

VUE 3 COMPOSITION API CHEAT SHEET

```
<template>
  <div>
    <p>Spaces Left: {{ spacesLeft }} out of {{ capacity }}</p>
    <h2>Attending</h2>
    <ul>
      <li v-for="(name, index) in attending" :key="index">
        {{ name }}
      </li>
    </ul>
    <button @click="increaseCapacity()">Increase Capacity</button>
  </div>
</template>
<script>
import { ref, computed } from "vue";
export default {
  setup() {
    const capacity = ref(4);
    const attending = ref(["Tim", "Bob", "Joe"]);
    const spacesLeft = computed(() => {
      return capacity.value - attending.value.length;
    });
    function increaseCapacity() {
      capacity.value++;
    }
    return { capacity, attending, spacesLeft, increaseCapacity };
  }
};
</script>
```

Use the composition API when:

The component is too large, and should be organized by logical concerns(feature).

AND/OR

Code needs to be extracted and reused across multiple components, as an alternative to Mixins/Scoped Slots.

AND/OR

Type safety in TypeScript is important.

If using Vue 2 with Composition API plugin configured:

```
import { ref, computed } from "@vue/composition-api";
```

Reactive Reference

Wraps primitives in an object to track changes

Computed Property

Access the value of a Reactive Reference by calling `.value`

Methods declared as functions

Gives our template access to these objects & functions

CAN ALSO BE WRITTEN AS:

```
import { reactive, computed, toRefs } from "vue";
export default {
  setup() {
    const event = reactive({
      capacity: 4,
      attending: ["Tim", "Bob", "Joe"],
      spacesLeft: computed(() => { return event.capacity - event.attending.length; })
    });
    function increaseCapacity() {
      event.capacity++;
    }
    return { ...toRefs(event), increaseCapacity };
  }
};
```

Reactive takes an object and returns a reactive object

Notice we don't have to use `.value` since the object is reactive

toRefs creates a plain object with reactive references



VUE 3 COMPOSITION API CHEAT SHEET

TO ORGANIZE BY FEATURE:

```
<template> ... </template>
<script>
export default {
  setup() {
    const productSearch = useSearch( 🔍 )
    const resultSorting = useSorting({ 📄 })

    return { productSearch, resultSorting }
  }
}
function useSearch(getResults) {
  🔍
}
function useSorting({ input, options }) {
  📄
}
</script>
```

TO EXTRACT SHARED CODE:

```
<template> ... </template>
<script>
import useSearch from '@use/search'
import useSorting from '@use/sorting'
export default {
  setup() {
    const productSearch = useSearch( 🔍 )
    const resultSorting = useSorting({ 📄 })

    return { productSearch, resultSorting }
  }
}
</script>
```

use/search.js

```
export default function useSearch(getResults) {
  🔍
}
```

use/sorting.js

```
export default function useSorting({ input, options }) {
  📄
}
```



Watch the Vue 3 Essentials course at VueMastery.com, taught by Gregg Pollack.

The setup() method

Called after beforeCreate hook and before created hook. Does not have access to this.

props

The first optional argument of setup:

```
export default {
  props: {
    name: String
  },
  setup(props) {
    watch(() => {
      console.log(`name is: ` + props.name)
    })
  }
}
```

Props are reactive and can be watched

context

The second optional argument of setup:

```
setup(props, context) {
  context.attrs;
  context.slots;
  context.emit;
}
```

Exposes properties previously accessed using this

life-cycle hooks

Declare them inside setup

```
setup() {
  onMounted(() => { ... });
  onUpdated(() => { ... });
  onUnmounted(() => { ... });
}
```

Instead of using beforeCreate or created hooks, just write code or call functions inside setup() instead.

See the API documentation for additional info.