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

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

Attribute Binding

```
<img :src="imageUrl" :alt="desc" />
<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

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

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

`immediate: true`	Run callback immediately on creation
`deep: true`	Deep watch nested objects
`flush: 'post'`	Run after DOM update
`once: true`	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

`<script setup>`	Composition API (recommended)
`<template>`	HTML template
`<style scoped>`	Component-scoped CSS
`<style module>`	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>
```

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

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

Lifecycle Hooks

Hook Order

`onBeforeMount`	Before initial DOM render
`onMounted`	DOM is ready (fetch data, add listeners)
`onBeforeUpdate`	Before reactive state re-renders DOM
`onUpdated`	After DOM re-render
`onBeforeUnmount`	Before component is destroyed
`onUnmounted`	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

`v-if`	Conditionally render (add/remove from DOM)
`v-else-if`	Else-if chain
`v-else`	Fallback branch
`v-show`	Toggle `display: none` (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

`v-model.lazy`	Sync on `change` instead of `input`
`v-model.number`	Auto-typecast to number
`v-model.trim`	Auto-trim whitespace

Event Modifiers

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