Chapter 3 – Reactivity Variables and Form Input Bindings

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

Reactivity Objects

Reactivity Variable with `ref()`

- Vue provides a `ref()` function which allows us to create reactive that **can hold any value type.**
- `ref()` takes the argument and returns it wrapped within **a ref object** with a `.value` property.
- When `ref` are accessed in the template, they are automatically "unwrapped" so there is no need to use `.value`

```
<script setup>
import { ref } from 'vue'
const msg = ref('hello, vue')
setTimeout(() => {
  msg.value = 'good bye'
  console.log(msg)
}, 3000)
</script>

<template>
  <p>Message: {{ msg }}</p>
</template>

<style></style>
```

Using `ref()` with Object and Array

//ref() with Object

```
<script setup>
import { ref} from 'vue'
const profile = ref({
  name: 'Evan You',
  creator: 'Vue.js’
})
console.log(profile.value.name)
console.log(profile.value.creator)

</script>
<template>
  <p>{{ profile.name }}</p>
  <p>{{ profile.creator }}</p>
</template>
<style></style>
```

//ref() with Array

```
<script setup>
import { ref } from 'vue'
const topics = ref(['template syntax',
'declarative rendering', 'ref function’])
console.log(topics.value[0])
console.log(topics.value[1])
console.log(topics.value[2])
// topics.value.forEach((topic) =>
console.log(topic))

</script>
<template>
  <div>{{ topics[0] }}</div>
  <div>{{ topics[1] }}</div>
  <div>{{ topics[2] }}</div>
</template>
<style></style>
```

Reactivity Variable with `reactive()`

- We can create a reactive object or array with the `reactive()` function.
- It only works for **object types** (objects, arrays, and collection types such as Map and Set).
- It **cannot** hold primitive types such as string, number or Boolean.

Using `reactive()` with Object and Array

//reactive() with Object

```
<script setup>
import { reactive} from 'vue'
const profile = reactive({
  name: 'Evan You',
  creator: 'Vue.js’
})
console.log(profile.name)
console.log(profile.creator)

</script>
<template>
  <p>{{ profile.name }}</p>
  <p>{{ profile.creator }}</p>
</template>
<style></style>
```

//reactive() with Array

```
<script setup>
import { reactive } from 'vue'
const topics = reactive(['template syntax',
'declarative rendering', 'ref function’])
console.log(topics[0])
console.log(topics[1])
console.log(topics[2])
// topics.forEach((topic) => console.log(topic))
</script>
<template>
  <div>{{ topics[0] }}</div>
  <div>{{ topics[1] }}</div>
  <div>{{ topics[2] }}</div>
</template>
<style></style>
```

Reactive() to Ref() Function

```
<script setup>
import { ref, reactive, toRef, toRefs } from 'vue'
const cube = reactive({
  length: 10,
  width: 20,
  height: 33
})
const length = toRef(cube, 'length')
console.log(length.value)

console.log(cube.width)
const { width, height } = toRefs(cube)
console.log(width.value)
console.log(height.value)
</script>

<template>
  <p>{{ length }}</p>
  <p>{{ width }}</p>
  <p>{{ height }}</p>
</template>
```

Template Syntax : `v-on`

- We can use the `v-on` directive, which we typically shorten to the `@` symbol, to listen to DOM events and run some JavaScript when they're triggered.
- The usage would be `v-on:click ="methodName"` or with the shortcut, `@click="methodName"`

```
<script setup>
import { ref } from 'vue’
const myCourses = ref([
  'Statistics for Information Technology',
  'Applied Mathematic for data science',
  'Programming Fundamental'
])

const newCourse = ref('')
const addCourse = () =>
myCourses.value.push(newCourse.value)
</script>

<template>
<input type="text" v-model="newCourse" />
<button @click="addCourse" style="cursor:
pointer">Add Course</button>
</template>

<style></style>
```

Event Modifiers for `v-on`

```
<!-- the submit event will no longer reload the page -->
 <form @submit.prevent="onSubmit"></form>
```

```
<!-- key modifier using keyAlias -->
<input @keyup.enter="onEnter" />
```

`computed ()` Function

- Sometimes we need state that depends on other state - in Vue this is handled with component computed properties.
- When you want to include this calculation in your template more than once or the template is no longer simple and declarative, you should use a **computed property** on your reactive data.
- A computed property will only re-evaluate when some of its reactive dependencies have changed.
- This means as long as your reactive data has not changed, computed property will immediately return the previously computed result without having to run the function again.

computed () Function

```
<script setup>
import { ref, computed } from 'vue'
const msg = ref('')

const isEmpty = computed(() => {
  return msg.value.trim().length === 0 ? true : false
})
</script>
<template>
  <input type="text" v-model="msg"/>
   <p :class="isEmpty ? 'text-danger' : 'text-success'">
    {{ msg }}
  </p>
  <p v-if="isEmpty" class="text-warning bg-dark">please input your data!</p>
</template>
```

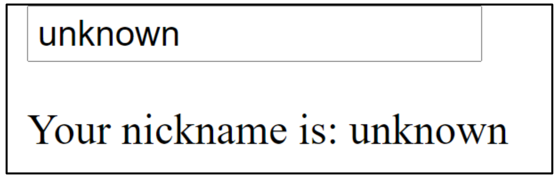
Form Input Bindings

Template Syntax : `v-model`

- Create a two-way binding on form `<input>`, `<textarea>`, and `<select>` elements.
- It automatically picks the correct way to update the element based on the input type.

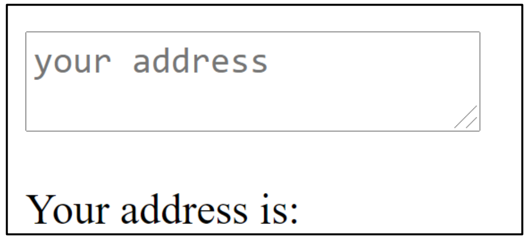
```
<script setup>
import { ref } from 'vue’
const yourName = ref('')

</script>

<template>
   <input type="text" v-model="yourName" />
  <h2>{{ yourName }}</h2>
</template>

<style></style>
```

Text


unknown

Your nickname is: unknown

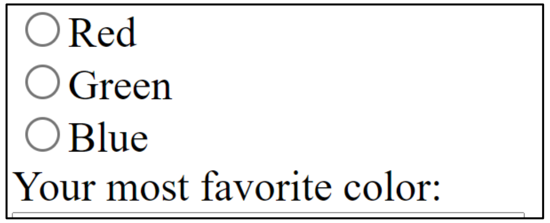
```
<input v-model= "nickname" placeholder= "your nickname" />
<p>Your nickname is: {{ nickname }}</p>
```


your address

Your address is:

```
<textarea v-model= "address" placeholder="your address"></textarea>
<p>Your address is: {{ address }}</p>
```

Radio

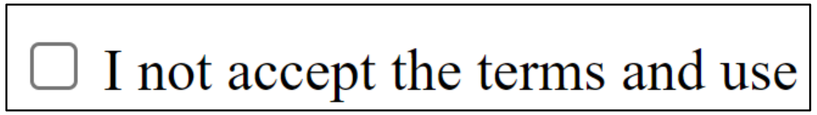


Your most favorite color:

```
<div>
<!-- favorColor is a string "Red" when the first option is checked -->
    <input type="radio" id= "red" value= "Red" v-model= "favorColor" />
    <label for= "red">Red</label> <br />
    <input type="radio" id= "green" value= "Green" v-model= "favorColor" />
    <label for= "green">Green</label> <br />
    <input type="radio" id= "blue" value= "Blue" v-model= "favorColor" />
    <label for= "blue">Blue</label> <br />
    <span>Your most favorite color: {{ favorColor }}</span>
</div>
```

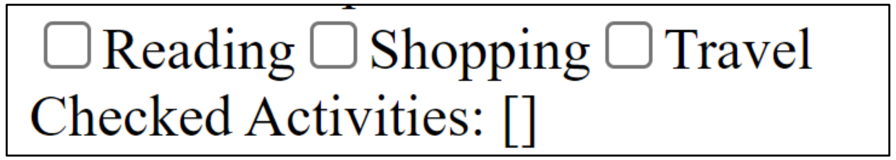
Checkbox

- Single checkbox, boolean value:


☐ I not accept the terms and use

```
<!-- agree is either true or false -->
<input type="checkbox" id= "agree" v-model= "agree" />
<label for="checkbox"> I {{ agree }} the terms and use </label>
```

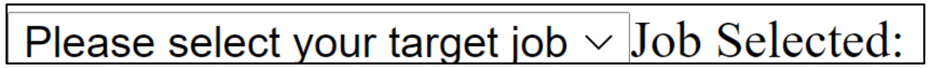
- Multiple checkboxes, bound to the same array:


☐ Reading ☐ Shopping ☐ Travel
Checked Activities: []

```
<div>
   <input type="checkbox" id= "reading" value= "Reading" v-model="checkedActivities" />
   <label for="reading">Reading</label>
   <input type="checkbox" id= "shopping" value= "Shopping" v-model=" checkedActivities " />
   <label for="shopping">Shopping</label>
   <input type="checkbox" id= "travel" value= "Travel" v-model=" checkedActivities " />
   <label for="travel">Travel</label>
   <br />
   <span>Checked Activities: {{ checkedActivities  }}</span>
</div>
```

Select

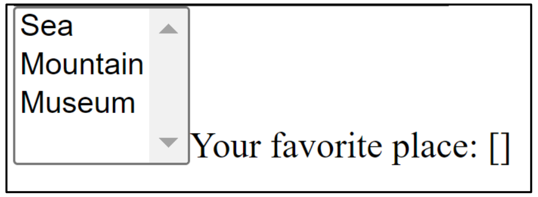
- Single Select:


Please select your target job ⌵ Job Selected:

```
<div >
<select v-model="jobSelected">
<! - - jobSelected  is a string " frontend" when the first option is selected -->
      <option value="">Please select your target job</option>
     <option value="frontend">Frontend Developer</option>
     <option value="backend">Backend Developer</option>
     <option value="devops">Devops</option>
     <option value="network">Network Admin</option>
     <option value="database">Database Admin</option>
   </select>
   <span>Job Selected: {{ jobSelected }}</span>
</div>
```

Select

- Multiple Select (bound to array):



Your favorite place: []

```
<select v-model= "favorPlace" multiple>
      <option value= "sea" >Sea</option>
      <option value= "mountain" >Mountain</option>
      <option value= "museum" >Museum</option>
</select>
<span>Your favorite place: {{ favorPlace }}</span>
```