



# Day3

 자료	<u>Vue</u>
 구분	Vue

## computed

다음 코드를 살펴보자.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <div id="app">
    <h1>오늘할일</h1>
    <ul>
      <li v-for="todo in todos" :key="todo.id">
        <span>{{ todo.text }}</span>
      </li>
    </ul>
    <button @click="hideCompleted = !hideCompleted">
      {{ hideCompleted ? "모두 보이기" : "완료된 일 숨기기" }}
    </button>
  </div>

  <script>

    const { createApp, ref } = Vue

    const app = createApp({

      setup() {
        const hideCompleted = ref(false);
        const todos = ref([
          { id: 1, text: 'HTML 배우기', done: true },
          { id: 2, text: 'JavaScript 배우기', done: true },
          { id: 3, text: 'Vue 배우기', done: false },
        ]);
        return {
          hideCompleted,
          todos
        }
      }
    });

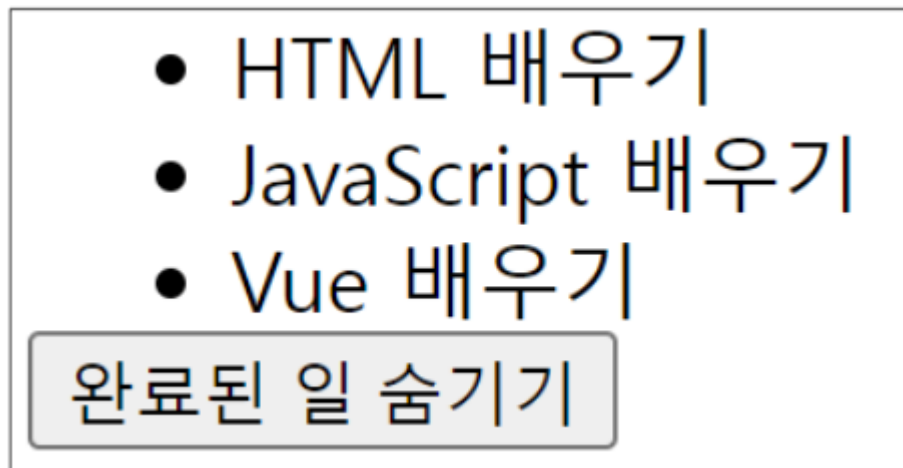
    app.mount('#app');
```

```

    }
  }
})
app.mount('#app')
</script>
</body>

</html>

```



간단한 토글버튼이며, 클릭할때마다 `hideComplete` 가 `true` / `false` 로 바뀌면서, 삼항연산자에 의해 버튼 텍스트도 바뀐다.

이 예제에서 하고싶은 일은, 버튼을 누를 때 배열의 `done` 을 판단해, 완료된 할 일을 숨기거나 전체 목록을 다 보이는 것이다. 이를 위해 JavaScript 배열 `filter` 메서드를 사용하면 될 것이다.

어떻게 하면 될까? `v-for` 반복의 기준이 되는 state 를 `todos` 에서, `filteredTodos` 라는, 상황에 따라 계산된(computed) 결과를 도출하는 특별한 state 로 바꿀 것이다.

이를 위해 `computed` 메서드를 사용할 것이며, `vue` 객체에서 импорт 해야 한다.

```

<script>

const { createApp, ref, computed } = Vue

const app = createApp({

  setup() {
    const hideCompleted = ref(false)

    const todos = ref([
      { id: 1, text: 'HTML 배우기', done: true },
      { id: 2, text: 'JavaScript 배우기', done: true },
      { id: 3, text: 'Vue 배우기', done: false },
    ])
  }
})

```

```

const filteredTodos = computed(() => {
  if (hideCompleted.value) {
    return todos.value.filter((todo) => !todo.done)
  } else {
    return todos.value
  }
})

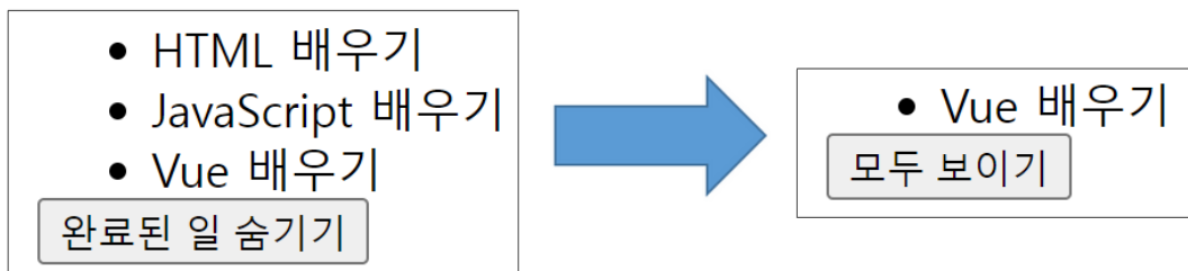
return {
  hideCompleted,
  todos,
  filteredTodos
}
}
})
app.mount('#app')
</script>

```

`computed` 메서드의 아규먼트로 콜백함수가 들어간다. 현재 `hideCompleted` 의 값을 판단하여 `filteredTodos` 의 값을 다르게 만든다.

그리고 `v-for` 구문의 `todos` 를 `filteredTodos` 로 변경해주자.

```
<li v-for="todo in filteredTodos" :key="todo.id">
```



- `computed` 를 사용하면, 템플릿에 `computed` 로 만들어진 state 가 있을 때, 변경이 일어나는 즉시 화면을 다시 렌더링한다. 언뜻 보면 일반 함수 사용과 다를바가 없어보이는데, `computed` 는 캐싱을 하기 때문에 변경이 자주 일어나는 경우 일반 함수보다 훨씬 효율적으로 동작한다.
- `computed` 는 함수라고 생각해서 안된다. side effect, 즉 리턴 이외에 값을 변경하는 행위를 해서는 안된다. 돔 조작이나 비동기통신이 가장 대표적인 예다. `computed` 함수는 태그 안에 지나치게 많은 논리를 간략하게 할 때 많이 쓰인다고 생각해도 과언은 아니다.

- computed와 method의 차이점에 대해서 분명하게 인지해야 한다. computed는 의존하는 데이터가 변경되면 자동으로 업데이트가 되지만, method는 호출을 해야 실행된다!
- computed와 watchers와도 혼동하지 않도록 유의 해야 한다. computed는 연산된 값 이, 필터링 된 목록 계산 등의 목적이고 watchers는 차후 비동기 API 요청 및 연관 데이터 업데이트 용도라는 것을 잊지 말자.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <div id="app">
    <p>책을 가지고 있는지 yes or no 로 대답하십시오</p>
    <span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
  </div>

  <script>

    const { createApp, ref } = Vue

    const app = createApp({

      setup() {
        const author = ref({
          name: 'John Doe',
          books: [
            'Vue 2 - Advanced Guide',
            'Vue 3 - Basic Guide',
            'Vue 4 - The Mystery'
          ]
        })

        return {
          author
        }
      }
    })
    app.mount('#app')
  </script>
</body>

</html>
```

다음과 같이 `computed` 에게 넘겨 가독성을 강화할 수 있다.

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <div id="app">
    <p>책을 가지고 있는지 yes or no 로 대답하시오</p>
    <span>{{ publishedBooksMessage }}</span>
  </div>

  <script>
    const { createApp, ref, computed } = Vue
    const app = createApp({

      setup() {
        const author = ref({
          name: 'John Doe',
          books: [
            'Vue 2 - Advanced Guide',
            'Vue 3 - Basic Guide',
            'Vue 4 - The Mystery'
          ]
        })
        const publishedBooksMessage = computed(() => {
          return author.value.books.length > 0 ? 'Yes' : 'No'
        })
        return {
          author,
          publishedBooksMessage
        }
      }
    })
    app.mount('#app')
  </script>
</body>

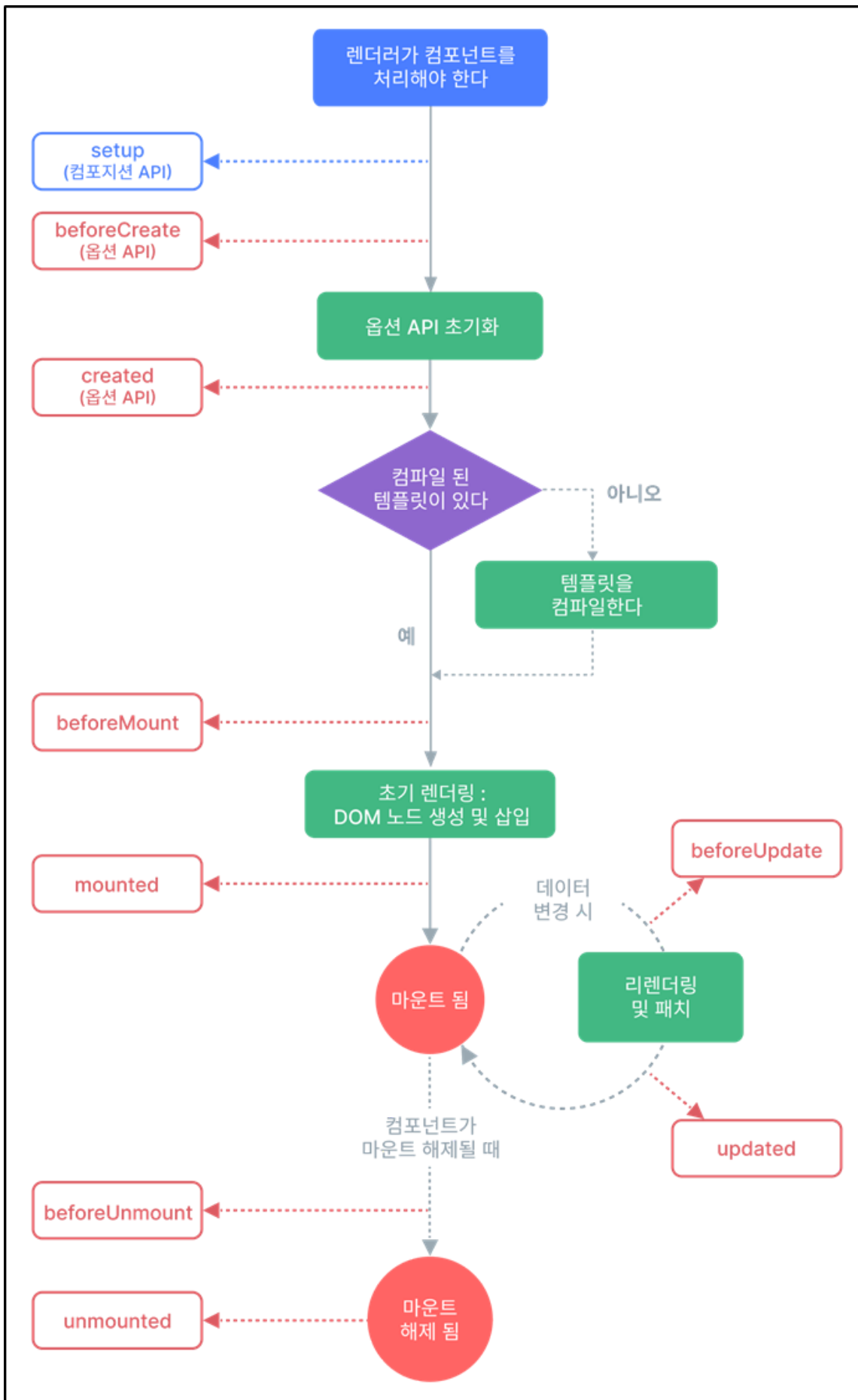
</html>

```

## 10. lifecycle hooks ( **onMounted** )

Vue.js 에선 컴포넌트의 생명주기에 따라 자동실행되는 함수들이 존재하며, 이를 lifecycle hooks 라고 한다. 이를 잘 활용해, 개발자는 브라우저에 랜더링 하는 특정 단계에서 자신이 의도하는 로직을 실행할 수 있다.





아래 문서에서 전체 API 를 확인할 수 있다.

<https://v3-docs.vuejs-korea.org/api/composition-api-lifecycle.html>

컴포넌트 생성부터 마운팅 해제까지 각 단계별 어떠한 일이 일어나는지 간략하게만 살펴보자.

- Creation : 컴포넌트 초기화 단계
  - beforecreate - 뷰 인스턴스 초기화 직후 생성단계를 의미한다. 즉 뷰 인스턴스가 딱 만들어 졌지만 셋팅은 덜 된 상태를 의미한다. 코드를 예를 들자면 `const { createApp, ref, onMounted } = Vue` 코드만 딱 실행 되었을 타이밍을 의미한다. (뷰 인스턴스가 막 생성된 단계)
  - created - 뷰 인스턴스의 셋팅이 완료가 되었을때를 의미하며 이때 부터 data와 event가 활성화 되어 접근이 가능하다. 컴포넌트가 생성되긴 했지만 아직 화면에 보이지 않은(마운팅 되지 않은) 단계를 의미한다.
- Mounting : 돔(DOM) 삽입 단계로, 마운트(Mount)는 DOM 객체가 생성되고 브라우저에 나타나는 것을 의미한다. 보통 화면에 붙는다 라고 표현 한다. 말 그대로 브라우저에 '나타나는' 것이기 때문에 유저가 직관적으로 확인할 수 있는 부분이다.
  - beforeMount - 컴포넌트가 DOM에 추가 되기 직전 단계를 의미 하는데 거의 의식하고 사용을 하지 않는 라이프사이클 훅이다.
  - Mounted - 컴포넌트가 DOM에 추가 된 후 호출되는 단계로 DOM에 접근이 가능한 시점이다.
- Update - 컴포넌트에서 사용되는 속성들이 변경되는 과정 그리고 컴포넌트가 재 렌더링 되면 실행 되는 라이프 사이클이다.
- Destroy - 컴포넌트가 제거 될 때를 의미하는 라이프 사이클 이다. destroyed 단계가 되면 컴포넌트의 모든 이벤트 리스너와(@click, @change 등) 디렉티브(v-model, v-show 등)의 바인딩이 해제 되고, 하위 컴포넌트도 모두 제거되는 단계를 말한다.

사실, 중급 개발자가 되기 전까지, lifecycle hooks 는 당장 사용 할 일이 없다고 봐도 무방하다. 다만 Vue 의 필수 개념중의 하나로 분류되기 때문에 개념은 살펴보고 일단 넘어 갈 뿐이다. Vue.js 개발자는 모든 라이프 사이클 훅을 다 알고 활용하진 않는다. 개발 하고자 하는 앱에 따라, 단 하나의 라이프사이클 훅도 사용하지 않을 수도 있다. 이 라이프 사이클 훅 중에,



라이브 시간에는 그나마 활용성 있는 `onMounted` 혹에 대해 알아보았다. `mounted` 라는건 무슨 뜻일까? 컴포넌트가 화면에 붙은 후의 시점을 의미한다. 즉, `onMounted` 는 컴포넌트가 화면에 붙은 후 자동 실행되는 콜백함수라고 볼 수 있겠다.