data: { modelVariable: "" },
  template: `<magic-input v-model="modelVariable"></magic-input>`,
}).mount('#app');
setInterval(() => vm.modelVariable = "Test", 1000);
```

Hints:

- Use the following case-switching snippet.

```
text.replace(/./g,
  x => x.toUpperCase() == x ? x.toLowerCase() : x.toUpperCase())
```

- Use the **created** option to initially set a data variable to the the value of the **value** prop.

- Watch for changes both in the prop variable and your data variable.

Usage example:

```
Vue.createApp({
  data: { text: "Hola" },
  template: `<div>
      <magic-input v-model="text"></magic-input>
      <input v-model="text">
      {{text}}
    </div>`,
}).mount('#app');
```

Figure 29: Initial state.

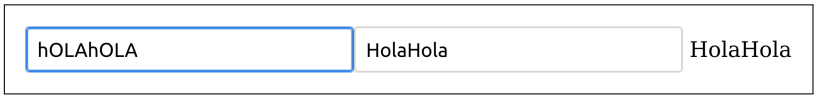

Figure 30: After appending 'hOLA' in the `<magic-input>`.

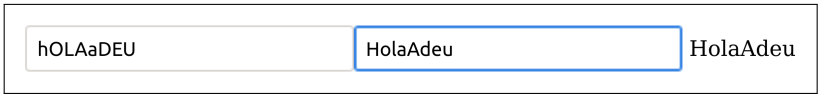

Figure 31: After appending 'Adeu' in the regular `<input>`. Note that the `<magic-input>` is updated automatically.