1. **Composition API initial boiler plate,**

```
<template>
  <div class="home">
    <p>Home</p>
  </div>
</template>

<script>
// @ is an alias to /src

export default {
  name: 'HomeView',
  setup() {
    console.log('Component setup');
  },

  created() {
    console.log('Component created');
  },

  mounted() {
    console.log('Component mounted');
  }

}
</script>
```

composition API er **setup** function er kaj ta jevabe execute hoy,

```
export default {
  name: 'HomeView',
  setup() {

    // properties

    let name = "mario";
    let age = 30;

    // functions

    const handleClick = () => {
      console.log('You clicked me')
    }

    // return to use in templates

    return {
      name, age, handleClick
    }
  },
}
```

Now from templates,

```
<template>
  <div class="home">
    <p>{{ name }}</p>
    <button @click="handleClick">Handle Click</button>
  </div>
</template>
```

2. Composition API [Template Refs],

Step 1: import kore nite hobe **ref** ke,

```
import {ref} from "vue";
```

Step 2: kono ekta variable er moddhe assign kore nite hobe,

```
const p = ref(null);
```

Step 3: return the value,

```
return {
  p
}
```

Step 4; ekhn template er moddhe **ref** ke assign kora jabe,

```
<p ref="p">{{ name }}</p>
```

Step 5: ekhn ref er ei value proyojon moto use kora jabe,

```
const handleClick = () => {
  console.log(p.value.textContent)
}
```

3. Composition API [Making value reactive],

Step 1: changing values in **ref** mode,

```
let name = ref('HomeView')
let age = ref(10)
```

Step 2: for an example, changing values through functions,

```
let name = ref('HomeView')
let age = ref(10)
```

For an example, ref add korar por reactive state e input type e v-model add kore data changing test kora,

```
<input  type="text" v-model="name" placeholder="Name" />
```

Full code snippet,

```
<template>
  <div class="home">
    <p ref="p">{{ name }}</p>
    <button @click="handleClick">Handle Click</button>
    <input  type="text" v-model="name" placeholder="Name" />
  </div>
</template>

<script>
// @ is an alias to /src

import {ref} from "vue";

export default {
  name: 'HomeView',
  setup() {
    // properties

    const p = ref(null);

    let name = ref('HomeView')
    let age = ref(10)

    // functions

    const handleClick = () => {
      name.value = "Dd"
    }

    // return to use in templates

    return {
      name, age, handleClick, p
    }
  },
}
</script>
```

4. Computed Property,

Basic snippet of computed property usage,

```
<template>
  <div class="home">
    <h1>{{ name }}</h1>
  </div>
</template>

<script>

// import {ref} from "vue";

import {computed} from "vue";

export default {
  name: 'HomeView',
  setup() {

    const name = computed(()=>{
      return 'Shawn'
    })

    return {
      name
    }
  },
}
</script>
```

- **Computed** property always recomputes, when changes appear it modifies

An example of dynamic search using computed properties,

```
<template>
  <div class="home">
    <input type="text" v-model="search">
    <div v-for="name_data in matchingNames" :key="name_data">
      {{ name_data }}
    </div>

  </div>
</template>

<script>
// @ is an alias to /src


import {ref, computed} from "vue";

export default {
  name: 'HomeView',
  setup() {

    let search = ref('');
    const names = ref(['mario', 'yoshi', 'luigi', 'toad', 'bowser', 'koopa', 'peach']);


    // Use a computed property to reactively update the filtered names

    const matchingNames = computed(() => {
      return names.value.filter(el =>
          el.toLowerCase().includes(search.value.toLowerCase())
      );
    });


    return {
      search, matchingNames
    }
  },
}
</script>
```

Watch property,

Watch property hocche kono changes pele identify kore output back kore

```
watch(search, (value) => {
  console.log(value)
})
```

watch() property er snippet,

```
import {ref, computed, watch} from "vue";

export default {
  name: 'HomeView',
  setup() {

    let search = ref('');

    watch(search, (value) => {
      console.log(value)
    })


    return {
      search
    }
  },

}
</script>
```

watchEffect() eta onkta mounted() property er motoi kaj kore, kono parameter expect kore na. component e kono changes pelei run kore, jekhane mounted only ekbar e execute hoto

```
watchEffect(()=>{
  console.log("watch effect")
});
```

Stopping the effects,

Step 1:

```
const stopEffect = watchEffect(()=>{
  console.log("watch effect")
});
```

Step 2: kono ekta function or click event er moddhe evabe trigger korlei effect off ohye jabe,

```
stopEffect();
```