

Day1

📎 자료	<u>Vue</u>
☰ 구분	Vue

1. Hello Vue
2. `{{ }}` (mustache)
3. `v-model`
4. `@` (`v-on`)

1. Hello Vue

먼저, 원하는 디렉터리를 vscode로 열어보자. 그리고 다음과 같이 입력해 프로젝트를 생성한다.

```
$ npm init vue@3
```

npm 은 Node Package Manager의 약자로 전 세계 오픈소스 자바스크립트 패키지를 모아 놓은 저장소다.

```
Vue.js - The Progressive JavaScript Framework

// 프로젝트 이름
√ Project name: ... vue-project

// 타입스크립트 사용. No 선택
√ Add TypeScript? ... No / Yes

// JSX 사용. No 선택
√ Add JSX Support? ... No / Yes

// 라우터 사용. Yes 선택
√ Add Vue Router for Single Page Application development? ... No / Yes

// Pinia 사용. Yes 선택
√ Add Pinia for state management? ... No / Yes

// 유닛테스팅. No 선택
```

```
√ Add Vitest for Unit Testing? ... No / Yes

// 테스트 솔루션. No 선택
√ Add an End-to-End Testing Solution? » No

// ESLint. No 선택
√ Add ESLint for code quality? ... No / Yes

Scaffolding project in C:\Users\SSAFY\Desktop\vue-project...
```

폴더가 생성 되면서 vue로 프로젝트를 진행 할 때 필요한 필수 요소들이 최소한으로 생성 되었다. 그리고 폴더를 열어보면 `vue-project/package.json` 파일이 있는데 이는 이 프로젝트의 간단한 정보 및 의존성을 명시해 놓은 것이다.

`cd vue-project` : 현재 위치에 존재하는 `vue-project/` 디렉터리로 이동하자.

```
$ cd vue-project/
```

이제 `package.json` 에 명시 되어 있는 의존 패키지 들을 설치 할 것이다.

```
$ npm install
```

`npm install` 을 통해서

`vue-project/package.json` 의 `dependencies` , `devDependencies` 에 기록된 패키지 들을 설치했다. 그럼 `node_modules` 라는 폴더가 생성 되었을 것이다.

그리고 프론트엔드 서버를 구동 해 보자.

```
$ npm run dev
```

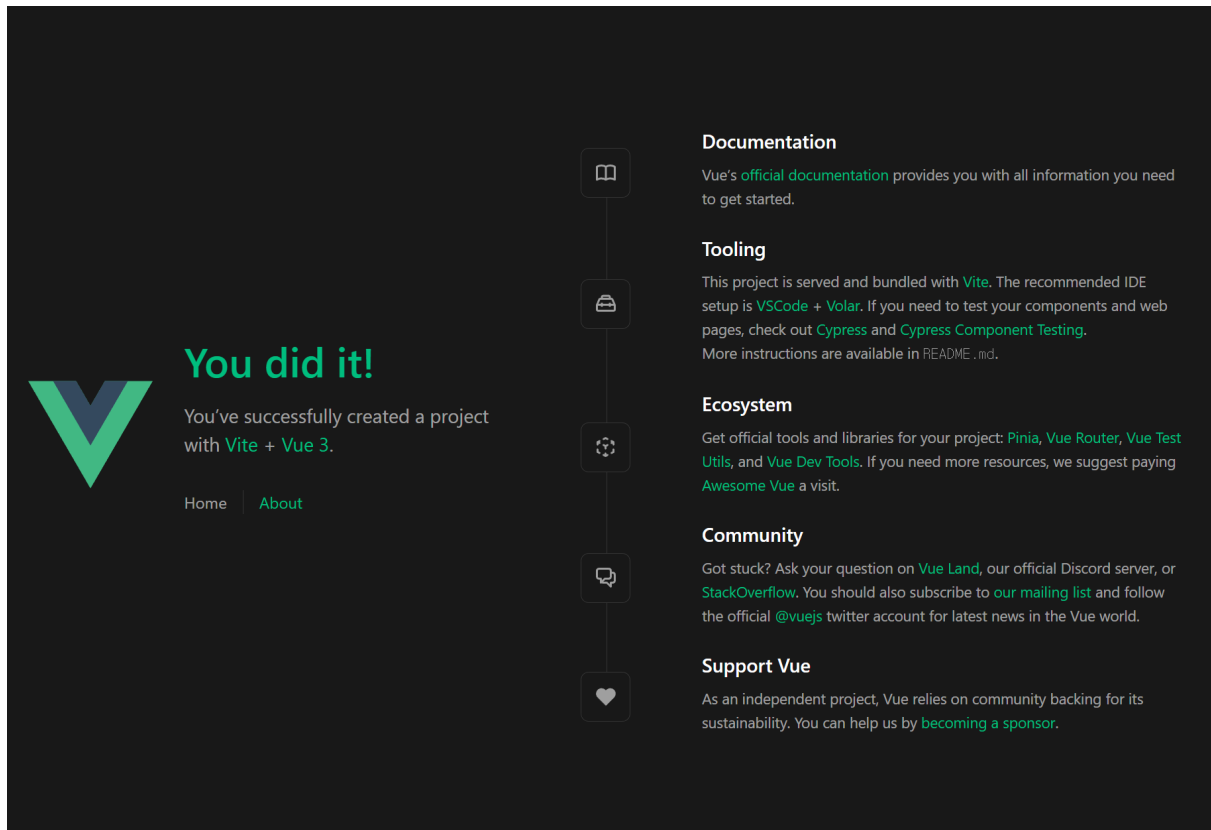
세 가지 명령어를 그대로 따라 치면, 다음과 같은 화면이 나온다.

```
VITE v4.5.0 ready in 383 ms

➔ Local:   http://localhost:5173/
```

- Network: use `--host` to expose
- press `h` to show help

브라우저를 켜고 `http://localhost:5173` 으로 접속하면 다음과 같다.



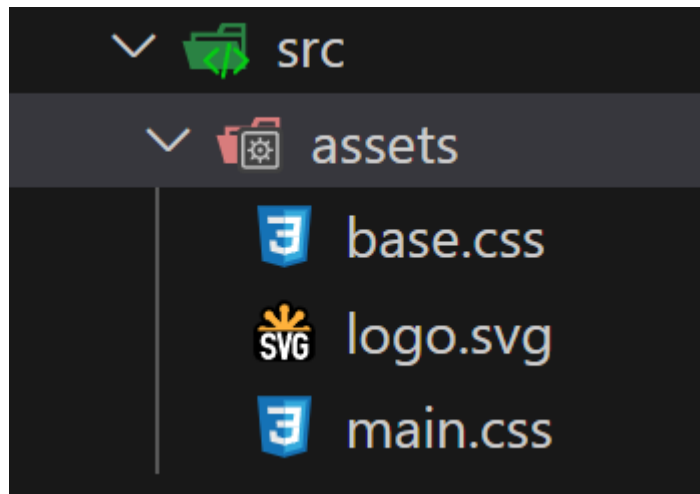
이것은 프론트엔드 서버이다. 우리가 프로젝트에서 코드를 수정하면 즉각 이 화면에 반영된다.

`ctrl + c` 로 서버 종료하다.

혹시 vscode 확장패키지를 안 했다면 지금이라도 설치하자.

싸피 공용문서를 보고 설치하면 된다.

그다음, `src/` 에서 필요없는 파일들을 삭제하겠다.



`src/assets/` 디렉터리부터 정리하겠다.

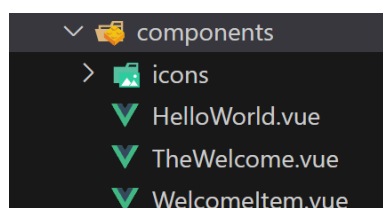
우선 `base.css` 와 `logo.svg` 를 삭제하고, `main.css` 의 모든 내용을 지운 후, 다음과 같이 입력한다.

```
* {  
  box-sizing: border-box;  
  margin: 0;  
}
```

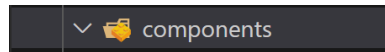
- `main.css` 는 전역 스타일링 파일이다.
- `box-sizing` 은 박스의 크기를 화면에 표시하는 방식을 변경하는 속성이다. 테두리가 있는 경우에는 테두리의 두께로 인해서 원하는 크기를 찾기가 어려운데, `box-sizing` 속성을 `border-box` 로 지정하면 테두리를 포함한 크기를 지정할 수 있기 때문에 크기를 예측하기가 더 쉽다. 기본적으로 프로젝트를 하나 만들면, 모든 엘리먼트에 이 값을 지정하고 시작한다.
- `margin: 0` 은 각 태그에 기본적으로 잡혀있는 공백을 제거해준다.

다음, `src/components/` 하위 모든 파일, 디렉터를 삭제한다.

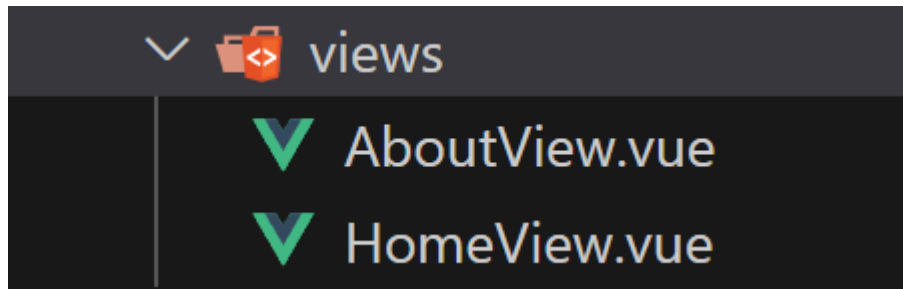
[삭제전]



[삭제후]



다음, `src/views/` 아래의 `HomeView.vue` 와 `AboutView.vue` 를 다음과 같이 변경한다.



```
<template>
  <h1>About</h1>
</template>
```

```
<template>
  <h1>Home</h1>
</template>
```

마지막으로, `src/App.vue` 를 다음과 같이 변경한다.



```
<script setup>
import { RouterLink, RouterView } from 'vue-router'
</script>

<template>
  <header>
    <nav>
      <RouterLink to="/">Home</RouterLink> |
      <RouterLink to="/about">About</RouterLink>
    </nav>
  </header>
```

```
<RouterView />
</template>
```

여기까지 완료한 후, 서버를 동작시킨다.

```
$ npm run dev
```

```
VITE v4.5.0 ready in 383 ms
```

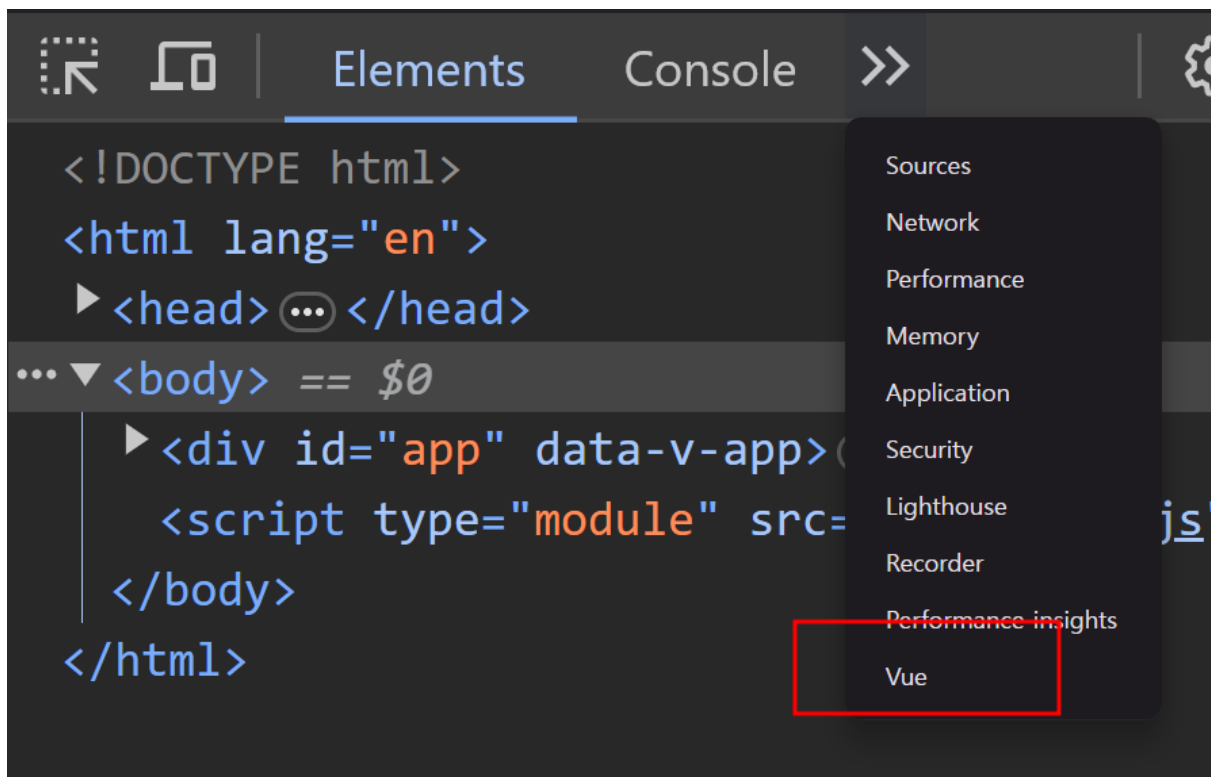
```
→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

`localhost:5173` 로 접속해보자.

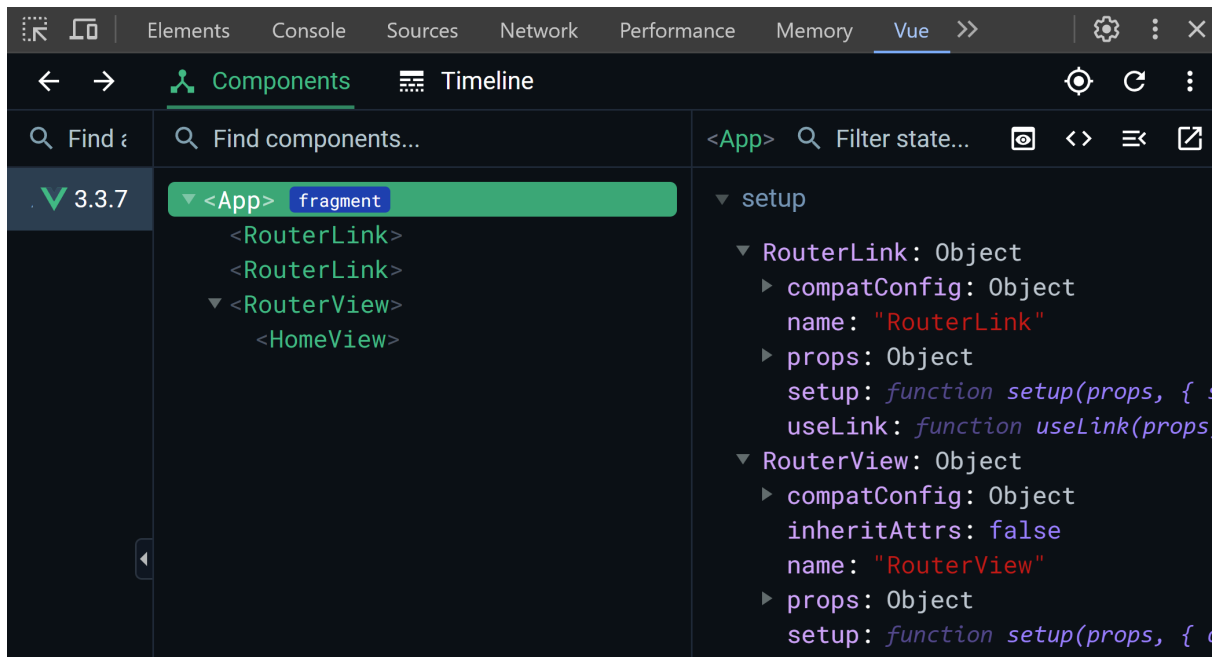
[Home](#) | [About](#)
Home

`Home` , `About` 을 누를 때마다 화면이 바뀌는 것을 확인할 수 있다.

`F12` 버튼을 눌러보자.



화살표 `>>` 를 눌러, `Vue` 가 보이는지 확인한다.



위와 같이 확인된다면 크롬 확장 프로그램 `Vue.js devtools` 가 성공적으로 설치된 것이다.

- 앞으로 우리는 특정 시기에 이르기 전까지는 거의 `src/views/HomeView.vue` 를 변경해가며 실습을 진행할 것이다.
- 프로젝트의 구조에 관한 상세 설명은, component 챕터에 들어가기 직전에 진행하도록 하겠다. 처음부터 지나치게 이론만 설명하면 Vue의 재미를 느끼기 힘들기 때문이다.

2. `{{ }}` (mustache)

`{{ }}` 라는 기호는 콧수염처럼 생겼다고 해서 mustache 라고 부른다.

일단은 무지성으로 다음 코드를 입력해보자.

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

const message = ref("Hello Vue");
</script>

<template>
```

```
<div>message: {{ message }}</div>
</template>
```

message: Hello Vue

`HomeView.vue` 처럼, `.vue` 확장자가 붙은 파일을 하나의 “컴포넌트”라고 부른다.

```
<script setup>
// js code 가 들어감
</script>

<template>
<!-- html code 가 들어감 -->
</template>

<style scoped>
/* css code 가 들어간다 */
</style>
```

이 중, 당장은 스타일링을 할 필요가 없으니 CSS 를 담당하는 `<style>` 은 잠시 생략 한다.

당연히, `template` 에는 화면에 나올 HTML 코드를 작성하는 부분이다.

`script` 에는 자바스크립트 또는 타입스크립트 등을 이용해 스크립트 코드를 작성한다. 이를 통해서 템플릿에서 사용한 `state` 라고 하는 변수를 조작 할 수 있다.

다시 아까 Hello Vue 를 출력 했던 코드를 보자.

`ref` 는 state (상태)라고 불리는 특별한 변수를 생성할 때 사용하며, `ref` 를 사용을 위해 `vue` 에서 import 해야한다.

`ref` 의 argument 로 들어가는 값은 `message` state 의 값이 된다.

- state 는 특별한 변수다. 보통의 변수 값이 바뀐다고 해서 화면 내용이 바뀌진 않는다. 그러나 state 가 변하면 화면에서 해당 state 와 연결된 부분이 다시 그려지게 된다.
- 화면을 그린다는 표현을 프론트엔드 개발에선 렌더링이라고 한다.
- 앞으로 일반 변수와 구분하기 위해, 변수라고 표기하지 않고 state 라고 언급 하겠다.

화면에 출력되는 것을 보고 판단해봤을 때, `{{ message }}` 는 `const message` 로 선언된 state 임을 알 수 있다.

즉, `{{ }}` 는 state 를 가져오는 역할을 한다.

- 이게 뭐가 특별한 일인가? 싶겠지만 HTML 은 프로그래밍 언어가 아니라 마크업 언어이다. 즉, HTML 에선 변수, if, for 등을 사용할 수 없다. 그러나 Vue.js 는 HTML 안에서 변수, 정확하게는 state 를 사용 함으로써 심지어 if 와 for 도 적용해 특정 조건에 맞추어 태그를 화면에 보이게 하거나, 태그를 반복 시킬 수도 있다.

한 가지만 더 살펴보자.

방금 작성했던 코드에서는 `<script setup>` 이라고 `<script>` 안에 setup을 적어 주었는데 이는 라이브 교재(교안)에 있는 `setup()` 함수 대신 사용해 준 것이다.

`<script setup>` 은 Vue 컴포넌트의 `setup()` 함수를 더 간단하게 작성할 수 있도록 도와주는 기능으로 Vue3 공식 문서 에서도 `<script setup>` 을 사용하라고 권장하고 있다.

하지만 이건 공식문서의 입장일 뿐이다. 실제로 `<script setup>` 을 가지고 개발을 해보면 개발자가 작성한 javascript 코드를 아직 까지는 100% 완벽하게 인식을 못한다.

예를 들면 `<script setup>` 기능은 javascript의 this 키워드를 사용해서 인스턴스에 접근이 불가능 하다. 뿐만 아니라, Vue3 이상의 버전에서만 사용할 수 있으며 일부 기능, 예를 들면 data, method, computed 와 같은 Options api 사용은 제한될 수 있다는 단점이 있다.

그러한 이유로 ssafy 교안 에서는 범용성을 생각해서 `setup()` 컴포지션 함수를 채택해서 사용 한 것 같다는 것이 내 생각이다. 그러나 `<script setup>` 는 javascript가 아닌 typescript 를 사용함에 있어서 아주 좋은 `setup()` 함수의 대안이라서 앞으로 더 지켜봐야 할 부분이다. 코드가 정말 많이 간결해 지기 때문에 앞으로 기대 할 만 하다.

(아 그렇구나.. 하고 넘어가자. 아직까지 현업에서는 vue2로 구현된 서비스가 많아서 vue2 란 어떤 점이 다른지도 알고는 있어야 한다고 생각 되어서 적어 보았다. 참고로 vue2 버전에서는 data, method, computed를 사용하는 Options api 라는 것 만 존재 했다. 하지만 우리가 학습 할 `setup()` 함수는 `<script setup>` 와 함께 Vue3에서 새로 도입된 API 다. 이번 기

수에서는 vue2에서 사용되는 Options api를 사용하는 것이 아니라, `setup()` 함수를 사용해서 프로젝트를 진행한다.)

어찌 되었든, setup 함수에 대해서 알아보자.

지금까지 vue 실습을 npm 설치 방식으로 하였으나 교안에 있는대로 cdn 방식으로 먼저 실습을 진행 하겠다. (핑계를 대자면.. 사실 Vue3 새 교재가 몇 일 전에 배포되었다. 그래서 수업 초반에 cdn 방식으로 할 줄은 생각지도 못했다. 어제 알았다ㅎ)

미안하지만 다 지우고 ㅎㅎ 새로 폴더 만들어서 오늘은 교안대로 html 파일 하나 만들어서 실습 해 보자.

setup 함수를 구성하는 내용은 javascript처럼 스크립트로 작성하면 된다.

setup 함수는 객체를 반환하는데 이 객체에는

javascript의 이벤트핸들러 함수 그리고 화면을 담당하는 HTML에서 사용할 변수 등이 들어간다.

```
<!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="심우석학생은이서진답아서매력적이라고생각함">
    <h1>{{ hello }}</h1>
  </div>
  <script>
    const {createApp,ref}=Vue

    app = createApp({
      setup() {
        const hello = ref('hello Vue')

        return {
          hello
        }
      }
    })

    app.mount( '#심우석학생은이서진답아서매력적이라고생각함' )
  </script>
```

```
</body>
</html>
```

앞으로 한글은 사용하지 말자! 진심이다.

위 코드에 주석을 달면 다음과 같을 것이다.

```
<!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>
  <!-- cdn 복붙하기 -->
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <div id="심우석학생은이서진닭아서매력적이라고생각함">
    <h1>{{ hello }}</h1>
  </div>

  <script>

    // const createApp=Vue.createApp
    // const ref=Vue.ref
    // 1. 속성명 축약한 것이다.
    const {createApp,ref}=Vue

    // 2. 모든 Vue앱은 createApp 함수를 사용하여 새로운 앱 인스턴스를 생성하는 것으로 시작
    app = createApp({

      // 3. setup 함수를 통해서 객체를 반환하는데
      setup() {

        // 4. ref 함수는 hello 라는 state변수 생성시 사용하는 함수
        const hello = ref('hello Vue')

        // 5. 반환하는 객체 안에는 javascript의 이벤트핸들러 함수 그리고(또는)
        // 화면을 담당하는 HTML에서 사용할 state변수가 반환 되어야 한다.
        return {
          hello
        }
      }
    })

    // 6. app 마운트 하기: 앱 인스턴스는 .mount() 메서드가 호출될 때까지 아무것도 렌더링하지 않는다.
    app.mount( '#심우석학생은이서진닭아서매력적이라고생각함' )
  </script>
</body>
</html>
```

이번에는 `message` state 를 변경해보자.

3. `v-model`

`v-model` 은 `<input>` 태그에 사용된다. 코드를 다음과 같이 수정해보자.

```
<!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="심우석학생은이서진 닮아서 매력적이라고생각함">
    <h1>message: {{ hello }}</h1>
    <input type="text" v-model="hello">
  </div>

  <script>
    const {createApp,ref}=Vue

    app = createApp({

      setup() {

        const hello = ref("")

        return {
          hello
        }
      }
    })
    app.mount( '#심우석학생은이서진 닮아서 매력적이라고생각함' )
  </script>
</body>
</html>
```

[`<script setup>` 옵션은 참고용으로만 봐주세요]

```

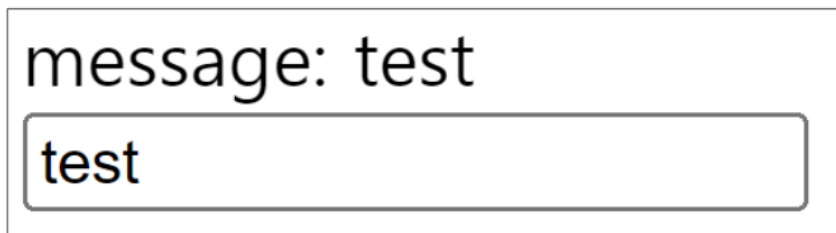
<script setup>
import { ref } from "vue";

const message = ref("");
</script>

<template>
  <div>message: {{ message }}</div>
  <input type="text" v-model="message" />
</template>

```

`message` 를 빈 문자열로 `""` 두었고, `<input>` 태그에 `v-model="message"` 를 달았다.



입력할 때마다 메시지가 바뀌어서 렌더링이 되는 신기한 현상을 경험할 수 있다.

즉, `v-model` 은 사용자 입력값을 state 에 실시간 저장할 때 사용한다.

`v-model` 은 모든 `<input>` 태그에 사용 가능하므로, 체크박스도 물론 활용 가능하다.

```

<!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="심우석학생은이서진 님아서매력적이라고생각함">
    <h1>message: {{ hello }}</h1>
    <input type="text" v-model="hello">

    <p>

```

```

      <input type="checkbox" id="checkbox" v-model="checked">
      <label for="checkbox">{{checked}}</label>
    </p>

  </div>

  <script>
    const {createApp,ref}=Vue

    app = createApp({

      setup() {

        const hello = ref("")
        const checked = ref(false)

        return {
          hello,
          checked,
        }
      }
    })
    app.mount( '#심우석학생은이서진애피아서매력적이라고생각함' )
  </script>
</body>
</html>

```

[<script setup> 옵션은 참고용으로만 봐주세요]

```

<script setup>
import { ref } from "vue";

const checked = ref(false);
</script>

<template>
  <input type="checkbox" id="checkbox" v-model="checked" />
  <label for="checkbox">{{ checked }}</label>
</template>

```



클릭할때마다 `true`, `false` 가 변경되는것을 알 수 있다.

- `v-model` 은 `<input>` 태그에 쓰이고, `<input>` 태그가 나오면 무조건 `v-model` 을 사용한다. 이 원칙은 변하지 않으며, 추후 배우게 될 `v-bind` 를 `<input>` 태그에 사용하는 일은 없도록 하자.

=====

4. @ (v-on)

이번에는 이벤트를 받아 보자.

```
<!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="심우석학생은이서진닭아서매력적이라고생각함">
    <h1>message: {{ hello }}</h1>
    <input type="text" v-model="hello">

    <p>
      <input type="checkbox" id="checkbox" v-model="checked">
      <label for="checkbox">{{checked}}</label>
    </p>

    <p>
      <div>{{ text }}</div>
      <button v-on:click="changeText">클릭하면 글자가 변해!</button>
    </p>

  </div>

  <script>

    const { createApp, ref } = Vue

    app = createApp({
      setup() {
        const hello = ref("")
        const checked = ref(false)
        const text = ref("??")
```

```

    const changeText=()=>{
      text.value="짜잔"
    }
    return {
      hello,
      checked,
      text,
      changeText,
    }
  }
})

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

</html>

```

[<script setup> 옵션은 참고용으로만 봐주세요]

```

<script setup>
import { ref } from "vue";

const text = ref("???");

function changeText() {
  text.value = "짜잔";
}
</script>

<template>
  <div>{{ text }}</div>
  <button v-on:click="changeText">클릭하면 글자가 변해!</button>
</template>

```



이벤트를 받을 땐 `v-on` 으로 받으며, 콜론 `:` 뒤에 받을 이벤트명 `click` 을 적어준다.

이끌 `=` 기호 다음에 이벤트 발생 시 실행할 함수를 콜백으로 달아주면 된다.

클릭하면 `changeText` 함수가 실행되며, `text` 는 `"???"` 에서 `"짜잔"` 으로 바뀐다.

- 함수 안에서 state 의 값에 접근하려면, 반드시 `.value` 를 붙여줘야한다. 이 경우엔 `text.value` 로 접근한다.
- 그러나, `<template>` 안에선 `.value` 를 붙이지 않고 그냥 사용하면 된다.

`v-on:` 은 `@` 으로 축약이 가능한데, 다음과 같이 변경 가능하다.

```
<button v-on:click="changeText">클릭하면 글자가 변해!</button>

<button @click="changeText">클릭하면 글자가 변해!</button>
```

- 축약구문을 쓰던, 풀네임을 쓰던 개인의 자유지만,
프로젝트 전체에 축약을 쓸거면 축약만, 풀네임을 쓸거면 풀네임만 쓰도록 한다.
- 화살표 함수도 잘 작동하지만, Vue 공식문서에서 `function` 키워드를 사용하므로 되고
록 `function` 키워드로 사용하자.

이벤트는 `click` 만 있는게 아니다. `keyup` 을 사용해보자.

```
<!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="심우석학생은이서진닭아서매력적이라고생각함">
    <h1>message: {{ hello }}</h1>
    <input type="text" v-model="hello">

    <p>
      <input type="checkbox" id="checkbox" v-model="checked">
      <label for="checkbox">{{checked}}</label>
    </p>

    <p>
      <div>{{ text }}</div>
      <button @click="changeText">클릭하면 글자가 변해!</button>
    </p>
```

```

    <p>
      <input type="text" @keyup="go" />
    </p>
  </div>

  <script>

    const { createApp, ref } = Vue

    app = createApp({
      setup() {
        const hello = ref("")
        const checked = ref(false)
        const text = ref("??")
        const changeText = () => {
          text.value = "짜잔"
        }
        function go(evt) {
          console.log(evt.target.value);
        }
        return {
          hello,
          checked,
          text,
          changeText,
          go
        }
      }
    })

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

</html>

```

```

<script setup>
function go(evt) {
  console.log(evt.target.value);
}
</script>

<template>
  <input type="text" @keyup="go" />
</template>

```

이벤트는 매개변수 `evt` 로 받는다.

dfdtest	d
	df
	dfd
	dfdtd
	dfdte
	dfdtes
	dfdtest
	>

사용자가 입력할 때마다, 콘솔이 찍히는것을 확인할 수 있다.

- 이벤트 목록은 다음에서 확인 가능하다.

https://www.w3schools.com/jsref/dom_obj_event.asp

굉장히 많은 이벤트가 있으며, 처음 개발을 시작할 때에는 클릭만 제대로 익혀도 대부분의 개발은 가능하다.

오늘은 여기까지 보자. 😊