Vue 3 Composition API Ultimate Cheatsheet

Setup Function

- **Purpose**: Entry point for Composition API logic
- **Replaces**: `data()`, `methods`, `computed`, and lifecycle hooks
- **Auto-Exposure**: Returned values are template-accessible

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

const count = ref(0);
onMounted(() => console.log('Mounted!'));
</script>
```

Reactivity Fundamentals

ref()

- **Use Case**: Primitive values or object references
- **Note**: Requires `.value` in JS, auto-unwraps in templates

```
const counter = ref(0);
counter.value = 5; // Update value
```

reactive()

- **Use Case**: Complex objects/collections
- **Warning**: Avoid direct destructuring (use `toRefs`)

```
const state = reactive({
  user: { name: 'John', age: 30 },
  items: [],
```

```
});
state.user.age = 31;
```

computed()

- **Best For**: Derived values with caching
- **Performance**: Only re-calculates when dependencies change

```
const fullName = computed(() => `${firstName.value} ${lastName.value}`);
```

Watch System

watch()

- **Use When**: Need explicit control over watched sources
- **Deep Watch**: Add `{ deep: true }` option

```
watch(
  [user, posts],
  ([newUser, newPosts], [oldUser, oldPosts]) => {
    // Handle changes
  },
  { immediate: true }
);
```

watchEffect()

- **Use When**: Immediate reactive dependency tracking
- **Cleanup**: Automatic on unmount

```
const stop = watchEffect(() => {
  console.log('Window width:', window.innerWidth);
});
// Manually stop
stop();
```

watchPostEffect/watchSyncEffect

- **Advanced Timing**: Control effect flush timing

```
watchPostEffect(() => {
  // Runs after DOM updates
});
```

Lifecycle Hooks

- **Usage**: Import and use directly in setup
- **Equivalents**:
  - `onBeforeMount` → `beforeMount`
  - `onMounted` → `mounted`
  - `onBeforeUpdate` → `beforeUpdate`
  - `onUpdated` → `updated`
  - `onBeforeUnmount` → `beforeDestroy`
  - `onUnmounted` → `destroyed`

```
import { onMounted, onUnmounted } from 'vue';

onMounted(() => {
  window.addEventListener('resize', handleResize);
});

onUnmounted(() => {
  window.removeEventListener('resize', handleResize);
});
```

Composables

- **Pattern**: Reusable stateful logic
- **Convention**: Name starting with `use*`
- **Best Practice**: Return reactive references

```
// 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 };
}

// Component usage
const { x, y } = useMouse();
```

State Management

Pinia (Recommended)

- **Features**: Type-safe, DevTools support, modular

```
// stores/counter.js
export const useCounterStore = defineStore('counter', {
  state: () => ({ count: 0 }),
  getters: {
    double: (state) => state.count * 2,
  },
  actions: {
    increment() {
      this.count++;
    },
  },
});

// Component usage
const store = useCounterStore();
store.increment();
```

Vuex 4

- **Legacy Support**: For existing projects

```
import { useStore } from 'vuex';
const store = useStore();
store.commit('increment');
```

Component Communication

Props

```
const props = defineProps({
  title: {
    type: String,
    required: true,
    validator: (v) => v.length > 3,
  },
});
```

Emits

```
const emit = defineEmits({
  submit: (payload) => {
    if (payload.email) return true;
    console.warn('Invalid submit!');
    return false;
  },
});

function onSubmit() {
  emit('submit', { email: 'user@example.com' });
}
```

provide/inject

- **Use Case**: Cross-component dependency injection

```
// Ancestor
provide(
  'userData',
  reactive({
    id: 1,
    preferences: { theme: 'dark' },
  })
);

// Descendant
const userData = inject('userData', defaultValue);
```

Advanced Reactivity

toRefs()

- **Use When**: Destructuring reactive objects

```
const state = reactive({ x: 0, y: 0 });
const { x, y } = toRefs(state); // Maintain reactivity
```

shallowRef()

- **Optimization**: Skips deep reactivity

```
const heavyObject = shallowRef({
  /* 10k+ items */
});
```

customRef()

- **Custom Logic**: Create specialized refs

```
function useDebouncedRef(value, delay = 200) {
  return customRef((track, trigger) => {
    let timeout;
    return {
      get() {
```

```
        track();
        return value;
      },
      set(newValue) {
        clearTimeout(timeout);
        timeout = setTimeout(() => {
          value = newValue;
          trigger();
        }, delay);
      },
    };
  });
}
```

Template Refs & Directives

DOM Refs

```
<template>
  <input ref="emailInput" />
</template>

<script setup>
const emailInput = ref(null);
onMounted(() => emailInput.value.focus());
</script>
```

Custom Directives

```
const vHighlight = {
  mounted(el, binding) {
    el.style.backgroundColor = binding.value || 'yellow'
  },
  updated(el, binding) {
    el.style.backgroundColor = binding.value
  }
}
```

```
// Usage
<div v-highlight="'#ff0'"></div>
```

Async & Suspense

Async Components

```
const AsyncComp = defineAsyncComponent(() => import('./components/AsyncC
```

Async Setup

```
async function setup() {
  const data = await fetchData()
  return { data }
}

// With Suspense boundary
<Suspense>
  <template #default> <AsyncComponent /> </template>
  <template #fallback> Loading... </template>
</Suspense>
```

TypeScript Support

Type Inference

```
interface User {
  id: number;
  name: string;
}

const user = ref<User>({ id: 1, name: 'John' });
const users = reactive<User[]>([]);

// Component Props
defineProps<{
```

```
  title: string;
  items?: string[];
}>();
```

Effect Scope

- **Use Case**: Group effects for batch cleanup

```
const scope = effectScope();

scope.run(() => {
  watchEffect(() => console.log('Effect 1'));
  watchEffect(() => console.log('Effect 2'));
});

// Later
scope.stop(); // Cleans both effects
```

SSR Utilities

useSSRContext

```
import { useSSRContext } from 'vue';

// Server-side only
if (import.meta.env.SSR) {
  const ctx = useSSRContext();
  ctx.head += '<title>SSR Page</title>';
}
```

Performance Optimizations

markRaw()

- **Use When**: Opt-out of reactivity

```
const nonReactiveConfig = markRaw({
  immutable: true,
});
```

readonly()

- **Immutable Data**: Prevent accidental mutations

```
const protectedState = readonly(
  reactive({
    secret: '123',
  })
);
```

Utility Functions

unref()

- **Smart Access**: Returns .value for refs, else original

```
const value = unref(maybeRef);
```

isRef()/isReactive()

- **Type Checking**: Validate reactivity status

```
if (isRef(someVar)) {
  // Handle ref
}
```

toRef()

- **Property Conversion**: Create ref from reactive property

```
const user = reactive({ name: 'John' });
const nameRef = toRef(user, 'name');
```

Render Functions & JSX

h() Function

```
import { h } from 'vue';

export default {
  setup() {
    return () => h('div', { class: 'container' }, 'Hello World');
  },
};
```

useSlots()/useAttrs()

```
const slots = useSlots();
const attrs = useAttrs();
```

Error Handling

onErrorCaptured

```
import { onErrorCaptured } from 'vue';

onErrorCaptured((err, instance, info) => {
  console.error('Error:', err);
  return false; // Prevent propagation
});
```

Teleport

- **Use Case**: Render content outside component tree

```
<teleport to="#modals">
  <div class="modal">
    <!-- Modal content -->
  </div>
</teleport>
```

KeepAlive Integration

```
<KeepAlive :include="['ComponentA']" :max="5">
  <component :is="currentComponent" />
</KeepAlive>
```

Plugin Integration

```
// myPlugin.js
export default {
  install(app, options) {
    app.provide('myService', options.service);
    app.directive('focus' /* ... */);
  },
};

// main.js
import { createApp } from 'vue';
createApp(App).use(myPlugin, { service });
```

Debugging Tools

Debugging Refs

```
const debugRef = ref(0);
watchEffect(() => {
  console.log('Current ref value:', debugRef.value);
});
```

Component Inspector

```
import { getCurrentInstance } from 'vue';

const instance = getCurrentInstance();
console.log('Component instance:', instance);
```

Testing Utilities

Component Testing

```
import { mount } from '@vue/test-utils';

test('renders message', async () => {
  const wrapper = mount(Component);
  expect(wrapper.text()).toContain('Hello World');
});
```

Composables Testing

```
import { renderHook } from '@testing-library/vue';

test('useCounter', async () => {
  const { result } = renderHook(() => useCounter());
  expect(result.value.count).toBe(0);
  result.value.increment();
  expect(result.value.count).toBe(1);
});
```