Vue Mastery

VUE 3 COMPOSITION API

CHEAT SHEET (Part 1)

```
<script setup>
import { ref, computed } from "vue"

const capacity = ref(4);
const attending = ref(["Tim", "Bob", "Joe"])

const spacesLeft = computed(() => {
  return capacity.value - attending.value.length
});

const increaseCapacity = () => {
  capacity.value++
}
</script>

<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>
```

If using Vue 2 with Composition API plugin configured:

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

Reactive Reference
Wraps primitives/array in an object to track changes

Computed Property

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

***Methods** declared as functions*

The template has access to all `<script>` *variables automatically*

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 mulitiple components, as an alternative to Mixins/Scoped Slots.

AND / OR

Type safety in TypeScript is important.

CAN ALSO BE WRITTEN AS:

```
import { reactive, computed, toRefs } from "vue"

const event = reactive({
  capacity: 4,
  attending: ["Tim", "Bob", "Joe"],
  spacesLeft: computed(() => { return event.capacity - event.attending.length; })
});

function increaseCapacity() {
  event.capacity++
}

const { capacity, attending, spacesLeft } = toRefs(event)
```

***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 Mastery

VUE 3 COMPOSITION API
CHEAT SHEET (Part 2)

TO ORGANIZE BY FEATURE:

```
<script setup>

function useSearch(getResults) {
  🔍
}

function useSorting({ input, options }) {
  📝
}

const productSearch = useSearch(🔍)
const resultSorting = useSorting({📝})

</script>

<template>
 …
</template>
```

TO EXTRACT SHARED CODE:

```
<script setup>
import { useSearch } from '@use/search'
import { useSorting } from '@use/sorting'

const productSearch = useSearch(🔍)
const resultSorting = useSorting({📝})
</script>

<template>
 …
</template>
```

use/search.js

```
export const useSearch = function(getResults) {
  🔍
}
```

use/sorting.js

```
export const useSorting = function({ input, options }) {
  📝
}
```

To master Vue 3, explore our course library at **VueMastery.com**

Missing hooks in `<script setup>`

There are no **`onBeforeCreated`** and **`onCreated`** hooks. Just put the corresponding code inside **`<script setup>`**.

props — The defineProps compiler macro:

```
<script setup>
const props = defineProps({
  name: String
})
watch(() => {
  console.log(`name is: ` + props.name)
})
</script>
```

Props are reactive and can be watched

context — Use the context-related composables

```
<script setup>
import { useSlots, useAttrs } from "vue";

const slots = useSlots()
const attrs = useAttrs()
</script>
```

life-cycle hooks — Write them inside <script setup>

```
<script setup>
import { onMounted, onUpdated } from "vue";

onMounted(() => { ... })
onUpdated(() => { ... })
</script>
```

Instead of using `mounted` or `updated` hooks, just write code or call functions inside setup() instead.

See the API documentation for additional info.