

# Vue.js Quick Reference

Templates, reactivity, components, composition API, router

## Template Syntax

### Text & Expressions

```
<span>{{ message }}</span>
<span>{{ count + 1 }}</span>
<span>{{ ok ? 'Yes' : 'No' }}</span>
<span v-html="rawHtml"></span>
```

### Directives

<b>{{ expr }}</b>	Text interpolation
<b>v-bind:attr / :attr</b>	Bind attribute to expression
<b>v-on:event / @event</b>	Attach event listener
<b>v-model</b>	Two-way binding (forms)
<b>v-if / v-else-if / v-else</b>	Conditional rendering
<b>v-show</b>	Toggle display CSS (stays in DOM)
<b>v-for</b>	List rendering
<b>v-slot / #name</b>	Named slot content

### Attribute Binding

```

<div :class="{ active: isActive }"></div>
<div :style="{ color: textColor }"></div>
<button :disabled="isLoading">Submit</button>
```

## Reactivity

### ref (Primitives)

```
import { ref } from 'vue'

const count = ref(0)
console.log(count.value) // 0
count.value++           // reactive update
```

### reactive (Objects)

```
import { reactive } from 'vue'

const state = reactive({ count: 0, name: 'Vue' })
state.count++ // no .value needed
```

### ref vs reactive

<b>ref()</b>	Any type; access via <b>.value</b> in script
<b>reactive()</b>	Objects/arrays only; direct property access
<b>Template</b>	Both auto-unwrap (no <b>.value</b> needed)
<b>Destructure</b>	<b>reactive</b> loses reactivity; use <b>toRefs()</b>

## Computed & Watchers

### Computed Properties

```
import { ref, computed } from 'vue'

const items = ref([1, 2, 3, 4, 5])
const evenItems = computed(() =>
  items.value.filter(n => n % 2 === 0)
)
```

Computed values are cached and only re-evaluate when dependencies change

## Watchers

```
import { ref, watch, watchEffect } from 'vue'

const query = ref('')

// Watch specific source
watch(query, (newVal, oldVal) => {
  console.log(`Changed: ${oldVal} → ${newVal}`)
})

// Auto-track dependencies
watchEffect(() => {
  console.log(`Query is: ${query.value}`)
})
```

### Watch Options

<b>immediate: true</b>	Run callback immediately on creation
<b>deep: true</b>	Deep watch nested objects
<b>flush: 'post'</b>	Run after DOM update
<b>once: true</b>	Trigger only once then stop

## Components

### Single-File Component (SFC)

```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

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

<style scoped>
button { font-size: 1.2em; }
</style>
```

### Registering Components

```
<!-- Auto-imported with <script setup> -->
<script setup>
import MyButton from './MyButton.vue'
</script>

<template>
  <MyButton label="Click me" />
</template>
```

### SFC Blocks

<b>&lt;script setup&gt;</b>	Composition API (recommended)
<b>&lt;template&gt;</b>	HTML template
<b>&lt;style scoped&gt;</b>	Component-scoped CSS
<b>&lt;style module&gt;</b>	CSS Modules (\$style object)

## Props & Events

### Defining Props

```
<script setup>
const props = defineProps({
  title: String,
  count: { type: Number, default: 0 },
  items: { type: Array, required: true }
})
</script>
```

## Emitting Events

```
<script setup>
const emit = defineEmits(['update', 'delete'])

function handleClick() {
  emit('update', { id: 1, value: 'new' })
}
</script>
```

### Parent Usage

```
<ChildComponent
  :title="pageTitle"
  :count="total"
  @update="handleUpdate"
  @delete="handleDelete"
/>
```

### v-model on Components

```
<!-- Parent -->
<CustomInput v-model="search" />

<!-- CustomInput.vue -->
<script setup>
const model = defineModel()
</script>
<template>
  <input :value="model" @input="model = $event.target.value" />
</template>
```

## Slots

### Default Slot

```
<!-- Card.vue -->
<template>
  <div class="card">
    <slot>Fallback content</slot>
  </div>
</template>

<!-- Usage -->
<Card><p>Custom content here</p></Card>
```

### Named Slots

```
<!-- Layout.vue -->
<template>
  <header><slot name="header" /></header>
  <main><slot /></main>
  <footer><slot name="footer" /></footer>
</template>

<!-- Usage -->
<Layout>
  <template #header><h1>Title</h1></template>
  <p>Main content</p>
  <template #footer><span>Footer</span></template>
</Layout>
```

### Scoped Slots

```
<!-- List.vue -->
<ul>
  <li v-for="item in items" :key="item.id">
    <slot :item="item" />
  </li>
</ul>

<!-- Usage -->
<List :items="todos">
  <template #default="{ item }">
    <span>{{ item.text }}</span>
  </template>
</List>
```

# Vue.js Quick Reference

## Composition API

### Composable Function

```
// useMouse.js
import { ref, onMounted, onUnmounted } from 'vue'

export function useMouse() {
  const x = ref(0)
  const y = ref(0)
  function update(e) {
    x.value = e.pageX
    y.value = e.pageY
  }
  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove', update))
  return { x, y }
}
```

### Using Composables

```
<script setup>
import { useMouse } from './useMouse'

const { x, y } = useMouse()
</script>
<template>
  <p>Mouse: {{ x }}, {{ y }}</p>
</template>
```

### provide / inject

```
// Parent
import { provide, ref } from 'vue'
const theme = ref('dark')
provide('theme', theme)

// Descendant (any depth)
import { inject } from 'vue'
const theme = inject('theme', 'light') // default
```

## Router (Vue Router)

### Route Definitions

```
import { createRouter, createWebHistory } from 'vue-router'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
    { path: '/about', component: About },
    { path: '/user/:id', component: User },
  ]
})
```

### Template Navigation

```
<router-link to="/">Home</router-link>
<router-link :to="{ name: 'user', params: { id: 1 } }">
  User 1
</router-link>
<router-view />
```

### Programmatic Navigation

```
import { useRouter, useRoute } from 'vue-router'

const router = useRouter()
const route = useRoute()

router.push('/about')
router.push({ name: 'user', params: { id: 1 } })
console.log(route.params.id)
```

## Route Features

<b>/user/:id</b>	Dynamic segment ( <b>route.params.id</b> )
<b>name: 'user'</b>	Named route for programmatic nav
<b>children: [...]</b>	Nested routes
<b>beforeEnter</b>	Per-route navigation guard
<b>meta: { auth: true }</b>	Custom metadata for guards
<b>redirect: '/new-path'</b>	Route redirect

## Lifecycle Hooks

### Hook Order

<b>onBeforeMount</b>	Before initial DOM render
<b>onMounted</b>	DOM is ready (fetch data, add listeners)
<b>onBeforeUpdate</b>	Before reactive state re-renders DOM
<b>onUpdated</b>	After DOM re-render
<b>onBeforeUnmount</b>	Before component is destroyed
<b>onUnmounted</b>	Cleanup (remove listeners, timers)

## Usage

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

onMounted(() => {
  console.log('Component mounted')
})

onUnmounted(() => {
  console.log('Cleanup here')
})
</script>
```

## Lists & Conditionals

### v-for

```
<li v-for="item in items" :key="item.id">
  {{ item.name }}
</li>
<li v-for="(item, index) in items" :key="item.id">
  {{ index }}: {{ item.name }}
</li>
<div v-for="(val, key) in obj" :key="key">
  {{ key }}: {{ val }}
</div>
```

Always use :key with v-for for efficient DOM updates

### v-if vs v-show

<b>v-if</b>	Conditionally render (add/remove from DOM)
<b>v-else-if</b>	Else-if chain
<b>v-else</b>	Fallback branch
<b>v-show</b>	Toggle <b>display: none</b> (stays in DOM)

Use v-show for frequent toggles, v-if for rare changes

### Conditionals Example

```
<div v-if="status === 'loading'">Loading...</div>
<div v-else-if="status === 'error'">Error!</div>
<div v-else>{{ data }}</div>
```

## Form Handling

### v-model Basics

```
<input v-model="text" />
<textarea v-model="message"></textarea>
<input type="checkbox" v-model="checked" />
<select v-model="selected">
  <option value="a">A</option>
  <option value="b">B</option>
</select>
```

## v-model Modifiers

<b>v-model.lazy</b>	Sync on <b>change</b> instead of <b>input</b>
<b>v-model.number</b>	Auto-typecast to number
<b>v-model.trim</b>	Auto-trim whitespace

## Event Modifiers

<b>@click.prevent</b>	Call <b>preventDefault()</b>
<b>@click.stop</b>	Call <b>stopPropagation()</b>
<b>@click.once</b>	Trigger at most once
<b>@keyup.enter</b>	Only on Enter key
<b>@submit.prevent</b>	Prevent form submission default