VUE.JS 빠른 참조

템플릿, 반응성, 컴포넌트, Composition API, 라우터

템플릿 구문

텍스트 및 표현식

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

디렉티브

`{{ expr }}`	텍스트 보간
`v-bind:attr / :attr`	속성을 표현식에 바인딩
`v-on:event / @event`	이벤트 리스너 연결
`v-model`	양방향 바인딩 (폼)
`v-if / v-else-if / v-else`	조건부 렌더링
`v-show`	display CSS 전환 (DOM에 유지)
`v-for`	리스트 렌더링
`v-slot / #name`	명명된 슬롯 콘텐츠

속성 바인딩

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

반응성

ref (원시값)

```
import { ref } from 'vue'

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

reactive (객체)

```
import { reactive } from 'vue'

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

ref vs reactive

`ref()`	모든 타입; 스크립트에서 `.value`로 접근
`reactive()`	객체/배열만; 직접 속성 접근
`Template`	둘 다 자동 언래핑 (`.value` 불필요)
`Destructure`	`reactive`는 분해 시 반응성 잃음; `toRefs()` 사용

계산된 속성 및 감시자

계산된 속성

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

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

계산된 값은 캐시되어 의존성이 변경될 때만 재평가됩니다

감시자

```
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 옵션

`immediate: true`	생성 즉시 콜백 실행
`deep: true`	중첩 객체 깊은 감시
`flush: 'post'`	DOM 업데이트 후 실행
`once: true`	한 번만 트리거 후 중지

컴포넌트

단일 파일 컴포넌트 (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>
```

컴포넌트 등록

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

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

SFC 블록

`<script setup>`	Composition API (권장)
`<template>`	HTML 템플릿
`<style scoped>`	컴포넌트 범위 CSS
`<style module>`	CSS 모듈 ($style 객체)

프롭 및 이벤트

프롭 정의

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

이벤트 emit

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

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

부모에서 사용

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

컴포넌트의 v-model

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

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

슬롯

기본 슬롯

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

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

명명된 슬롯

```
<!-- 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>
```

범위가 있는 슬롯

```
<!-- 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

컴포저블 함수

```
// 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 }
}
```

컴포저블 사용

```
<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
```

라우터 (Vue Router)

라우트 정의

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

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

템플릿 내비게이션

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

프로그래밍 방식 내비게이션

```
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)
```

라우트 기능

`/user/:id`	동적 세그먼트 (`route.params.id`)
`name: 'user'`	프로그래밍 내비게이션용 명명된 라우트
`children: [...]`	중첩 라우트
`beforeEnter`	라우트별 내비게이션 가드
`meta: { auth: true }`	가드용 커스텀 메타데이터
`redirect: '/new-path'`	라우트 리다이렉트

라이프사이클 훅

훅 순서

`onBeforeMount`	초기 DOM 렌더링 전
`onMounted`	DOM 준비 완료 (데이터 가져오기, 리스너 추가)
`onBeforeUpdate`	반응성 상태가 DOM을 재렌더링하기 전
`onUpdated`	DOM 재렌더링 후
`onBeforeUnmount`	컴포넌트 소멸 전
`onUnmounted`	정리 (리스너, 타이머 제거)

사용 방법

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

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

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

리스트 및 조건

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>
```

효율적인 DOM 업데이트를 위해 v-for에서 항상 :key 사용

v-if vs v-show

`v-if`	조건부 렌더링 (DOM에 추가/제거)
`v-else-if`	else-if 체인
`v-else`	폴백 브랜치
`v-show`	`display: none` 전환 (DOM에 유지)

자주 전환 시 v-show, 드물게 변경 시 v-if 사용

조건 예제

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

폼 처리

v-model 기본

```
<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 수정자

`v-model.lazy`	`input` 대신 `change`에서 동기화
`v-model.number`	자동으로 숫자로 타입 캐스팅
`v-model.trim`	자동으로 공백 트리밍

이벤트 수정자

`@click.prevent`	`preventDefault()` 호출
`@click.stop`	`stopPropagation()` 호출
`@click.once`	최대 한 번만 트리거
`@keyup.enter`	Enter 키만
`@submit.prevent`	폼 제출 기본 동작 방지