Vue.js

Single-file components, reactivity and templates.

01 Create an app

```
npm create vue@latest my-app
cd my-app && npm install && npm run dev
```

```
import { createApp } from 'vue'
import App from './App.vue'
createApp(App).mount('#app')
```

Prompts can add TypeScript, JSX, Router, Pinia, Vitest and ESLint. A CDN script needs no build but cannot use .vue files.

02 Single-file components

```
import { ref } from 'vue'
const count = ref(0)
const increment = () => count.value++
```

```
<template>
  <button @click="increment">+1</button>
  <p>Count: {{ count }}</p>
</template>
```

A .vue file: <script setup> (lang="ts" optional), <template>, optional <style scoped>. Top-level bindings reach the template without a return.

03 ref & reactive

```
const count = ref(0)
count.value++              // in script
const user = ref({ name: 'John' })
user.value.name = 'Jane'   // refs are deep
const state = reactive({ items: [] })
state.items.push(1)        // no .value
```

Prefer ref(): reactive() takes objects only and breaks when reassigned or destructured. Templates unwrap top-level refs.

04 computed

```
const total = computed(() =>
  price.value * qty.value)
const fullName = computed({
  get: () => first.value + ' ' + last.value,
  set(v) {
    [first.value, last.value] = v.split(' ')
  },
})
```

Cached until a dependency changes, unlike a method. Keep getters free of side effects.

05 watch & watchEffect

```
watch(count, (now, prev) =>
  console.log(`${prev} -> ${now}`))
watch(() => user.age, onAgeChange)
watch(src, cb, { immediate: true })
watchEffect(() => console.log(count.value))
```

Watch a reactive property via a getter. Options: immediate, deep, flush. watchEffect runs at once.

06 v-if, v-show & v-for

```
<p v-if="ok">Saved</p><p v-else>Draft</p>
<p v-show="open">Hidden with CSS</p>
<li v-for="(u, i) in users" :key="u.id">
  {{ i }}. {{ u.name }}
</li>
<template v-for="t in todos" :key="t.id">
  <li v-if="!t.done">{{ t.text }}</li>
</template>
```

v-if outranks v-for on one element, so loop on a <template> and key the template, not its children.

07 Events & modifiers

```
<button @click="count++">Add</button>
<a @click="say('hi', $event)">Hi</a>
<form @submit.prevent="save"></form>
<a @click.stop.prevent="open">Open</a>
<input @keyup.enter="search" />
<input @keyup.ctrl.enter="send" />
```

Also .self, .capture, .once, .passive. Keys: .tab, .delete, .esc, .space, arrows; .exact needs an exact combo.

08 Attribute, class & style

```
<img :src="photo" :alt="caption" />
<img :src :alt />  <!-- 3.4+ shorthand -->
<div class="card" :class="{ active: isOn }"
  :style="{ fontSize: size + 'px' }"></div>
<p :class="[base, { error: hasError }]"></p>
<Card v-bind="cardProps" :[attr]="value" />
```

: is v-bind. Class and style merge with static ones. null removes an attribute; false only removes boolean ones.

09 Forms & v-model

```
<input v-model.trim="name" />
<input v-model.number="age" />
<input type="checkbox" v-model="agree" />
<input type="checkbox" v-model="tags"
  value="vue" />
<select v-model="country">
  <option disabled value="">Pick</option>
  <option value="ca">Canada</option>
</select>
```

Several checkboxes bind an array. .lazy syncs on change. File inputs have no v-model: use @change.

Vue.js

Components, routing, state and transitions.

10 Props

```
const props = defineProps({
  title: String,
  count: { type: Number, required: true },
  tags: { type: Array, default: () => [] },
})
const { msg = 'Hi' } = defineProps<Props>()
```

Read-only in the child. Last line: the lang="ts" form, defaults by destructure (3.5+); before 3.5, withDefaults().

11 Emits & v-model

```
const emit = defineEmits(['remove'])
emit('remove', 42)
const model = defineModel<string>()
const title = defineModel('title')
```

```
<Item @remove="drop" v-model="text"
  v-model:title="heading" />
```

Component events do not bubble. defineModel (3.4+) syncs a ref with the modelValue prop and update:modelValue.

12 Slots

```
<slot name="header" />  <!-- in Card -->
<slot :item="item">No item</slot>
<Card>                  <!-- parent -->
  <template #header>Title</template>
  <template #default="{ item }">
    {{ item.name }}
  </template>
</Card>
```

is short for v-slot. A scoped slot hands child data (item) to the parent template.

13 Provide & inject

```
export const userKey =     // keys.ts
  Symbol() as InjectionKey<Ref<User>>
provide(userKey, user)     // ancestor
const me = inject(userKey) // descendant
const theme = inject('theme', 'light')
```

app.provide() shares app-wide. inject(key, () => make(), true) builds the default lazily. Keep mutations in the provider.

14 Lifecycle & template refs

```
const input = useTemplateRef('search')
onMounted(() => input.value?.focus())
const id = setInterval(tick, 1000)
onUnmounted(() => clearInterval(id))
```

useTemplateRef (3.5+) reads <input ref="search" />. Hooks: onBeforeMount, onMounted, onBeforeUpdate, onUpdated, onBeforeUnmount, onUnmounted, onErrorCaptured. Composables (use* functions) call them too.

15 Transition & Teleport

```
<Transition name="fade" mode="out-in">
  <p v-if="show">Hello</p>
</Transition>
<Teleport to="body"><Modal /></Teleport>
```

fade-enter-from	Start state, gone next frame
fade-enter-active	Whole enter: set transition
fade-enter-to	End state of the enter
fade-leave-*	Same three when leaving

Transition takes one child. Suspense (experimental) shows #fallback while async setup runs.

16 Vue Router

```
const routes = [
  { path: '/user/:id', name: 'user',
    component: () => import('./User.vue') },
]
const router = createRouter({
  history: createWebHistory(), routes })
```

app.use(router) registers it; import() lazy-loads.

17 Navigation & guards

```
<RouterLink to="/about">About</RouterLink>
<RouterView />
```

```
const route = useRoute()  // route.params.id
router.beforeEach((to) => {
  if (to.meta.requiresAuth && !isAuthed())
    return '/login'
})
```

Return false to cancel or a path to redirect; next() is deprecated in Router 5. useRouter().push() navigates.

18 Pinia

```
export const useCounterStore = defineStore(
  'counter', {
  state: () => ({ count: 0 }),
  getters: { double: (s) => s.count * 2 },
  actions: { increment() { this.count++ } },
})
const store = useCounterStore()
const { count, double } = storeToRefs(store)
```

Setup stores take a function: refs are state, computeds getters, functions actions. storeToRefs keeps state reactive.