숫자 게임

2020-04-15

이승진

**학습목표**

객체의 속성값 변화 감지

Vue.set 메소드

**목차**

[1. 앱의 속성 2](#_Toc38767397)

[1) 앱의 속성 생성 2](#_Toc38767398)

[2) 앱의 속성값 변경 감지 4](#_Toc38767399)

[3) 객체의 속성값 변화 감지 6](#_Toc38767400)

[4) Vue.set 9](#_Toc38767401)

[2. 동적 서식 적용 10](#_Toc38767402)

[1) v-bind:class 10](#_Toc38767403)

[2) v-bind:style 18](#_Toc38767404)

[3. 숫자 게임 20](#_Toc38767405)

[1) created 메소드 20](#_Toc38767406)

[2) 숫자 뒤섞기 24](#_Toc38767407)

[3) 순서대로 클릭하여 숫자 제거 27](#_Toc38767408)

[4) this.$forceUpdate() 29](#_Toc38767409)

[5) 시간 재기 & toFixed 메소드 31](#_Toc38767410)

[6) 칸의 서식 34](#_Toc38767411)

[4. 과제 39](#_Toc38767412)

[1) color1.html 39](#_Toc38767413)

# 앱의 속성

## 앱의 속성 생성

### property1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 50px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; font-size: 11pt; }      button { padding: 0.5em 2em; }    </style>  </head>  <body>    <div id="app">      <h1>value1: {{ value1 }}</h1>      <h1>value2: {{ value2 }}</h1>      <button type="button" v-on:click="changeValue">값 변경</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value1: 7        },        methods: {          changeValue: function() {            this.value1++;            if (!this.value2) this.value2 = 0;            this.value2++;          }        }      })     </script>  </body>  </html> |

data: {

value1: 7

},

앱에 vulue1 속성이 생성되고 7 값이 대입된다.

<button type="button" v-on:click="changeValue">값 변경</button>

값 변경 버튼을 클릭하면, changeValue 메소드가 호출된다.

this.value1++;

이 메소드에서 value1 속성을 값을 ++ 한다.

value1 속성의 값이 변경되었으므로, {{ value1 }} 부분에 변경된 값이 즉시 출력된다.

if (!this.value2) this.value2 = 0;

앱에 value2 속성이 없다. 즉 this.value2 값은 undefined 이다.

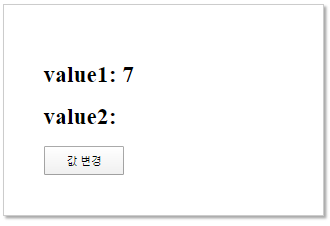
undefined 값은 false로 취급되고, **!**undefined 값은 true로 취급된다.

따라서 위 if 문의 조건식은 true 이다.

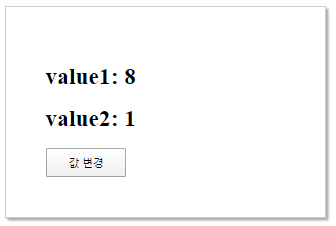
this.value2 = 0;

this는 앱 객체를 가르킨다.

앱에 value2 속성이 없으므로, 새로 만들어지고 0 이 대입된다.



처음 실행하면, value2 속성은 없기 때문에 출력되지 않는다.



값 변경 버튼을 누르면, value2 속성이 생겼기 때문에 출력된다.

## 앱의 속성값 변경 감지

### property2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 50px; margin: 30px auto; max-width: 300px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; font-size: 11pt; }      button { padding: 0.5em 2em; }    </style>  </head>  <body>    <div id="app">      <h1>value1: {{ value1 }}</h1>      <h1>value2: {{ value2 }}</h1>      <button type="button" v-on:click="changeValue1">value1 변경</button>      <button type="button" v-on:click="changeValue2">value2 변경</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value1: 7        },        methods: {          changeValue1: function() {            this.value1++;          },          changeValue2: function() {            if (!this.value2) this.value2 = 0;            this.value2++;          }        }      })     </script>  </body>  </html> |

data: {

  value1: 7

}

value1 속성은 처음부터 앱에, 생성된다.

changeValue2: function() {

  if (!this.value2) this.value2 = 0;

  this.value2++;

}

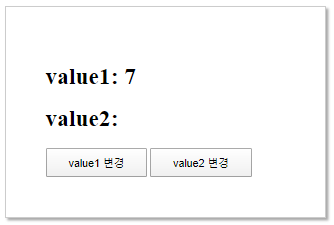
value2 속성은 changeValue2 메소드가 처음 호출될 때, 앱에 생성된다.

### 앱의 속성값 변경 감지

data 부분에 선언되어 생성된 앱의 속성은, 속성값이 변경되면, 그 변경된 값이 즉시 다시 출력된다.

data 부분에 선언되지 않고 나중에 생성된 앱의 속성은,

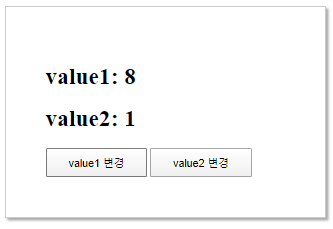
속성값이 변경되아도 다시 출력되지 않는다.



처음 실행하면, value2 속성은 없기 때문에 출력되지 않는다.

value2 변경 버튼을 누르면, changeValue2 메소드가 호출되고, 앱에 value2 속성이 만들어진다.

그렇지만 이렇게 나중에 생성된 속성은 화면에 즉시 출력되지 않는다.



value1 변경 버튼을 누르면, changeValue1 메소드가 호출되고, value1 속성값이 1 증가한다.

value1 속성값이 변경되었으므로 화면이 즉시 다시 출력된다.

value1 변경 버튼을 누르기 전에, 이미 value2 버튼을 눌렀다면, 앱에 value2 속성이 만들어졌을 것이고,

그 value2 속성의 값도 동시에 출력된다.

다시 value2 변경 버튼을 눌러보자, 화면에 변화가 없을 것이다.

value1 변경 버튼을 눌러야 화면이 다시 그려지고,

그동안 화면에 그려지지 않았던 value2 속성의 값도 같이 그려질 것이다.

## 객체의 속성값 변화 감지

### property3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 50px; margin: 30px auto; max-width: 300px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; font-size: 11pt; }      button { padding: 0.5em 2em; }    </style>  </head>  <body>    <div id="app">      <h1>value1: {{ box.value1 }}</h1>      <h1>value2: {{ box.value2 }}</h1>      <button type="button" v-on:click="changeValue1">값1 변경</button>      <button type="button" v-on:click="changeValue2">값2 변경</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          box: { value1: 7 }        },        methods: {          changeValue1: function() {            this.box.value1++;          },          changeValue2: function() {            if (!this.box.value2) this.box.value2 = 0;            this.box.value2++;          }        }      })     </script>  </body>  </html> |

data: {

  box: { value1: 7 }

}

box 속성의 값은 객체이다.

이 객체에는 value1 속성이 들어있다.

box 속성과 value1 속성이 앱이 시작될 때 생성된다.

changeValue2: function() {

if (!this.box.value2) this.box.value2 = 0;

  this.box.value2++;

}

changeValue2 메소드가 처음 호출될 때, this.box 속성이 참조하는 객체에 value2 속성이 생성된다.



if (!this.box.value2) this.box.value2 = 0;

실행결과



### 객체의 속성값 변경 감지

위 그림에서 box 속성의 값은 객체이다.

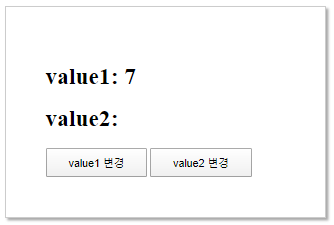
이 객체의 value1 속성은, data 부분에 선언되어 있다.

이렇게 data 부분에 선언된 객체의 속성은, 그 값이 변경되면, 즉시 다시 출력된다.

data 부분에 선언되지 않고 나중에 생성된 속성은,

속성값이 변경되아도 다시 출력되지 않는다.

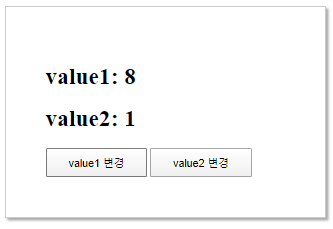
즉 나중에 생성된 value2 속성은, 그 값이 변경되어도 다시 출력되지 않는다.



처음 실행하면, this.box 객체에 value2 속성은 없기 때문에 출력되지 않는다.

value2 변경 버튼을 누르면, changeValue2 메소드가 호출되고, this.box 객체에 value2 속성이 만들어진다.

그렇지만 이렇게 나중에 생성된 속성은 화면에 즉시 출력되지 않는다.



value1 변경 버튼을 누르면, changeValue1 메소드가 호출되고, this.box 객체의 value1 속성값이 1 증가한다.

value1 속성값이 변경되었으므로 화면이 즉시 다시 출력된다.

value1 변경 버튼을 누르기 전에, 이미 value2 버튼을 눌렀다면,

this.box 객체에 value2 속성이 만들어졌을 것이고,

그 value2 속성의 값도 동시에 출력된다.

다시 value2 변경 버튼을 눌러보자, 화면에 변화가 없을 것이다.

value1 변경 버튼을 눌러야 화면이 다시 그려지고,

그동안 화면에 그려지지 않았던 value2 속성의 값도 같이 그려질 것이다.

### 요약

data: { ... }

부분에 선언된 속성의 경우, 객체 내부 속성이라도, 그 값 변경을 Vue 엔진이 감지할 수 있다.

나중에 생성된 속성의 경우, Vue 엔진이 감지할 수 없다.

## Vue.set

### property4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 50px; margin: 30px auto; max-width: 300px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; font-size: 11pt; }      button { padding: 0.5em 2em; }    </style>  </head>  <body>    <div id="app">      <h1>value1: {{ box.value1 }}</h1>      <h1>value2: {{ box.value2 }}</h1>      <button type="button" v-on:click="changeValue1">값1 변경</button>      <button type="button" v-on:click="changeValue2">값2 변경</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          box: { value1: 7 }        },        methods: {          changeValue1: function() {            this.box.value1++;          },          changeValue2: function() {            if (!this.box.value2) Vue.set(this.box, "value2", 0);            this.box.value2++;          }        }      })     </script>  </body>  </html> |

노란색으로 칠한 부분만 앞의 예제와 다르다.

### Vue.set(this.box, "value2", 0)

this.box 속성이 참조하는 객체에, value2 속성이 생성되고 0 이 대입된다.

Vue.set 메소드로 생성되는 속성의 경우, 그 값 변경을 Vue 엔진이 감지할 수 있다.

그래서 value2 속성값이 변경되면, 화면이 즉시 다시 그려진다.

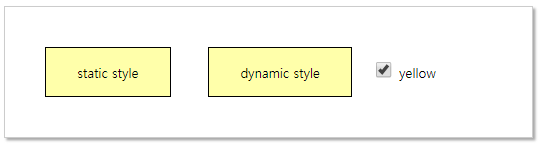
# 동적 서식 적용

## v-bind:class

### toggle1.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }      .yellow { background-color: #ffa; }    </style>  </head>  <body>    <div id="app">      <span class="yellow"> static style </span>      <span v-bind:class="{ yellow: value }"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        yellow      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        }      })     </script>  </body>  </html> |





data: {

  value: false

}

앱에 value 속성이 생성된다. 이 속성의 초기 값은 false 이다.

<input type="checkbox" v-model="value" />

이 체크 박스와 앱의 value 속성이 연결된다.

체크 박스가 체크되면 value 속성값이 true가 되고,

체크가 꺼지면 value 속성값이false가 된다.

.yellow { background-color: #ffa; }

class="yellow" 속성을 가진 태그에 이 서식이 적용된다.

<span class="yellow"> static style </span>

이 태그에 .yellow 서식이 적용되어, 노란색 배경색이 된다.

class 속성은 표준 html 속성이다.

이 속성은 위와 같이 태그에 서식을 적용하기 위한 용도로 사용된다.

<span v-bind:class="{ yellow: value }"> dynamic style </span>

class 속성 앞에 v-bind: 접두어가 붙었으므로, 이 속성의 값은 javascript 표현식으로 해석된다.

javascript 표현식으로 해석하면, { yellow: value } 이것은 객체이다.

여기서 yellow 부분은 객체의 속성명이고, value 부분은 yellow 속성의 값이다.

yellow 부분은 속성명이므로 문자열로 취급된다. ("yellow" 문자열)

value 부분은 속성값이므로 javascript 표현식으로 취급된다.

value 가 앱의 속성이고, 값은 false 이다.

value 값이 false 이면, 위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class=""> dynamic style </span>

value 값이 true 이면, 위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class="yellow"> dynamic style </span>

이 태그에 .yellow 서식이 적용된다.

### toggle2.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 500px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }      .yellow { background-color: #ffa; }      .green { background-color: #bfb; }    </style>  </head>  <body>    <div id="app">      <span class="yellow"> static style1 </span>      <span class="green"> static style2 </span>      <span v-bind:class="{ yellow: value, green: !value }"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        toggle      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        }      })     </script>  </body>  </html> |

.yellow { background-color: #ffa; }

class="yellow" 속성을 가진 태그에 이 서식이 적용된다.

.green { background-color: #bfb; }

class="green" 속성을 가진 태그에 이 서식이 적용된다.

<span class="yellow"> static style </span>

<span class="green"> static style </span>

위 두 span 태그에는 각각 .yellow 서식과 .green 서식이 적용되어 다음과 같이 그려진다.



<span v-bind:class="{ yellow: value, green: !value }"> dynamic style </span>

class 속성 앞에 v-bind: 접두어가 붙었으므로, 이 속성의 값은 javascript 표현식으로 해석된다.

javascript 표현식으로 해석하면, { yellow: value, green: !value } 이것은 객체이다.

여기서 yellow, green 부분은 객체의 속성명이고, value, !value 부분은 각각 yellow, green 속성의 값이다.

yellow, green 부분은 속성명이므로 문자열로 취급된다. ("yellow", "green" 문자열)

value, !value 부분은 속성값이므로 javascript 표현식으로 취급된다.

value 가 앱의 속성이고, value 값은 false, !value 값은 true 이다.

value 값이 false 이면 !value 부분이 true이므로,

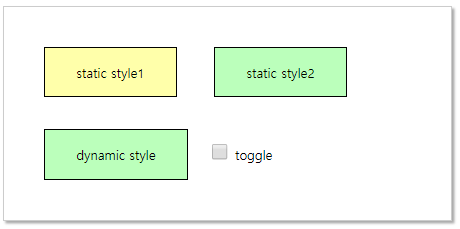
위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

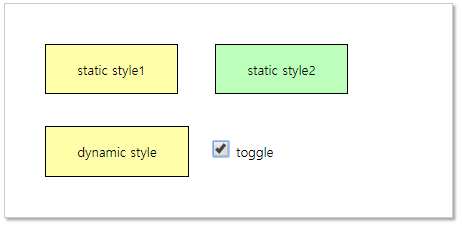
<span class="green"> dynamic style </span>

value 값이 true 이면, 위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class="yellow"> dynamic style </span>

이 태그에 .yellow 서식이 적용된다.





### toggle3.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }      .yellow { background-color: #ffa; }      .yellow::before { content: "(노랑)"}      .green { background-color: #bfb; }      .green::after { content: "(녹색)"}    </style>  </head>  <body>    <div id="app">      <span class="yellow"> static style1 </span>      <span class="green"> static style2 </span>      <span v-bind:class="{ yellow: value, green: !value }"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        toggle      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        }      })     </script>  </body>  </html> |

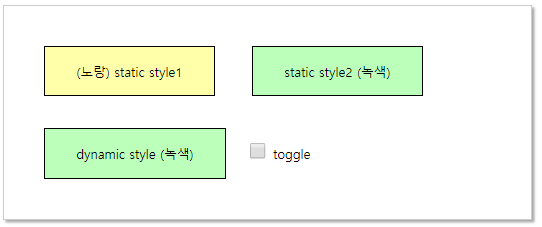
노란색으로 칠한 부분만 앞의 예제와 다르다.

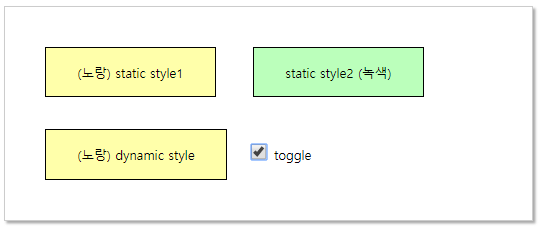
.yellow::before { content: "(노랑)"}

class="yellow" 속성을 가진 태그의 내용 앞 부분에 "(노랑)" 문자열이 추가된다.

.green::after { content: "(녹색)"}

class="green" 속성을 가진 태그의 내용 뒷 부분에 "(녹색)" 문자열이 추가된다.





### toggle4.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }      .yellow { background-color: #ffa; }      .yellow::before { content: "(노랑)"}      .green { background-color: #bfb; }      .green::after { content: "(녹색)"}    </style>  </head>  <body>    <div id="app">      <span class="yellow"> static style1 </span>      <span class="green"> static style2 </span>      <span v-bind:class="value ? 'yellow' : 'green'"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        toggle      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        }      })     </script>  </body>  </html> |

<span v-bind:class="value ? 'yellow' : 'green'"> dynamic style </span>

class 속성 앞에 v-bind: 접두어가 붙었으므로, 이 속성의 값은 javascript 표현식으로 해석된다.

javascript 표현식으로 해석하면, value ? 'yellow' : 'green' 이것은 3항 연산자 표현식이다.

value 값이 true 이면, 이 표현식의 값은 'yellow' 문자열이고,

value 값이 false 이면, 이 표현식의 값은 'green' 문자열이고,

value 값이 false 이면, 위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class='green'> dynamic style </span>

value 값이 true 이면, 위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class='yellow'> dynamic style </span>

이 태그에 .yellow 서식이 적용된다.

html 태그나 javascript 언어의 문자열을 묶을 때,

"문자열" 쌍따옴표와 '문자열' 단따옴표,

둘 다 가능하다.

### toggle5.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }      .yellow { background-color: #ffa; }      .yellow::before { content: "(노랑)"}      .green { background-color: #bfb; }      .green::after { content: "(녹색)"}    </style>  </head>  <body>    <div id="app">      <span class="yellow"> static style1 </span>      <span class="green"> static style2 </span>      <span v-bind:class="getCssClass()"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        toggle      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        },        methods: {          getCssClass() {            return this.value ? "yellow" : "green";          }        }      })     </script>  </body>  </html> |

<span v-bind:class="getCssClass()"> dynamic style </span>

class 속성 앞에 v-bind: 접두어가 붙었으므로, 이 속성의 값은 javascript 표현식으로 해석된다.

javascript 표현식으로 해석하면, getCssClass() 이것은 메소드 호출이다.

이 메소드의 리턴값이 표현식 값이된다.

value 값이 false 이면, getCssClass() 메소드는 "green" 문자열을 리턴하고,

위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class="green"> dynamic style </span>

value 값이 true 이면, getCssClass() 메소드는 "yellow" 문자열을 리턴하고,

위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span class="yellow"> dynamic style </span>

## v-bind:style

### toggle6.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }      span { display: inline-block; margin: 20px; padding: 20px 40px;        border: 1px solid black; }      input[type=checkbox] { width: 20px; height: 20px; }    </style>  </head>  <body>    <div id="app">      <span style="background-color: #ffa;"> static style1 </span>      <span style="background-color: #bfb;"> static style2 </span>      <span v-bind:style="{ backgroundColor: getColor() }"> dynamic style </span>      <label>        <input type="checkbox" v-model="value" />        toggle      </label>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          value: false        },        methods: {          getColor() {            return this.value ? "#ffa" : "#bfb";          }        }      })     </script>  </body>  </html> |

<span style="background-color: #ffa;"> static style1 </span>

style 속성은 표준 html 속성이다.

CSS 서식을 위와 같이 태그에 직접 구현할 수도 있다.

이렇게 태그에 구현된 서식을 inline style 이라고 부른다.

이 태그의 배경색은 #ffa 이 된다.

<span v-bind:style="{ backgroundColor: getColor() }"> dynamic style </span>

style 속성 앞에 v-bind: 접두어가 붙었으므로, 이 속성의 값은 javascript 표현식으로 해석된다.

{ backgroundColor: getColor() } 이것은 javascript 객체이다.

backgroundColor 부분은 속성명이고, getColor() 부분은 속성값이다.

즉 getColor() 메소드의 리턴값이 backgroundColor 속성값이 된다.

CSS 속성 이름에서 두 단어는 - 문자로 연결된다. (예: background-color)

javascript 속성, 변수, 함수 이름에 - 문자를 사용할 수 없다.

그래서 background-color 이름 대신 backgroundColr 형태의 이름을 사용해야 한다.

value 값이 false 이면, getColor() 메소드는 "#bfb" 문자열을 리턴하고,

위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span style="background-color: #bfb;"> dynamic style </span>

value 값이 true 이면, getCssClass() 메소드는 "#ffa" 문자열을 리턴하고,

위 v-bind 부분을 Vue 엔진이 실행한 결과, 태그는 다음과 같은 형태가 된다.

<span style="background-color: #ffa;"> dynamic style </span>

# 숫자 게임

## created 메소드

### game1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }    </style>  </head>  <body>    <div id="app">      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2">            {{ value }}          </td>        </tr>      </table>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: []        },        created() {          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = r \* this.size + c + 1;          }        }      })     </script>  </body>  </html> |

Vue 앱을 생성하는 생성자 파라미터 객체의 구조를 요약하면 다음과 같다.

{

      el: '#app',

      data: { ... },

      created() { .... }

}

위 객체의 구조는 아래와 동일하다.

{

      el: '#app',

      data: { ... },

      created: function() { .... }

}

### created 메소드

이 created 메소드는,

Vue 앱 객체가 생성된 직후이고, 아직 태그들이 생성되기 전에 호출된다.

이렇게 Vue 앱 객체의 어떤 특정 시점에 호출되는 메소드들을 lifetime hook 이라고 한다.

지난 시간에 mounted() lifetime hook 메소드를 배웠다.

실행 순서는 다음과 같다.

(1) 앱 객체가 생성되고

(2) created() 메소드가 호출되고

(3) 태그들이 생성되고

(4) mounted() 메소드가 호출되고

(5) 화면이 그려진다.

앱이 시작할 때 최초로 한 번 실행해야 할 작업들 중에서

태그들이 생성되기 전에 해야할 작업은 created 메소드에 구현하고

태그들이 생성된 후에 해야할 작업은 mounted 메소드에 구현한다.

data: {

size: 6,

  matrix: []

},

matrix 속성값은 2차원 배열이어야 한다.

size 행, size 열 크기로 2차원 배열을 생성하는 작업을 최초로 한 번 실행해야 한다.

중첩된 v-for 구현에 의해서,

matrix 2차원 배열의 크기로 tr 태그와 td 태그가 생성되어야 한다.

따라서 태그들이 생성되기 전에, matrix 2차원 배열을 만들어야 한다.

즉 created 메소드에서, matrix 2차원 배열을 만들어야 한다.

### 2차원 배열 만들기

for (let r = 0; r < this.size; ++r) {

  this.matrix[r] = [];

  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = r \* this.size + c + 1;

}

최초에 this.matrix 배열은 [] 형태이다.

### 최초 반복

let r = 0;

this.matrix[r] = [];

이 코드를 실행하면 this.matrix 배열은 [[]] 형태이다.

즉 1행, 0열 2차원 배열이다.

r = 0; 인 상황에서

  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = r \* this.size + c + 1;

위 반복문을 실행한 후 this.matrix 배열은 [[1, 2, 3, 4, 5, 6]] 형태이다.

즉 1행, 6열 2차원 배열이다.

### 2 번째 반복

++r

this.matrix[r] = [];

이 코드를 실행하면 this.matrix 배열은 다음과 같다.

[[1, 2, 3, 4, 5, 6],

[]

]

2행이고, 첫째 행은 6열, 둘째 행은 0열 이다.

r = 1; 인 상황에서

  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = r \* this.size + c + 1;

위 반복문을 실행한 후 this.matrix 배열은 다음과 같다.

[[1, 2, 3, 4, 5, 6],

[7, 8, 9, 10, 11, 12]

]

2행 6열 배열이다.

### 3 번째 반복

++r

this.matrix[r] = [];

이 코드를 실행하면 this.matrix 배열은 다음과 같다.

[[1, 2, 3, 4, 5, 6],

[7, 8, 9, 10, 11, 12],

[]

]

3행이고, 첫째 둘째 행은 6열, 셋째 행은 0열 이다.

r = 2; 인 상황에서

  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = r \* this.size + c + 1;

위 반복문을 실행한 후 this.matrix 배열은 다음과 같다.

[[1, 2, 3, 4, 5, 6],

[7, 8, 9, 10, 11, 12],

[13, 14, 15, 16, 17, 18]

]

3행 6열 배열이다.

<table>

<tr v-for="(row, index1) in matrix" v-bind:key="index1">

   <td v-for="(value, index2) in row" v-bind:key="index2">

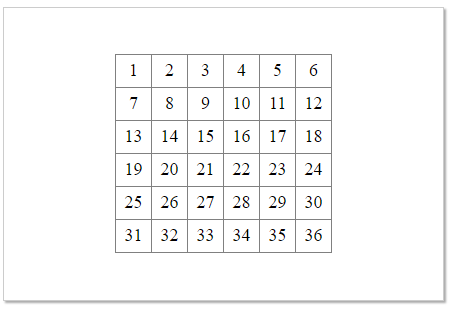
     {{ value }}

    </td>

  </tr>

</table>

matrix 2차원 배열을 table 태그로 출력한다.



## 숫자 뒤섞기

### game2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }    </style>  </head>  <body>    <div id="app">      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2">            {{ value }}          </td>        </tr>      </table>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: []        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

노란색으로 칠한 부분만 앞의 예제와 다르다.

this.lastNumber = this.size \* this.size;

2차원 배열에 채워야할 가장 큰 수를 계산한다.

size 값이 6 이미로, lastNumber는 36 이다.

let a = [];

for (let i = 0; i < this.lastNumber; ++i)

a[i] = i + 1;

lastNumber 크기의 1차원 배열을 생성한다.

function shuffleArray(a) {

  for (let i = 0; i < a.length; ++i) {

    let j = Math.floor(Math.random() \* a.length);

    [a[i], a[j]] = [a[j], a[i]];

  }

}

1차원 배열에 원소들을 뒤섞는다.

for (let r = 0; r < this.size; ++r) {

  this.matrix[r] = [];

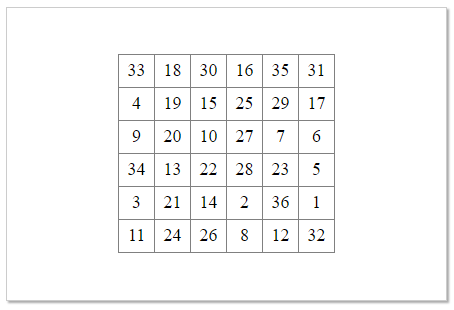
  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = a.pop();

}

size 행, size 형 2차원 배열을 생성한다.

a 배열에 들어있는 랜덤한 값들을 2차원 배열에 채운다.



### 분석

for (let r = 0; r < this.size; ++r) {

  this.matrix[r] = [];

  for (let c = 0; c < this.size; ++c)

    this.matrix[r][c] = Math.floor(Math.random() \* this.lastNumber);

}

값이 뒤섞인 2차원 배열을 만드는 코드를 이렇게 구현하면 어떤 문제가 있을까?

소스코드를 위와 같이 수정하고 실행해 보자.

## 순서대로 클릭하여 숫자 제거

### game3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }    </style>  </head>  <body>    <div id="app">      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2"              v-on:click="onClick(index1, index2)">            {{ value }}          </td>        </tr>      </table>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: [],          currentNumber: 1        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        },        methods: {          onClick: function(r, c) {            if (this.matrix[r][c] == this.currentNumber) {              this.matrix[r][c] = "";              this.currentNumber++;            }          }        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

1부터 순서대로 클릭하여 숫자를 제거하는 기능을 구현하자.

currentNumber: 1

currentNumber 속성은 제거할 현재 번호이다.

v-on:click="onClick(index1, index2)"

td 태그를 클릭하면, onClick 메소드가 호출된다.

index1 행, index2 열 값이 파라미터로 전달된다.

onClick: function(r, c) {

  if (this.matrix[r][c] == this.currentNumber) {

    this.matrix[r][c] = "";

    this.currentNumber++;

  }

}

클릭된 칸의 숫가자 currentNumber와 같으면,

그 칸의 숫자를 제거하고, currentNumber 속성을 1 증가한다.

this.matrix[r][c] = "";

현재 칸의 값이 "" 로 바뀌면 화면이 즉시 다시 그려질까?

2차원 배열의 원소의 값이 바뀌면 화면이 즉시 다시 그려지나?

그렇지 않다.

Vue 엔진은 배열의 원소의 값이 위와 같이 바뀌는 것을 감지하지 못한다.

## this.$forceUpdate()

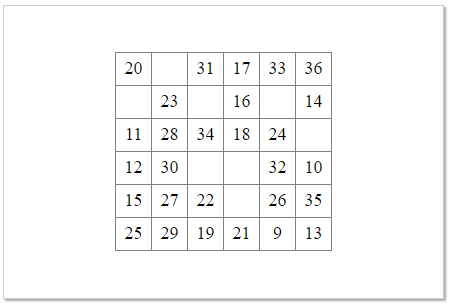
### game4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }    </style>  </head>  <body>    <div id="app">      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2"  v-on:click="onClick(index1, index2)">            {{ value }}          </td>        </tr>      </table>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: [],          currentNumber: 1        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        },        methods: {          onClick: function(r, c) {            if (this.matrix[r][c] == this.currentNumber) {              this.matrix[r][c] = "";              this.currentNumber++;              this.$forceUpdate();            }          }        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

클릭된 칸의 값이 "" 이 되면,

this.$forceUpdate(); 메소드가 호출되고,

그래서 화면이 다시 그려진다.



## 시간 재기 & toFixed 메소드

### game5.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }    </style>  </head>  <body>    <div id="app">      <h2>{{ sec }} 초 {{ message }}</h2>      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2"  v-on:click="onClick(index1, index2)">            {{ value }}          </td>        </tr>      </table>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: [],          currentNumber: 1,          sec: 0,          message: ""        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        },        methods: {          onClick: function(r, c) {            if (this.currentNumber == 1) this.initTimer();            if (this.matrix[r][c] == this.currentNumber) {              this.matrix[r][c] = "";              this.currentNumber++;              if (this.currentNumber > this.lastNumber) {                clearInterval(this.timerId);                this.message = "(성공)";              }            }          },          initTimer() {            this.startTime = new Date();            this.timerId = setInterval(() => {              this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);            }, 100);          }        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

sec: 0,

게임 플레이 시간(초)를 기록할 속성이다.

message: ""

게임 성공 메시지를 출력하기 위한 속성이다.

<h2>{{ sec }} 초 {{ message }}</h2>

게임 플레이 시간(초)와 성공 메시지가 여기에 출력된다.

if (this.currentNumber == 1) this.initTimer();

1번 칸을 클릭하면, 게임 플레이 시간을 계산하기 시작한다.

initTimer() {

  this.startTime = new Date();

  this.timerId = setInterval(() => {

    this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);

  }, 100);

}

this.startTime = new Date();

게임 시작 시각을 기록한다.

### toFixed 메소드

this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);

현재 시각에서 게임 시작 시간을 빼면, 그 시간차이 밀리초가 된다.

밀리초를 1000으로 나누면 초 값이 된다.

숫자에 .toFixed(1) 메소드를 호출하면, 그 숫자를 소숫점 1째 자리로 반올림한 새 숫자를 리턴한다.

이 숫자를 this.sec 속성에 대입한다.

이 값이 {{ sec }} 초 부분에 즉시 출력된다.

0.1 초 간격으로 sec 속성값이 변경되고, 이 변경 때문에 화면이 다시 그려지므로,

this.$forceUpdate() 메소드 호출은 필요 없어졌다.

  this.timerId = setInterval(() => {

    this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);

  }, 100);

노란색으로 칠한 부분이 화살표 함수 문법으로 구현된 콜백 함수이다.

이 콜백함수가 0.1초 간격으로 호출되도록 등록한다.

이 등록 id를 timerId 속성에 대입한다.

if (this.currentNumber > this.lastNumber) {

  clearInterval(this.timerId);

  this.message = "(성공)";

}

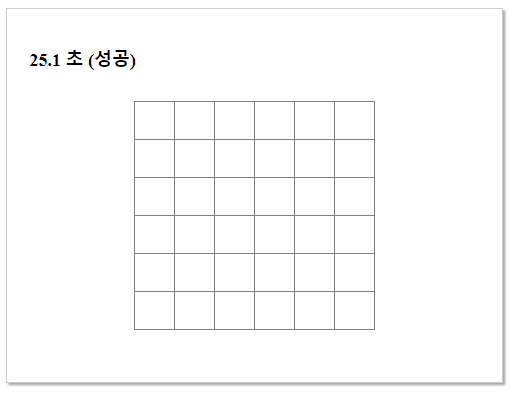
클릭해야할 숫자에 해당하는 currentNumber 값이,

최대 숫자 lastNumber 보다 크면, 모든 숫자를 끌릭한 것이고, 게임 끝이다.

setInterval 등록을 취소한다. 이제 콜백함수는 호출되지 않게 된다.

messaage 속성에 "(성공)" 문자열을 대입한다.

이 값이 {{ message }} 부분에 즉시 출력된다.



## 칸의 서식

### game6.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }      button { padding: 0.5em 2em; }      td.yellow { background-color: #ffa; }      td.green { background-color: #bfb; }      td.blue  { background-color: #8df; }      td.red { background-color: #fbb; }    </style>  </head>  <body>    <div id="app">      <h2>{{ sec }} 초 {{ message }}</h2>      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2"  v-on:click="onClick(index1, index2)"              v-bind:class="cssClass(value)">            {{ value }}          </td>        </tr>      </table>      <button type="button" onclick="location.reload()" v-show="message">재도전</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: [],          currentNumber: 1,          sec: 0,          message: ""        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        },        methods: {          onClick: function(r, c) {            if (this.currentNumber == 1) this.initTimer();            if (this.matrix[r][c] == this.currentNumber) {              this.matrix[r][c] = "";              this.currentNumber++;              if (this.currentNumber > this.lastNumber) {                clearInterval(this.timerId);                this.message = "(성공)";              }            }          },          initTimer() {            this.startTime = new Date();            this.timerId = setInterval(() => {              this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);            }, 100);          },          cssClass(value) {            let digit = Math.floor(value / 10);            return { yellow: digit==0, green: digit==1, blue: digit==2, red: digit==3 };          }        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

td.yellow { background-color: #ffa; }

td.green { background-color: #bfb; }

td.blue  { background-color: #8df; }

td.red { background-color: #fbb; }

td 태그의 class 속성값에 따라 배경색을 적용하가 위한 서식이다.

v-bind:class="cssClass(value)"

cssClass(value) 메소드 호출의 리턴 값에 의해서, td 태그의 class 속성값이 결정된다.

cssClass 메소드의 파라미터 값으로 전달되는 것은, 그 칸(td)의 숫자이다.

cssClass(value) {

  let digit = Math.floor(value / 10);

  return { yellow: digit==0, green: digit==1, blue: digit == 2, red: digit == 3 };

}

let digit = Math.floor(value / 10);

칸의 숫자(value)를 10으로 나눈 몫에서 소수점 아래를 버린 정수를 digit 변수에 대입한다.

이 값은 0, 1, 2, 3 중에 하나이다.

return { yellow: digit==0, green: digit==1, blue: digit == 2, red: digit == 3 };

digit 값이 0 이면, { yellow: **true**, green: false, blue: false, red: false } 객체가 리턴된다.

digit 값이 1 이면, { yellow: false, green: **true**, blue: false, red: false } 객체가 리턴된다.

digit 값이 3 이면, { yellow: false, green: false, blue: **true**, red: false } 객체가 리턴된다.

digit 값이 4 이면, { yellow: false, green: false, blue: false, red: **true** } 객체가 리턴된다.

<button type="button" onclick="location.reload()" v-show="message">재도전</button>

v-show="message"

message 속성값이 빈 문자열인 경우에는 이 버튼이 보이지 않는다.

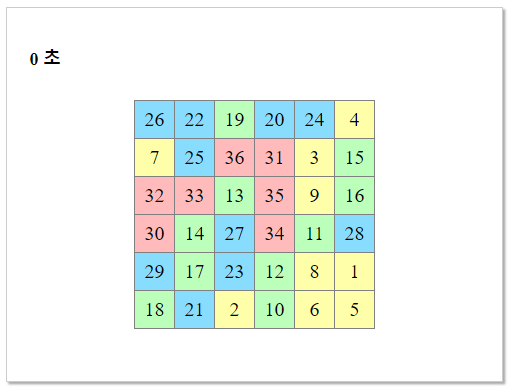
v-show 지시자의 값은 boolean 타입의 javascrpt 표현식으로 해석된다.

javascript 언어에서 0, "", null, undefined, NaN 값들은, 조건식에서 false 취급된다.

onclick="location.reload()"

이 태그가 클랙되면 웹브라우저 창이 새로고침 된다.

이 부분은 Vue 기능이 아니고, javascript 기능이다.



### game7.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      table { border-collapse: collapse; margin: 40px auto; }      td { width: 50px; height: 50px; border: 1px solid gray; font-size: 20pt;        text-align: center; cursor: pointer; }      button { padding: 0.5em 2em; }    </style>  </head>  <body>    <div id="app">      <h2>{{ sec }} 초 {{ message }}</h2>      <table>        <tr v-for="(row, index1) in matrix" v-bind:key="index1">          <td v-for="(value, index2) in row" v-bind:key="index2"  v-on:click="onClick(index1, index2)"              v-bind:style="{ backgroundColor: color(value) }">            {{ value }}          </td>        </tr>      </table>      <button type="button" onclick="location.reload()" v-show="message">재도전</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          size: 6,          matrix: [],          currentNumber: 1,          sec: 0,          message: ""        },        created: function() {          this.lastNumber = this.size \* this.size;          let a = [];          for (let i = 0; i < this.lastNumber; ++i)            a[i] = i + 1;          shuffleArray(a);          for (let r = 0; r < this.size; ++r) {            this.matrix[r] = [];            for (let c = 0; c < this.size; ++c)              this.matrix[r][c] = a.pop();          }        },        methods: {          onClick: function(r, c) {            if (this.currentNumber == 1) this.initTimer();            if (this.matrix[r][c] == this.currentNumber) {              this.matrix[r][c] = "";              this.currentNumber++;              if (this.currentNumber > this.lastNumber) {                clearInterval(this.timerId);                this.message = "(성공)";              }            }          },          initTimer() {            this.startTime = new Date();            this.timerId = setInterval(() => {              this.sec = ((new Date() - this.startTime) / 1000).toFixed(1);            }, 100);          },          color(value) {            const a = ["#ffa", "#bfb", "#8df", "#fbb"];            return a[Math.floor(value / 10)];          },        }      })      function shuffleArray(a) {        for (let i = 0; i < a.length; ++i) {          let j = Math.floor(Math.random() \* a.length);          [a[i], a[j]] = [a[j], a[i]];        }      }     </script>  </body>  </html> |

v-bind:style="{ backgroundColor: color(value) }"

td 태그의 background-color 값은 color(value) 메소드의 리턴값이된다.

color(value) {

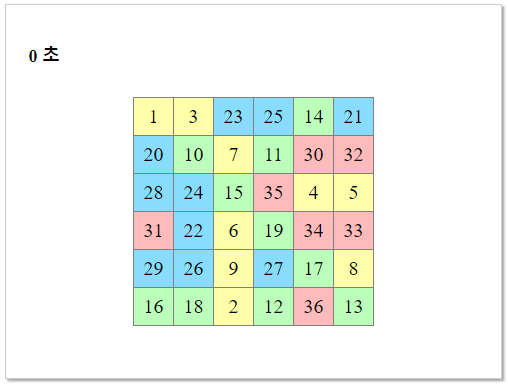
  const a = ["#ffa", "#bfb", "#8df", "#fbb"];

  return a[Math.floor(value / 10)];

},

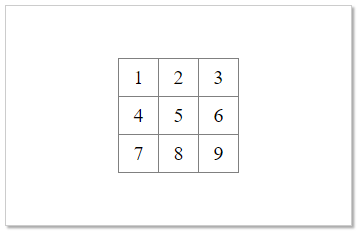
칸의 숫자(value)의 10의 자리가 0이면 "#ffa" 를 리턴하고,

1 이면 "#bfb"를, 2 이면 "#8df"를, 3 이면 "#fbb"를 리턴한다.



# 과제

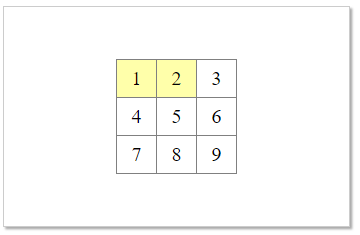
## color1.html



실행하면 위와 같이 출력되어야 한다.

구현 방법은 자유이다.

2차원 배열을 만들어서 출력해도 좋고, 그냥 table, tr, td 태그를 전부 만들어도 좋다.



칸을 클릭하면, 클릭된 칸의 배경색이 노란색으로 바뀌어야 한다.

어떤 방법으로 구현해도 좋다.

색만 바뀌면 된다.