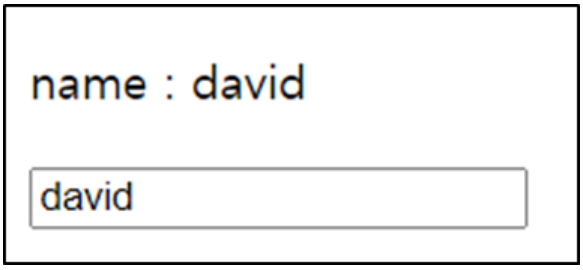
Vue day2 -Vue directive (지시문)

1. `v-model`

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

```
<body>
  <div id="app">
    <p id="name">name : {{ message }}</p>
    <input type="text" v-model="message" />
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

  <script>
    const app = new Vue({
      el: "#app",
      data: {
        message: "",
      },
    });
  </script>
</body>
```

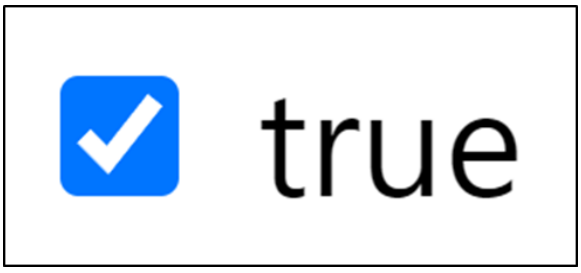
`message` 를 빈 문자열로 `""` 두었고, `<input>` 태그에 `v-model="message"` 를 달았다.



입력할 때마다 메세지가 바뀌는 현상을 경험할 것이다. `v-model` 은 사용자 입력값을 변수로 실시간 저장할 때, 즉 사용자의 입력과 양방향 데이터 바인딩을 공유할 때 사용한다.

`v-model` 은 모든 `<input>` 태그에 사용 가능하므로 체크박스도 활용 가능하다.
예를 보자 (가능한 복붙말고 직접 타이핑 해보자)

```
<body>
  <div id="app">
    <input type="checkbox" id="checkbox" v-model="checked" />
    <label for="checkbox">{{ checked }}</label>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

  <script>
    const app = new Vue({
      el: "#app",
      data: {
        checked: false,
      },
    });
  </script>
</body>
```



클릭할때마다 `true`, `false` 가 변경되는것을 알 수 있다. `v-model` 은 `<input>` 태그 뿐만 아니라 `<select>`, `<textarea>`에 서도 쓰이며, `<input>` 태그가 나오면 무조건 `v-model` 을 사용한다 라고 생각하고 추후 배우게 될 `v-bind` 를 `<input>` 태그에 사용하는 일은 없도록 하자.

2. `v-on`

v-on을 통해서 이벤트를 받아보자. (JS 에서 이벤트리스너랑 비슷한 역할이다)

`v-on` 디렉티브를 사용하여 DOM 이벤트 핸들링이 기능하다. 이벤트 리스너는 HTML element 를 querySelector 로 가져와 이벤트를 붙여줬다면, Vue 는 HTML element 자체에 이벤트를 붙여준다.

```
<body>
    <div id="app">
```

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

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

    <script>
      const app = new Vue({
        el: "#app",
        data: {
          text: "오늘 데자와빵 주인공은?",
        },
        methods: {
          changeText() {
            this.text = "짜잔 김웅서 교육생 축하합니다.";
          },
        },
      });
    </script>
  </body>
```

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

이꼴 `=` 기호 다음에 이벤트 발생 시 실행할 메서드를 콜백으로 달아주면 된다.

클릭하면 `methods` 안에 `changeText` 메서드가 실행되며, `text` 는 `"오늘 데자와빵..?"` 에서 `"짜잔"` 으로 바뀐다.

- `data` 객체 안에 변수에 접근하거나, `methods` 객체 안에 다른 메서드에 접근하려면 `this` 를 사용해야한다.
- `method` 가 아니라 `methods` 로 복수로 써야 작동한다. 자주 실수하는 부분이니까 주의하자!!
- `method` 작성시 `arrow function` 사용 시 사용하지 않으며(this가 window를 가르킴), `method` 는객체 축약 문법으로 작성된다.

```
methods: {
  changeText: function () {
    this.text = "짜잔..";
  },
},

// 아래와 같이 축약
methods: {
  changeText() {
    this.text = "짜잔..";
  },
},
```

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

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

// 아래과 같이 변경 가능
<button @click="changeText">클릭하면 글자가 변해!</button>
```

- 축약구문을 쓰던, 풀네임을 쓰던 개인의 자유지만, 프로젝트 전체에 축약을 쓸거면 축약만, 풀네임을 쓸거면 풀네임만 쓰도록 한다.

이벤트는 `click` 만 있는 것이 아니겠쥬. `keyup` 을 사용해보자.

```
<body>
    <div id="app">
      <input type="text" @keyup="gogo" />
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

    <script>
      const app = new Vue({
        el: "#app",
        data: {},
        methods: {
          gogo(evt) {
            console.log(evt.target.value);
          },
        },
      });
    </script>
  </body>
```

이벤트는 매개변수 `evt` 로 받는다.

사용자가 입력하고 키보드에서 손을 땔 때마다, 콘솔이 찍히는 것을 확인할 수 있다.

- 이벤트 목록은 다음에서 확인 가능하다.

https://www.w3schools.com/js/js_events_examples.asp

3. `v-bind`

태그의 속성(attribute) 를 변수로 지정 시 사용한다.

(혼동주의: 절대 HTML 태그값을 다루는 것이 아니다.)

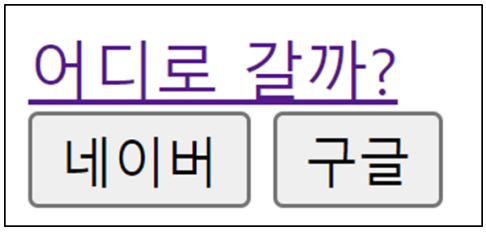
```
<body>
  <div id="app">
    <div>
      <a v-bind:href="URL">어디로 갈까?</a>
    </div>
    <button v-on:click="setURLtoNaver">네이버</button>
```

```
    <button v-on:click="setURLtoGoogle">구글</button>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

  <script>
    const app = new Vue({
      el: "#app",
      data: {
        URL: "",
      },
      methods: {
        setURLtoNaver() {
          this.URL = "https://naver.com";
        },
        setURLtoGoogle() {
          this.URL = "https://google.com";
        },
      },
    });
  </script>
</body>
```

현재 `<a>` 태그를 보면, `v-bind:href="URL"` 이 달려 있는 것을 볼 수 있다. 원래라면 `href="https://naver.com"` 처럼, 고정된 URL 이 들어가지만, 지금 우리는 상황에 따라서 `href` 애트리뷰트의 값을 다르게 만들고 싶다.

이에 `v-bind` 를 사용해 `href` 는 `URL` 변수에 묶이도록 하고, 현재 `URL` 변수는 빈 문자열 `""` 로 지정되어있다.

버튼을 누르면, 해당 `URL` 은 네이버가 될 수도, 구글이 될 수도 있다.



이처럼, `v-bind` 는 태그의 애트리뷰트를 “변수화” 할때 사용한다.

- `v-bind:` 의 축약은 `:` 이다. 즉, 다음과 같이 변경 가능하다.

```
<a v-bind:href="URL">어디로 갈까?</a>

// 아래과 같이 변경 가능
<a :href="URL">어디로 갈까?</a>
```

마찬가지로, 프로젝트 전체에서 `v-on` 이든 `v-bind` 든, 축약을 쓰려면 축약만, 풀네임을 쓸 것이라면풀네임만 사용하도록 하자.

4. `v-if` , `v-else-if` , `v-else`

```
<body>
    <div id="app">
      <div v-if="isActive">active</div>
      <div v-else>rest</div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

    <script>
      const app = new Vue({
        el: "#app",
        data: {
          isActive: true,
        },
      });
    </script>
  </body>
```

해당 변수의 true or false 값을 받은 후, 태그를 보여줄지 말지 결정한다. 결과를 확인해보고, `isActive` 의 `true` 를 `false` 로 바꿔보자.

`v-if` 와 비슷한 문법으로 `v-show`가 있다.

```
<body>
    <div id="app">
      <p v-show="isActive">보이니? 안보이니?</p>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>
    <script>
      const app3 = new Vue({
        el: "#app",
        data: {
          isActive: true,
        },
      });
    </script>
  </body>
```

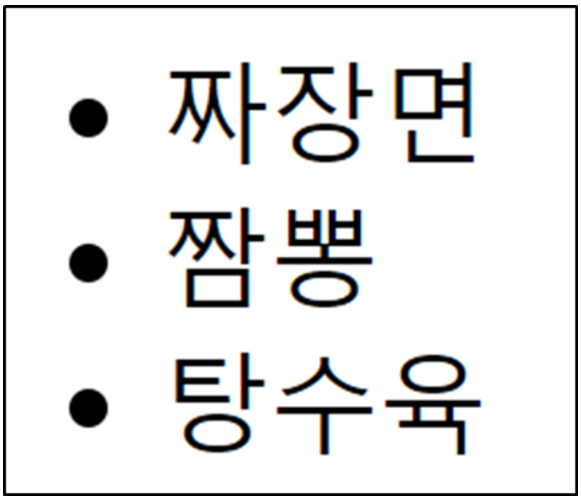
`v-if` 와 `v-show` 둘 다 boolean 값이 변경 될 때 마다 반응을 한다. 하지만 차이점은 `v-if` 는 조건에 맞지 않으면 렌더링 자체를 하지 않는다. 반면에 `v-show` 는 조건과 관계 없이 일단 렌더링을 실시 한 다음에 조건이 거짓이면 CSS display 속성을 `display: none` 으로 바꿔 화면에서 숨겨 버린다. 따라서 화면에 표현 전환(on/off)가 잦다면 `v-show` 를 사용하는 것이 `v-if` 를 사용 하는 것 보다는 랜더링 비용이 적을 것이다.

5. `v-for`

태그의 반복문이다.

```
<body>
  <div id="app">
    <ul>
      <li v-for="(menu, idx) in menus" :key="idx">{{ menu }} </li>
    </ul>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>
  <script>
    const app = new Vue({
      el: "#app",
      data: {
        menus: ["짜장면", "짬뽕", "탕수육"],
      },
    });
  </script>
</body>
```

확인해보면, `menus` 의 각각을 `menu` 와 `idx` 로 받고 `idx` 를 `:key` 로 지정한다.



총 세 개의 태그가 찍힌 것을 확인할 수 있다.

- idx 는 옵션이다. 불필요하다면 제외해도 되나, `:key` 를 반드시 지정 해야 하므로 받았다. 이때

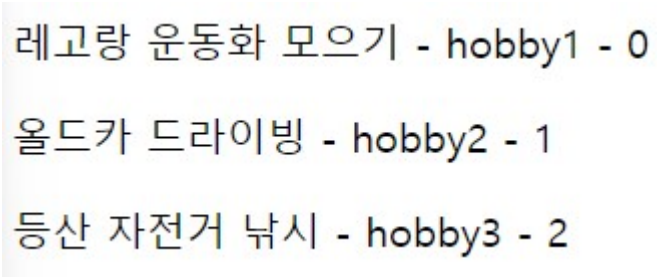
`:key` 는 반드시 중복되지 않는 값으로 지정해야 한다.

```
<body>
  <div id="app">
    <ul>
      <li v-for="menu in menus" :key="menu.id">{{ menu.name }}</li>
    </ul>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>
  <script>
    const app = new Vue({
      el: "#app",
      data: {
        menus: [
          {
            id: 1,
            name: "짜장면",
          },
          {
            id: 2,
            name: "짬뽕",
          },
          {
            id: 3,
            name: "탕수육",
          },
        ],
      },
    });
  </script>
</body>
```

위 예제의 경우, 인덱스를 따로 받진 않았다. 각 객체에 `id` 라는, 중복되지 않는 값이 존재하기 때문이며, 이것을 `key` 로 삼으면 된다. 참고로 `v-for` 에 대해서 조금 더 살펴 보면 `v-for` 를 사용하여 객체의 속성을 반복할 수 있는데 이 때 최대 세 개의 전달 인자를 제공할 수 있다.

```
<body>
  <div id="app">
    <p v-for="(a, b, c) in minho">
      {{ a }} - {{ b }} - {{ c }}
    </p>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

```
  <script>
    const app = new Vue({
      el: 'div',
      data: {
        minho: {
          hobby1: '레고랑 운동화 모으기',
          hobby2: '올드카 드라이빙',
          hobby3: '등산 자전거 낚시'
        }
      }
    })
  </script>
</body>
```



출력결과를 살펴보면 첫 번째 인자는 객체의 `value`, 두 번째 인자는 객체의 `key`, 세 번째 인자로 객체의 `index`라는 것을 확인 할 수 있다. `v-for`를 사용하면서 중요한 점은 `v-for`를 사용한 엘리먼트에는`v-if`를 사용하면 안된다는 점이다.(if 와 for의 우선순위가 존재하여 잘 작동이 안됨) 보통 for 문으로 순회 중 원하는 조건으로 필터링을 하고 싶다면 `.filter()` 함수를 사용한다.

우선순위 A: 필수(오류 예방) | Vue.js (vuejs.org)

6. `computed` (연산 프로퍼티)

```
<body>
    <div id="app">
      <p>{{ msg.split('').reverse().join('') }}</p>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

    <script>
      const app = new Vue({
        el: "#app",
```

```
        data: {
          msg: "Text interpolation",
        },
      });
    </script>
  </body>
```

위 코드를 살펴 보자. “Text interpolation” 문자열을 거꾸로 출력하는 코드이다. 출력 결과는 noitalopretni txeT 가 될 것이다. 거꾸로 문자를 출력 하기 위해서 템플릿 안데 {{ msg.split('').reverse().join('') }} 코드를 직접 기술 하는 것이 편리는 하겠지만 코드를 직접 템플릿에 적다보면 코드가 길어질 경우 가독성과 유지보수가 쉽지 않을 것이다. 그래서 템플릿에서 사용 할 만한 복잡한 로직은 `computed`에서 사용 해 보도록 하자. `computed`를 사용한 예제는 아래와 같다.

```
<body>
      <div id="app">
        <p>{{ reverseMessage }}</p>
      </div>

      <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

      <script>
        const app = new Vue({
          el: "#app",
          data: {
            msg: "Text interpolation",
          },
          computed: {
            reverseMessage: function () {
              return this.msg.split('').reverse().join('');
            },
          },
        });
      </script>
    </body>
```

예시를 보면 코드의 수는 늘어났지만 html의 코드는 깔끔하고 더 명확하게 된 것을 알 수 있다. 뿐만 아니라 computed로 만든 덕분에 다른 곳에서도 자유롭게 문자열을 reverse 하는 메소드를 사용할 수 있게 되어 재사용 성이 좋아 졌다.

우리가 앞에서 배운 `methods`가 있는데 굳이 `computed` 를 따로 또 사용하는 이유는 무엇일까? 물론 `computed` 대신 `methods` 에 문자열을 reverse 하는 코드를 넣어서 같은 결과물을 출력 할 수 있다.

```
methods: {
  reverseMessage: function () {
    return this.message.split('').reverse().join('')
  }
}
```

하지만 `methods` 와 `computed` 는 약간의 차이가 있는데, `computed`는 리액티브(Reactive, 반응적)인 의존관계에 의해 한번 불러진 것들은 캐쉬화 되어진다. 다시말해, `message`가 변하지 않는 한 `reversedMessage`가 여러 번 다시 불려 저도 함수는 다시 실행되는 것이 아닌 이전에 계산되어진 결과를 즉시 돌려준다. 반면에 `methods`는 렌더링(rendering)을 다시 할 때마다 항상 함수를 다시 실행 한다는 차이점이 있다.

`computed` 예시를 참고 삼아 하나만 더 살펴보자.

```
<body>
      <div id="app">
        <p>{{ myname }}</p>
      </div>

      <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

      <script>
        const app = new Vue({
          el: "#app",
          data: {
            first_name: "Minho",
            last_name: "Choi"
          },
          computed: {
            myname: function () {
              return this.first_name+' '+this.last_name
            },
          },
        });
      </script>
    </body>
```

data의 first_name과 last_name을 더해서 full name을 만들어 주는 함수를 선언하는 코드 <선언형 코드> 를 작성해 보았다. 출력 결과는 Minho Choi 가 될 것이다.

7. `watch`

`watch`은 인스턴스에 정의된 data값이 변경이 일어나는지 감시를 하고, 변경이 일어나면 지정된 함수를 실행시키는 기능을 한다.

```
<body>
      <div id="app">
        <span> Value= {{value }}</span> <br />
        <span>changeValue= {{ changeValue }}</span>
        <button @click="change">변경</button>
      </div>
      <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>

      <script>
        const app = new Vue({
          el: "#app",
          data: {
            value: 0,
            changeValue: "",
          },
          watch: {
            value() {
              this.changeValue = this.value;
            },
          },
          methods: {
            change() {
              this.value++;
            },
          },
        });
      </script>
    </body>
```

위에 코드는 value가 변경 시 `watch`문이 value() 함수가 실행되면서 changeValue 값을 변경하고 있다. 즉, `watch`는 Vue 인스턴스의 데이터(data) 변경을 관찰하고 그것에 반응하는 속성이다. 따라서 버튼을 클릭하면 value가 변경되고 `watch`가 실행되는 구조이다.

하지만 Array, Object 와 같은 중첩된 데이터의 변경에 대해서는 `watch` 만을 이용해서는 변경을 감지하지 못한다. 객체나 배열을 `watch`할 때 `deep` 속성을 설정하여 Vue에게 중첩된 데이터를 `watch`해야 한다고 따로 알려줘야 한다는 점을 알아두자.

https://vueframework.com/docs/v3/ko/ko-KR/guide/migration/watch.html#개요

[참고] watch 속성은 언제 많이들 사용할까?

1. 데이터 변경에 대한 응답으로 비동기 작업 수행 할 때 사용 가능하다. 검색어 입력 필드가 있는 경우 검색어가 변경될 때마다 서버에서 데이터를 가져와 화면을 업데이트 할 때 사용 가능 할 것이다.

2. 데이터 변경에 대한 유효성 검사 수행 할 때 사용 가능하다. 사용자가 로그인 폼에 이메일 주소를 입력할 때마다 이메일 주소의 유효성을 검사하여 올바른 형식인지 확인 할 때 사용 가능할 것이다.

3. Vue.js의 컴포넌트 간 데이터 공유 및 동기화 할 때 사용 가능하다. 아직 컴포넌트에 대해서 배우지 않았지만 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달할 때, watch를 사용하여 데이터 변경을 감지하고 하위 컴포넌트를 업데이트할 수 있다 라는 점은 아직 이해가 되지 않겠지만 그렇구나.. 하고 일단은 넘어가자. 차후에 컴포넌트를 배우고 watch 코드를 본다면 바로 이해가 될 것이다.

<끝>