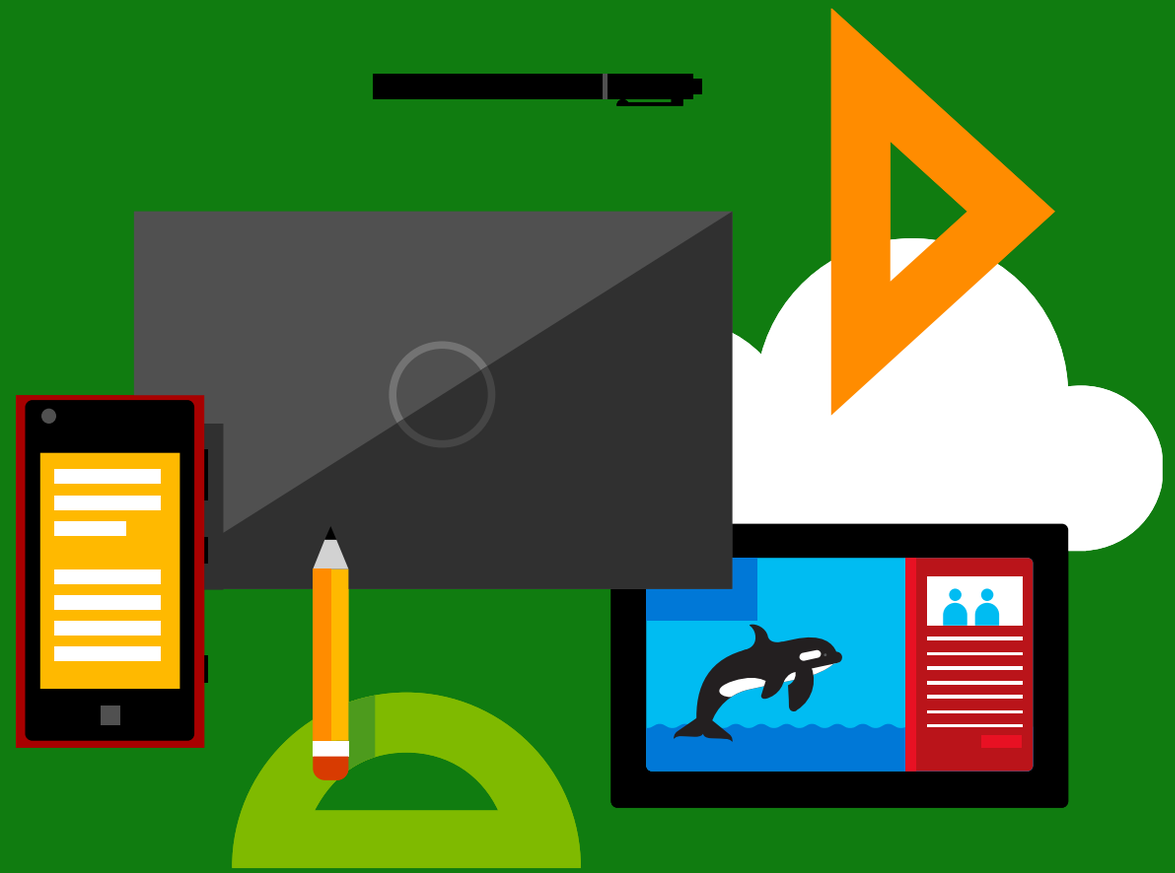
Desenvolvimento Web

HTML, CSS e Vue.js

Instrutor: Júlio Pereira Machado (julio.machado@pucrs.br)

Vue.js



Vue.js

- Framework para a construção de aplicativos web
- Código aberto

https://vuejs.org/

Instalação

- Criação de projetos via pacote *create-vue*
- Template básico com opções configuráveis
  - Instalar via commando `npm init vue@latest`

Vue Devtools

- Plugin para navegadores
- Permite depurar os components no ambiente do navegador

https://devtools.vuejs.org/

Aplicação

- Toda aplicação Vue é inicializada através da função *createApp()*
- Esta função recebe um *componente raiz* que será renderizado
- O ponto de renderização (ou montagem) do componente raiz é indicado pela função *mount()*

```
import './assets/main.css'

import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

Arquivo main.ts

SFC - Vue Single File Component

- Bloco de código que encapsula HTML, CSS, e Script (TS, no nosso caso)
  - <script>, <template>, <style>
- Arquivos *.vue
- Conceito: Declarative Rendering + Reactivity System
  - <template> para extender HTML, com vinculação (*binding*) baseado no estado do script
  - Estado se altera, HTML é atualizado
  - Estados ficam nos componentes, e mudança ocorre quando é considerada reativa
- IMPORTANTE:
  - Necessitam de um passo de *build* para pré-compilação do arquivo SFC
  - Para utilizar Vue.js sem o passo de *build* a configuração de componentes é diferente!

SFC - Vue Single File Component

```
<script setup lang="ts">

</script>

<template>

</template>

<style scoped>

</style>
```

APIs

- Dois estilos diferentes de APIs para programar componentes no Vue.js:
  - Options API
  - Composition API

APIs - Options

```
<script>
export default {
  data() {
    return {
      count: 0
    };
  },
  methods: {
    increment() {
      this.count++;
    }
  },
  mounted() {
    console.log(`The initial count is ${this.count}.`);
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

APIs - Composition

```
<script setup>
import { ref, onMounted } from 'vue';

const count = ref(0);

function increment() {
  count.value++;
}

onMounted(() => {
  console.log(`The initial count is ${count.value}.`);
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

TypeScript

- Vários elementos do Vue.js quando consumidos em TypeScript podem necessitar de tipagem explícita
- Ver:
  - https://vuejs.org/guide/typescript/composition-api.html
  - https://vuejs.org/guide/typescript/options-api.html

Templates

- Vue.js utiliza templates baseados em HTML
- Suporta diferentes sintaxes para processo de vinculação (*binding*) entre elementos do DOM e dados/ações do componente

Binding de propriedades

- Sintaxe “moustache” {{ }}
  - Apenas para interpolação de texto

```
<span>Message: {{ msg }}</span>
```

Binding de propriedades

- Binding de outros tipos: usar diretivas v-___
- Diretivas
  - Atributo especial que começa com v-
  - Parte da sintaxe de template
  - *v-html* para vincular a conteúdo contendo marcação HTML
  - v-bind para vincular atributos HTML a propriedades

Binding de propriedades

- Diretivas v-_____

```
<p>Using text interpolation: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

```
<div v-bind:id="dynamicId"></div>
<div :id="dynamicId"></div>
<div v-bind="objectOfAttrs"></div>
```

```
const objectOfAttrs = {
  id: 'container',
  class: 'wrapper'
}
```

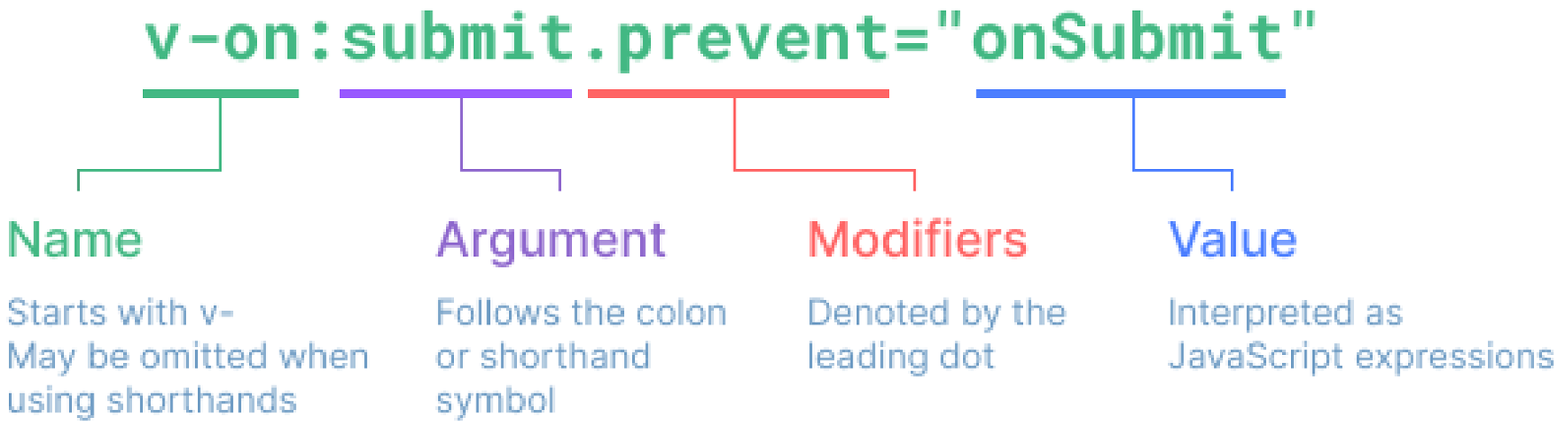
Binding de expressões

- Vue.js suporta a vinculação com expressões em JavaScript
- Cada vinculação pode ser realizada com somente uma expressão
- Cuidado:
  - Funções serão chamadas novamente a cada vez que o componente se atualiza, logo devem ser imutáveis!

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div :id="`list-${id}`"></div>

<time :title="toTitleDate(date)" :datetime="date">
  {{ formatDate(date) }}
</time>
```

Diretivas de binding

- Vue.js provê um conjunto de diretivas padrão
  - Ver https://vuejs.org/api/built-in-directives.html
- O papel central de uma diretiva é realizar atualizações reativas no DOM a partir de alterações na expressão vinculada



Tratadores de Eventos

- Eventos DOM são tratados com diretiva *v-on*

```
<a v-on:click="doSomething"> ... </a>

<a @click="doSomething"> ... </a>
```

Diretivas de binding - argumentos

- Argumentos passados a uma diretiva podem ser dinâmicos
- Sintaxe de nome da expressão dinâmica entre [ ]
  - Expressão deve ser avaliada em uma string

```
<a v-on:[eventName]="doSomething"> ... </a>

<a @[eventName]="doSomething">
```

Diretivas de binding - modificadores

- Modificadores são aplicados como sufixos às diretivas via “.”
- Indicam restrições especiais sobre à vinculação

```
<form @submit.prevent="onSubmit">...</form>
```

Indica que o tratador de evento irá executar automaticamente *event.preventDefault()*

Reatividade

- Para criar um estado reativo dentro de um componente utiliza-se a função *ref()* ou *reactive()*
- *ref()*
  - Argumento de entrada (qualquer valor) é encapsulado em um objeto de estado com a propriedade *value*
  - Estados reativos devem ser criados dentro do contexto de <script setup> para serem usados dentro de um <template>
  - Valores de um estado reativo são automaticamente acessados dentro de um <template> sem a necessida de acessar a propriedade *value* diretamente
  - Valores de um estado podem sofrer mutação
- *reactive()* – não é recomendado usar devido as suas limitações
  - Argumento é um objeto (não aceita tipos primitivos) que se torna um objeto de estado reativo, sem ser encapsulado em uma propriedade *value*
  - Vue.js cria um objeto *Proxy* com a funcionalidade reativa
  - CUIDADO: somente o objeto proxy é reativo! Não realize mutação no objeto original!

Reatividade

```
<script setup>
import { ref } from 'vue'

const count = ref(0)
console.log(count)
console.log(count.value)

function increment() {
  count.value++
}
</script>

<template>
  <button @click="increment">
    {{ count }}
  </button>
</template>
```

Reatividade

```
<script setup>
import { reactive } from 'vue’

interface Book {
  title: string
  year?: number
}

const book: Book = ref({ title: ‘Vue.js’ })
```

Reatividade

```
<script setup>
import { reactive } from 'vue'

const state = reactive({ count:0 })
console.log(state)

function increment() {
  state.count++
}
</script>

<template>
  <button @click="increment">
    {{ state.count }}
  </button>
</template>
```

Reatividade

- Como é realizada a atualização do DOM?
  - A atualização não é síncrona
  - Vue.js buferiza as atualizações até o próximo “tick” do ciclo de atualizações
  - Para esperar por uma atualização completa do DOM, usar *await nextTick()*

```
import { nextTick } from 'vue'

async function increment() {
  count.value++
  await nextTick()
  // Agora o DOM já foi atualizado
}
```

Reatividade

- Avaliar expressões com lógica de negócio complexa ao realizar a vinculação não é adequado para estados reativos
- Utilizar propriedades computadas através da função *computed()*
  - Valor retornado pela função "de leitura" (sem efeitos colaterais) é um *ref* associado ao resultado da função a ser computada quando vinculado no template
    - CUIDADI: não altere o valor ref computado retornado, altere somente os valores de suas dependências
  - Mantem a observação de todas as dependências reativas automaticamente
  - Realiza cache e somente dispara uma nova renderização caso uma das dependências sofra uma alteração
  - Em casos de necessidade suporta o uso de um objeto com getter e setter via métodos *get()* e *set(valor)*

Reatividade

```
<script setup>
import { ref, computed } from 'vue'
const author = ref({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})
const publishedBooksMessage = computed(() => {
  return author.value.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Has published books:

Yes

Renderização Condicional

- Diretiva *v-if*
  - Elemento somente será renderizado se o valor vinculado for avaliado como verdadeiro
  - Implica que o elemento será removido da árvore DOM

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

Renderização Condicional

- Diretiva *v-else*
  - Bloco “else” associado a uma diretiva *v-if*


```
<script setup>
import { ref } from 'vue'

const awesome = ref(true)
</script>

<template>
  <button @click="awesome = !awesome">toggle</button>
  <h1 v-if="awesome">Vue is awesome!</h1>
  <h1 v-else>Oh no 😢</h1>
</template>
```

Renderização Condicional

- Diretiva *v-else-if*
  - Bloco “else if” associado a uma diretiva *v-if*

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

Renderização Condicional

- As diretivas de renderização condicional somente podem ser aplicadas a um único elemento
- Para aplicar a condição a “vários elementos”, associar as diretivas ao <template>

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

Renderização Condicional

- Diretiva *v-show*
  - Elemento somente será renderizado se o valor vinculado for avaliado como verdadeiro
  - Efetua a troca do valor da propriedade *display* do CSS
  - Implica que o elemento sempre estará presente na árvore DOM

```
<h1 v-show="awesome">Vue is awesome!</h1>
```

Renderização de Listas

- Diretiva *v-for*
  - Renderiza uma lista de itens com base em um array
  - Sintaxe estilo "para cada" *item in array* ou *item of array*
  - Sintaxe opcional permite acessar o índice do elemento atual no array *(item,index) in array*
  - Podem ser aninhados

```
const items = ref([{ message: 'Foo' }, { message: 'Bar' }])

<li v-for="item in items">
  {{ item.message }}
</li>
```

Renderização de Listas

- Parent - 0 - Foo
- Parent - 1 - Bar

```
<script setup>
import { ref } from 'vue'
const parentMessage = ref('Parent')
const items = ref([{ message: 'Foo' }, { message: 'Bar' }])
</script>

<template>
    <li v-for="(item, index) in items">
    {{ parentMessage }} - {{ index }} - {{ item.message }}
    </li>
</template>
```

Renderização de Listas

- Diretiva *v-for*
  - Renderiza uma lista de itens com base nas propriedades de um objeto

```
const myObject = ref({
  title: 'How to do lists in Vue',
  author: 'Jane Doe',
  publishedAt: '2016-04-10'
})
```

```
<ul>
  <li v-for="value in myObject">
    {{ value }}
  </li>
</ul>
```

```
<li v-for="(value, key) in myObject">
  {{ key }}: {{ value }}
</li>
```

```
<li v-for="(value, key, index) in myObject">
  {{ index }}. {{ key }}: {{ value }}
</li>
```

Renderização de Listas

- Diretiva *v-for*
  - Suporta a parametrização de intervalor de valores
  - Sintaxe *n in limiteSuperior* resulta no intervalo de valores inteiros [1..n]

```
<span v-for="n in 10">{{ n }}</span>
```

Renderização de Listas

- Diretiva *v-for*
  - Para aplicar a condição a “vários elementos”, associar a diretiva ao <template>

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider" role="presentation"></li>
  </template>
</ul>
```

```
<template v-for="todo in todos">
  <li v-if="!todo.isComplete">
    {{ todo.name }}
  </li>
</template>
```

Renderização de Listas

- CUIDADO!
  - Para a manutenção reativa correta das mudanças dos elementos em uma lista renderizada com dependências complexas, o Vue.js utiliza uma chave única informada via atributo *key*
  - A chave deve ser um valor primitivo

```
<div v-for="item in items" :key="item.id">
  <!-- conteúdo -->
</div>
```

```
<template v-for="todo in todos" :key="todo.id">
  <li>{{ todo.id }}</li>
</template>
```

Renderização de Listas

- CUIDADO!
  - A detecção de alteração reativa no array se dá através dos métodos de mutação array: push(), pop(), shif(), unshift(), splice(), sort(), reverse(), etc
  - Métodos que retornam um novo array alterado necessitam substituir por completo o array associado à referência reativa

```
// `items` é um ref com value de array
items.value = items.value.filter((item) => item.message.match(/Foo/))
```

Renderização de Listas

- CUIDADO!
  - Alternativamente usar uma propriedade computadas ou uma função para casos de filtragem/ordenação

```
const numbers = ref([1, 2, 3, 4, 5])
const evenNumbers = computed(() => {
  return numbers.value.filter((n) => n % 2 === 0)
})
<li v-for="n in evenNumbers">{{ n }}</li>
```

```
const sets = ref([
  [1, 2, 3, 4, 5],
  [6, 7, 8, 9, 10]
])
function even(numbers) {
  return numbers.filter((number) => number % 2 === 0)
}
<ul v-for="numbers in sets">
  <li v-for="n in even(numbers)">{{ n }}</li>
</ul>
```

Two-way Binding

- A vinculação de mão-dupla pode ser implementada utilizando-se as diretivas *v-bind* e *v-on* em conjunto

```
<script setup>
import { ref } from 'vue'

const text = ref('')

function onInput(e) {
  text.value = e.target.value
}
</script>

<template>
  <input :value="text" @input="onInput" placeholder="Type here">
  <p>{{ text }}</p>
</template>
```

Two-way Binding

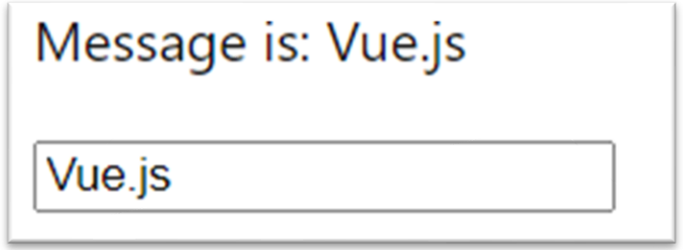
- Para simplificar a vinculação de mão-dupla utiliza-se a diretiva *v-model*

```
<script setup>
import { ref } from 'vue'

const text = ref('')

</script>

<template>
  <input v-model="text" placeholder="Type here">
  <p>{{ text }}</p>
</template>
```

Two-way Binding

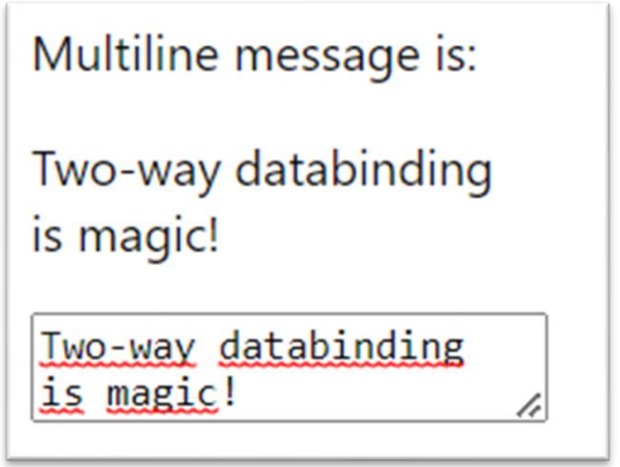
- A diretiva *v-model* pode ser utilizada com elementos de entrada de textos, checkboxes, botões de rádio, dropdowns, etc.
  - <input> e <textarea> vincula ao atributo *value* e evento *input*
  - <input type="checkbox"> e <input type="radio"> vincula ao atributo *checked* e evento *change*
  - <select> vincula ao atributo *selected* e evento *change*
- CUIDADO:
  - *v-model* ignora valor inicial dos atributos *value*, *checked* ou *selected* encontrados no HTML e utiliza o valor reativo configurado em código JavaScript

Two-way Binding




```
<script setup>
import { ref } from 'vue'

const message = ref('')
</script>

<template>
  <p>Message is: {{ message }}</p>
  <input v-model="message" placeholder="edit me" />
</template>
```

Two-way Binding




```
<script setup>
import { ref } from 'vue'

const message = ref('')
</script>

<template>
    <span>Multiline message is:</span>
    <p style="white-space: pre-line;">{{ message }}</p>
    <textarea v-model="message" placeholder="add multiple lines"></textarea>
</template>
```

Two-way Binding


```
<script setup>
import { ref } from 'vue'

const checked = ref(true)
</script>

<template>
    <input type="checkbox" id="checkbox" v-model="checked" />
    <label for="checkbox">{{ checked }}</label>
</template>
```

Two-way Binding

```
<input
  type="checkbox"
  v-model="toggle"
  true-value="yes"
  false-value="no"
/>
```

Valores não-booleanos

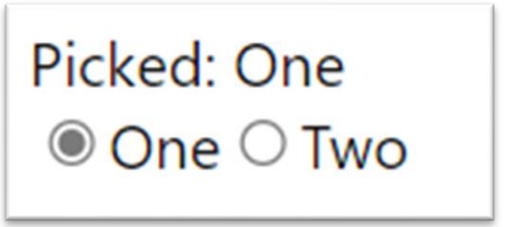
```
<input
  type="checkbox"
  v-model="toggle"
  :true-value="dynamicTrueValue"
  :false-value="dynamicFalseValue"
/>
```

Valores dinâmicos

Two-way Binding


```
<script setup>
import { ref } from 'vue'

const checkedNames = ref([])
</script>

<template>
  <div>Checked names: {{ checkedNames }}</div>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames" />
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames" />
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames" />
  <label for="mike">Mike</label>
</template>
```

Two-way Binding



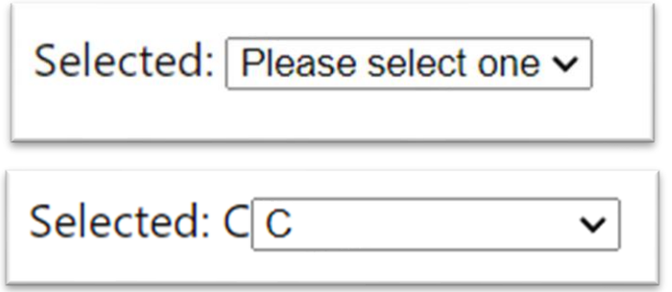

```
<script setup>
import { ref } from 'vue'

const picked = ref('One')
</script>

<template>
  <div>Picked: {{ picked }}</div>

    <input type="radio" id="one" value="One" v-model="picked" />
    <label for="one">One</label>

    <input type="radio" id="two" value="Two" v-model="picked" />
    <label for="two">Two</label>
</template>
```

Two-way Binding

```
<input type="radio" v-model="pick" :value="first" />
<input type="radio" v-model="pick" :value="second" />
```

Valores dinâmicos

Two-way Binding



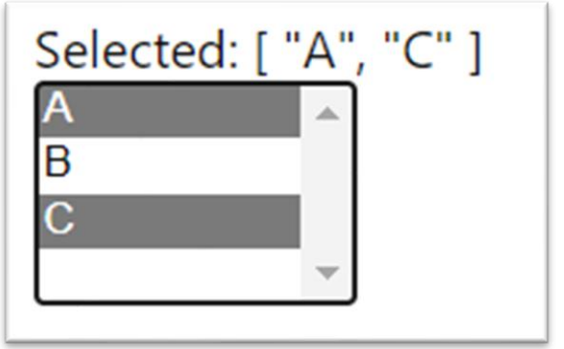

```
<script setup>
import { ref } from 'vue'

const selected = ref('')
</script>

<template>
  <span> Selected: {{ selected }}</span>

  <select v-model="selected">
    <option disabled value="">Please select one</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
</template>
```

Two-way Binding

```
<select v-model="selected">
  <option :value="{ number: 123 }">123</option>
</select>
```

Valores não-string

Two-way Binding




```
<script setup>
import { ref } from 'vue'
const selected = ref([])
</script>

<template>
  <div>Selected: {{ selected }}</div>
  <select v-model="selected" multiple>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
</template>
```

Two-way Binding


```
<script setup>
import { ref } from 'vue'
const selected = ref('A')
const options = ref([
  { text: 'One', value: 'A' },
  { text: 'Two', value: 'B' },
  { text: 'Three', value: 'C' }
])
</script>

<template>
  <select v-model="selected">
    <option v-for="option in options" :value="option.value">
      {{ option.text }}
    </option>
  </select>
  <div>Selected: {{ selected }}</div>
</template>
```

Two-way Binding

- Modificadores para elementos input:
  - .*lazy* permite trocar o evento de *input* para *change*
  - .*number* permite realizar a conversão para valor numérico automaticamente, caso a conversão não seja possível, utiliza o valor normal; é aplicado automaticamente para <input type="number">
  - .*trim* realiza a remoção de espaços em branco (aplicação da função trim())

```
<input v-model.lazy="msg" />
```

```
<input v-model.number="age" />
```

```
<input v-model.trim="msg" />
```