시계

2020-04-15

이승진

**학습목표**

타이머 구현

시간 계산

간단한 그래픽 구현

**목차**

[1. 디지탈 시계 2](#_Toc38204798)

[1) 현재 시각 출력 2](#_Toc38204799)

[2) mounted, setInterval 3](#_Toc38204800)

[3) time format 5](#_Toc38204801)

[2. 스톱 워치 10](#_Toc38204802)

[1) 밀리초 계산 10](#_Toc38204803)

[2) v-bind:disabled 13](#_Toc38204804)

[3. 타이머 21](#_Toc38204805)

[1) 밀리초 계산 21](#_Toc38204806)

[2) mp3 사운드 24](#_Toc38204807)

[4. canvas 그래픽 29](#_Toc38204808)

[1) 선과 사각형 29](#_Toc38204809)

[2) 경로 그리기 32](#_Toc38204810)

[3) 원둘레 그리기 34](#_Toc38204811)

[4) 그라디언트 칠하기 36](#_Toc38204812)

[5) 문자열 그리기 38](#_Toc38204813)

[5. 아날로그 시계 39](#_Toc38204814)

[1) 배경 이미지 39](#_Toc38204815)

[2) 현재 시각 그리기 41](#_Toc38204816)

[3) 시계 45](#_Toc38204817)

# 디지탈 시계

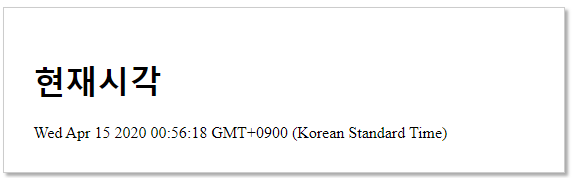
## 현재 시각 출력

### clock1.html

|  |
| --- |
| <html>  <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; }    </style>  </head>  <body>    <div id="app">      <h1>현재시각</h1>      <div id="time">{{ time }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: new Date()        }      })     </script>  </body>  </html> |
|  |

new Date()

Date 객체가 생성된다. 이 객체에는 현재 시각이 들어있다.



## mounted, setInterval

### clock2.html

|  |
| --- |
| <html>  <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; }    </style>  </head>  <body>    <div id="app">      <h1>현재시각</h1>      <div id="time">{{ time }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: new Date()        },      })     </script>  </body>  </html> |

Vue 생성자 파라미터 객체를 요약하면 다음과 같다.

{

el: '#app',

  data: { 생략 },

  mounted: function() { 생략 }

}

#### el

el 속성값은 Vue의 기능이 적용될 태그를 지정한다.

#### data

data 속성값은 Vue 앱의 속성을 정의한다.

#### mounted

mounted 속성값은 함수이다.

mounted 메소드는, Vue 앱 객체가 생성된 후이고, 아직 화면이 그려지기 전에 호출된다.

앱이 시작될 때 해야할 일을 mounted 메소드에 구현하자.

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

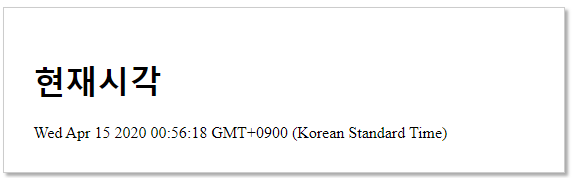
#### setInterval

let callback = () => this.time = new Date();

setInterval(callback, 1000);

1초 마다 한 번씩 앱의 time 속성에 현재 시각 객체를 대입한다.

'함수' 강의노트의 setInterval 함수 설명을 참고하라.



화면에 표시된 현재 시각이 1초마다 한 번씩 변경된다.

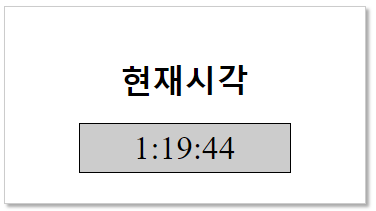
## time format

### clock3.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }    </style>  </head>  <body>    <div id="app">      <h1>현재시각</h1>      <div id="time">{{ time }}</div>    </div>    <script type="text/javascript">      var app = new Vue({         el: '#app',        data: {          time: ''        },        mounted: function() {          let callback = () => this.time = formatTime(new Date());          setInterval(callback, 1000);        }      })       </script>  </body>  </html> |

#### formatTime 함수

파라미터로 주어진 시각으로부터 "시:분:초" 형태의 문자열을 만들어 리턴한다.



1초마나 한 번씩 화면에 "시:분:초" 형태로 현재 시각이 출력된다.

그런데 예를 들어 9시 5분 8초의 경우에 "9:5:8" 형태로 출력된다.

아쉽게도 javascript 에는 날짜 시각 포멧을 지정하는 표준 함수나 메소드가 없다.

직업 구현하던가, 오픈 소스 라이브러리를 구해야 한다.

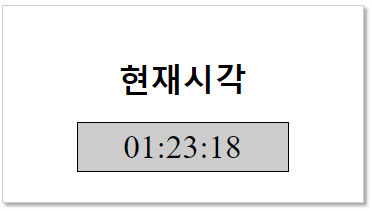
### clock4.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }    </style>  </head>  <body>    <div id="app">      <h1>현재시각</h1>      <div id="time">{{ time }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: ''        },        mounted: function() {          let callback = () => this.time = formatTime(new Date());          setInterval(callback, 1000);        }      })      function formatTime(date) {        let h = date.getHours(), m = date.getMinutes(), s = date.getSeconds();        if (h < 10) h = "0" + h;        if (m < 10) m = "0" + m;        if (s < 10) s = "0" + s;        return h + ":" + m + ":" + s;      }     </script>  </body>  </html> |

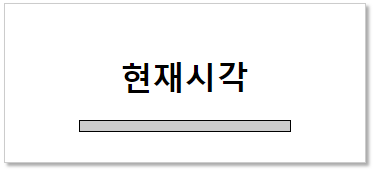
현재 시(hour)를 두 자리로 출력하기 위해서 아래 코드를 추가하였다.

if (h < 10) h = "0" + h;

h 값이 10 미만일 경우에, 앞에 "0"을 붙인다.



그런데 앱이 시작한 처음 화면은 다음과 같다.



앱이 시작되고 약 1초 동안 위와 같은 화면이 보인다.

앱의 time 속성의 초기 값이 '' 이기 때문이다.

1초 후에 setInterval 함수에 의해서 콜백 함수가 호출되면

앱의 time 속성에 현재 시각이 대입되고, 화면에 현재 시각이 출력된다.

이 문제를 해결하자.

### clock5.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }    </style>  </head>  <body>    <div id="app">      <h1>현재시각</h1>      <div id="time">{{ time }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: ''        },        mounted: function() {          let callback = () => this.time = formatTime(new Date());          callback();          setInterval(callback, 1000);        }      })      function formatTime(date) {        let h = date.getHours(), m = date.getMinutes(), s = date.getSeconds();        if (h < 10) h = "0" + h;        if (m < 10) m = "0" + m;        if (s < 10) s = "0" + s;        return h + ":" + m + ":" + s;      }     </script>  </body>  </html> |

mounted 함수는 앱이 생성된 후, 화면이 그려지기 전에 호출된다.

mounted 함수에서 콜백 함수를 호출하여, 앱의 time 속성에 현재 시각을 대입한다.

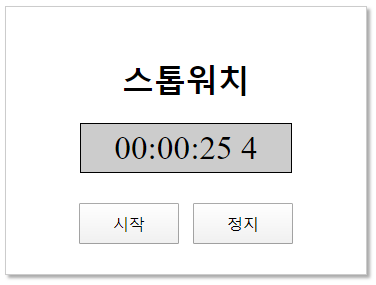
따라서 화면이 처음 그려질 때도, 현재 시각이 출력된다.

# 스톱 워치

## 밀리초 계산

### stopWatch1.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#timer { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>스톱워치</h1>      <div id="timer">{{ time }}</div>      <button type="button" v-on:click="start">시작</button>      <button type="button" v-on:click="stop">정지</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: '00:00:00 0'          startTime: null        },        methods: {          start() {            this.startTime = new Date();            let callback = () => {              let milisec = new Date() - this.startTime;              this.time = formatTimeFromMilisec(milisec);            };            this.timerId = setInterval(callback, 100);          },          stop() {            clearInterval(this.timerId);          }        }      })      const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;      function formatTimeFromMilisec(milisec) {        let h = Math.floor(milisec / HH);        let m = Math.floor(milisec % HH / MM);        let s = Math.floor(milisec % MM / SS);        let t = Math.floor(milisec % SS / 100);        if (h < 10) h = "0" + h;        if (m < 10) m = "0" + m;        if (s < 10) s = "0" + s;        return h + ":" + m + ":" + s + " " + t;      }     </script>  </body>  </html> |
|  |
|  |



button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }

padding: 0.5em 2em;

버튼 내부 위 아래 방향 여백은 현재 폰트의 크기의 0.5배 이고,

좌 우 방향 여백은 현재 폰트의 크기의 2배 이다.

data: {

  time: '00:00:00 0',

startTime: null

},

time 속성은 스톱워치 화면에 표시할 시각이다.

startTime 속성은 스톱워치가 시작된 시각이다.

start() {

this.startTime = new Date();

  let callback = () => {

   let milisec = new Date() - this.startTime;

    this.time = formatTimeFromMilisec(milisec);

  };

  this.timerId = setInterval(callback, 100);

},

시작 버튼이 클릭되면 이 메소드가 호출된다.

this.startTime = new Date();

startTime 속성에 현재 시각을 대입한다.

이 값은 스톱워치의 시작 시각이다.

let callback = () => {

let milisec = new Date() - this.startTime;

  this.time = formatTimeFromMilisec(milisec);

};

0.1 초 마다 한 번씩 호출될 콜백 함수이다.

현재 시각에서 스톱워치 시작 시각을 뺀 값을 milisec 변수에 대입한다.

이 값은 밀리초이다.

formatFileFromMilisec 함수는 밀리초로부터 'hh:mm:ss m' 형태의 문자열을 생성하여 리턴한다.

이 문자열을 앱의 time 속성에 대입한다.

이 문자열이 화면에 출력된다.

this.timerId = setInterval(callback, 100);

콜백 함수가 100 밀리초 마다 한 번씩 호출되도록 등록한다.

등록 결과 timer id 값을 앱의 timerId 속성에 대입한다.

앱의 data { ... } 부분에 timerId 속성이 정의되지 않았음에 주의하자.

이 부분에 등록된 속성만 실시간 화면 입출력 기능이 제공된다.

this.timerId = setInterval(callback, 100);

앱의 메소드에서 위 코드를 실행하면, 앱에 timerId 속성이 생성된다.

javascript의 속성(멤버 변수)에 어떤 값을 대입할 때, 그 속성이 이미 있다면 값이 대입되고,

그 속성이 아직 없다면 새로 생성된 후 대입된다.

그렇지만 timerId 속성은 data { ... } 부분에 들어있지 않기 때문에,

이 속성은 실시간 화면 입출력에 사용할 수는 없다.

html 태그 부분에서 입력을 받거나(v-model), 출력될 속성은 data { ... } 부분에 정의되어야 한다.

stop() {

clearInterval(this.timerId);

}

정지 버튼이 클릭되면 이 메소드가 호출된다.

setInterval로 등록했던 콜백 함수 자동 호출을 취소한다.

const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;

function formatTimeFromMilisec(milisec) {

let h = Math.floor(milisec / HH);

  let m = Math.floor(milisec % HH / MM);

  let s = Math.floor(milisec % MM / SS);

  let t = Math.floor(milisec % SS / 100);

  if (h < 10) h = "0" + h;

  if (m < 10) m = "0" + m;

  if (s < 10) s = "0" + s;

  return h + ":" + m + ":" + s + " " + t;

}

formatFileFromMilisec 함수는 밀리초로부터 'hh:mm:ss m' 형태의 문자열을 생성하여 리턴한다.

const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;

밀리초를 1000으로 나누면 초,

1000 \* 60으로 나누면 분,

1000 \* 60 \* 60으로 나누면 시이다.

let h = Math.floor(milisec / HH);

밀리초를 1000 \* 60 \* 60 으로 나누고 소수점 아래를 버리면, 시 부분만 남는다.

let m = Math.floor(milisec % HH / MM);

밀리초를 1000 \* 60 \* 60 으로 나눈 나머지를 구하면, 밀리초에서 시 부분만 없어진다.

그 나머지를 1000 \* 60 으로 나누고 소수점 아래를 버리면, 분 부분만 남는다.

let s = Math.floor(milisec % MM / SS);

밀리초를 1000 \* 60 으로 나눈 나머지를 구하면, 밀리초에서 시 부분과 분 부문이 없어진다.

그 나머지를 1000 으로 나누고 소수점 아래를 버리면, 초 부분만 남는다.

let t = Math.floor(milisec % SS / 100);

밀리초를 1000 으로 나눈 나머지를 구하면, 밀리초에서 시, 분, 초 부문이 없어진다.

그 나머지를 100 으로 나누고 소수점 아래를 버리면, 초 바로 아래 한 자리만 남는다.

즉 밀리초에서 백의 자리 한 자리만 남는다.

## v-bind:disabled

### disabled1.html

|  |
| --- |
| <html>  <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: 400px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>disabled</h1>      <button type="button" disabled="disabled">버튼1</button>      <button type="button" disabled>버튼2</button>      <button type="button" disabled="val">버튼3</button>      <button type="button" v-bind:disabled="val">버튼4</button>      <button type="button" v-bind:disabled="!val">버튼5</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          val: true        }      })     </script>  </body>  </html> |
|  |



<button type="button" disabled="disabled">버튼1</button>

태그에 disabled 속성이 있으면 그 태그는 비활성화 상태가 된다.

disabled 속성의 값은 무관한다.

<button type="button" disabled>버튼2</button>

태그에 disabled 속성이 있으면 그 태그는 비활성화 상태가 된다.

disabled 속성의 값은 무관한다.

<button type="button" disabled="val">버튼3</button>

태그에 disabled 속성이 있으면 그 태그는 비활성화 상태가 된다.

disabled 속성의 값은 무관한다.

<button type="button" v-bind:disabled="val">버튼3</button>

태그의 속성 앞에 v-bind: 접두어가 붙으면, 그 속성의 값은 javascript 표현식으로 해석된다.

그 표현식의 값이 true에 해당하는 값인 경우에만 그 태그는 비활성화 상태가 된다.

앱의 val 속성의 값이 true 이므로, 이 태그는 비활성화 상태가 된다.

<button type="button" v-bind:disabled="!val">버튼5</button>

val 속성의 값이 true이고, !val 표현식의 값이 false 이므로,

이 태그는 활성화 상태가 된다.

### disabled2.html

|  |
| --- |
| <html>  <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: 400px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=checkbox] { width: 20px; height: 20px; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>disabled</h1>      <input type="checkbox" v-model="val" />      <div>        <button type="button" v-bind:disabled="val">버튼1</button>        <button type="button" v-bind:disabled="!val">버튼2</button>      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          val: true        }      })     </script>  </body>  </html> |



체크를 켜고 끔에 따라 버튼들의 비활성화 상태가 바뀐다.

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

체크가 켜지면 앱의 val 속성에 true 값이 즉시 대입되고,

체크가 꺼지면 false 값이 즉시 대입된다.

<button type="button" v-bind:disabled="val">버튼1</button>

앱의 val 속성 값이 true 이면 비활성화 상태, false 이면 활성화 상태가 된다.

<button type="button" v-bind:disabled="!val">버튼2</button>

앱의 val 속성 값이 true 이면 활성화 상태, false 이면 비활성화 상태가 된다.

### disabled3.html

|  |
| --- |
| <html>  <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: 400px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>disabled</h1>      <button type="button" v-on:click="start" v-bind:disabled="run">시작</button>      <button type="button" v-on:click="stop" v-bind:disabled="!run">정지</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          run: false        },        methods: {          start() {            this.run = true;          },          stop() {            this.run = false;          }        }      })     </script>  </body>  </html> |
|  |
|  |



시작 버튼을 누르면, 정지 버튼만 활성화 상태가 되고,

정지 버튼을 누르면, 시작 버튼만 활성화 상태가 된다.

### disabled4.html

|  |
| --- |
| <html>  <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: 400px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>disabled</h1>      <button type="button" v-on:click="run = true" v-bind:disabled="run">시작</button>      <button type="button" v-on:click="run = false" v-bind:disabled="!run">정지</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          run: false        }      })     </script>  </body>  </html> |



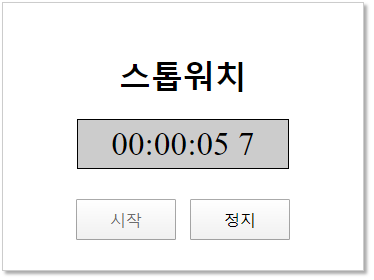
<button type="button" v-on:click="run = true" v-bind:disabled="run">시작</button>

이 버튼이 클릭되면 앱의 run 변수에 true 값을 대입한다.

태그 부분에서 속성을 사용할 때는, this 키워드를 생략한다.

### stopWatch2.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#timer { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>스톱워치</h1>      <div id="timer">{{ time }}</div>      <button type="button" v-on:click="start" v-bind:disabled="run">시작</button>      <button type="button" v-on:click="stop" v-bind:disabled="!run">정지</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          time: '00:00:00 0',          startTime: null,          run: false        },        methods: {          start() {            this.startTime = new Date();            let callback = () => {              let milisec = new Date() - this.startTime;              this.time = formatTimeFromMilisec(milisec);            };            this.timerId = setInterval(callback, 100);            this.run = true;          },          stop() {            clearInterval(this.timerId);            this.run = false;          }        }      })      const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;      function formatTimeFromMilisec(milisec) {        let h = Math.floor(milisec / HH);        let m = Math.floor(milisec % HH / MM);        let s = Math.floor(milisec % MM / SS);        let t = Math.floor(milisec % SS / 100);        if (h < 10) h = "0" + h;        if (m < 10) m = "0" + m;        if (s < 10) s = "0" + s;        return h + ":" + m + ":" + s + " " + t;      }     </script>  </body>  </html> |
|  |



스톱워치가 정지되어 있을 때는 '시작' 버튼만 활성화 상태이고,

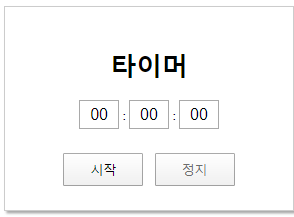
스톱워치가 시작되었을 때는, '정지' 버튼만 활성화 상태이다.

# 타이머

## 밀리초 계산

### timer1.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#timer { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      input[type=text] { font-size: 15pt; padding: 5px; width: 50px; text-align: center; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>타이머</h1>      <div>        <input type="text" v-model="h" /> :        <input type="text" v-model="m" /> :        <input type="text" v-model="s" />      </div>      <button type="button" v-on:click="start" v-bind:disabled="run">시작</button>      <button type="button" v-on:click="stop" v-bind:disabled="!run">정지</button>    </div>    <script type="text/javascript">      const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;      var app = new Vue({        el: '#app',        data: {          h : "00",          m : "00",          s : "00",          stopTime: null,          run: false        },        methods: {          start() {            let now = new Date();            let h = Number(this.h), m = Number(this.m), s = Number(this.s);            let milisec = ((h \* 60 + m) \* 60 + s + 1) \* 1000;            this.stopTime = new Date(now.getTime() + milisec);            let callback = () => {              let milisec = this.stopTime - new Date();              this.h = Math.floor(milisec / HH);              this.m = Math.floor(milisec % HH / MM);              this.s = Math.floor(milisec % MM / SS);              if (this.h < 10) this.h = "0" + this.h;              if (this.m < 10) this.m = "0" + this.m;              if (this.s < 10) this.s = "0" + this.s;              if (milisec < 1000) {                clearInterval(this.timerId);                this.run = false;              }            };            this.timerId = setInterval(callback, 500);            this.run = true;          },          stop() {            clearInterval(this.timerId);            this.run = false;          }        }      })     </script>  </body>  </html> |
|  |



타이머 실행 시간을 시, 분, 초 형태로 입력하고, 시작 버튼을 클릭하면 start 메소드가 호출된다.

입력된 시, 분, 초는 각각 h, m, s 속성에 대입된다.

|  |
| --- |
| let h = Number(this.h), m = Number(this.m), s = Number(this.s);  let milisec = ((h \* 60 + m) \* 60 + s + 1) \* 1000;  this.stopTime = new Date().getTime() + milisec; |

let h = Number(this.h), m = Number(this.m), s = Number(this.s);

앱의 h, m, s 속성은 입력된 시, 분, 초 값이다.

이 속성값들은 input 태그에서 입력된 값이라서 문자열이다.

이 값들을 숫자로 변환하여 각각 지역 변수 h, m, s 에 대입한다.

변환해야 하는 이유는 다음과 같다.

3 + 3 = 6

"3" + "3" = "33"

3 + "3" = "33"

let milisec = ((h \* 60 + m) \* 60 + s + 1) \* 1000;

h, m, s 지역 변수의 값은 각각 시, 분, 초 이다.

이 값들은 타이머 실행 시간이다.

이 값들을 밀리초 단위로 변환하여 합한다. 여기에 1초를 더한다.

new Date().getTime()

new Date() 표현식은 Date 객체를 생성한다. 이 객체의 값은 현재 시각이다.

Date 객체의 getTime() 메소드는 시각을 밀리초 단위 정수 형태로 리턴한다.

즉 위 표현식의 값은 현재 시각을 밀리초 단위 정수이다.

this.stopTime = new Date().getTime() + milisec;

밀리초 단위의 현재 시각 + 밀리초 단위의 타이머 시간 => 타이머가 종료할 시각 (밀리초 단위)

|  |
| --- |
| let callback = () => {    let milisec = this.stopTime - new Date().getTime();    this.h = Math.floor(milisec / HH);    this.m = Math.floor(milisec % HH / MM);    this.s = Math.floor(milisec % MM / SS);    if (this.h < 10) this.h = "0" + this.h;    if (this.m < 10) this.m = "0" + this.m;    if (this.s < 10) this.s = "0" + this.s;    if (milisec < 1000) {      clearInterval(this.timerId);      this.run = false;    }  };  this.timerId = setInterval(callback, 500); |

let milisec = this.stopTime - new Date().getTime();

this.stopTime 타이머가 종료할 시각 (밀리초 단위)

new Date().getTime() 현재 시각 (밀리초 단위)

let milisec = this.stopTime - new Date().getTime();

남은 시간 = 타이머가 종료할 시각 - 현재 시각 (밀리초 단위)

this.h = Math.floor(milisec / HH);

this.m = Math.floor(milisec % HH / MM);

this.s = Math.floor(milisec % MM / SS);

milisec 값에서 시, 분, 초 성분을 꺼내어 각각 h, m, s 속성에 대입한다.

이 속성들은 화면의 input 태그에 연결되에 있다. (v-model)

if (this.h < 10) this.h = "0" + this.h;

if (this.m < 10) this.m = "0" + this.m;

if (this.s < 10) this.s = "0" + this.s;

h, m, s 속성값이 10 보다 작으면, 앞에 "0" 문자열을 붙인다.

if (milisec < 1000) {

clearInterval(this.timerId);

  this.run = false;

}

남은 시각이 1초보다 작으면, 타이머를 종료한다.

this.timerId = setInterval(callback, 500);

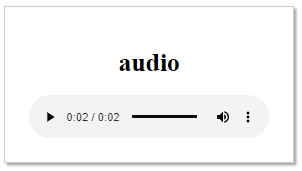
콜백 함수가 0.5초 마다 한 번씩 호출되도록 등록한다.

## mp3 사운드

강의자료 폴더의 ding-dong.mp3 파일을 소스코드 폴더에 복사하자.

### audio1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }    </style>  </head>  <body>    <div id="app">      <h1>audio</h1>      <audio controls src="ding-dong.mp3" type="audio/mpeg">      </audio>    </div>  </body>  </html> |

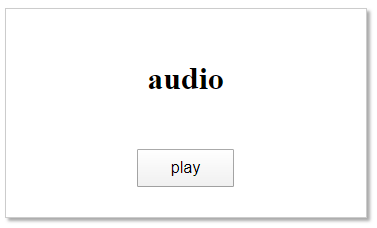


html5 웹브라우저에 사운드 파일 플레이 기능이 내장되어 있다.

### audio2.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>audio</h1>      <button type="button" v-on:click="play">play</button>    </div>    <script type="text/javascript">      var audio = new Audio();      audio.src = "ding-dong.mp3";      var app = new Vue({        el: '#app',        data: {        },        methods: {          play() {            audio.play();          }        }      })     </script>  </body>  </html> |
|  |

javascript 코드에서 mp3 파일을 플레이 하는 코드는 위와 같다.



var audio = new Audio();

Audio 객체를 생성하여 전역 변수 audio에 대입한다.

audio.src = "ding-dong.mp3";

사운드 파일을 서버로부터 다운로드 받아서 audio 객체에 대입한다.

audio 객체의 src 속성값은 사운드 파일의 URL 이어야 한다.

audio.play();

audio 객체의 play 메소드를 호출한다.

사운드가 플레이된다.

그런데,서버로부터 사운드 파일 전체를 다운로드 받기 전에 play 메소드를 호출하면

에러가 발생할 수 있다.

### timer2.html

|  |
| --- |
| <html>  <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: 300px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#timer { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      input[type=text] { font-size: 15pt; padding: 5px; width: 50px; text-align: center; }      button { padding: 0.5em 2em; margin: 30px 5px 0px 5px; font-size: 12pt; }    </style>  </head>  <body>    <div id="app">      <h1>타이머</h1>      <div>        <input type="text" v-model="h" /> :        <input type="text" v-model="m" /> :        <input type="text" v-model="s" />      </div>      <button type="button" v-on:click="start" v-bind:disabled="run">시작</button>      <button type="button" v-on:click="stop" v-bind:disabled="!run">정지</button>    </div>    <script type="text/javascript">      const SS = 1000, MM = 1000 \* 60, HH = 1000 \* 60 \* 60;      var audio = new Audio();      audio.src = "ding-dong.mp3";      var app = new Vue({        el: '#app',        data: {          h : "00",          m : "00",          s : "00",          stopTime: null,          run: false        },        methods: {          start() {            let h = Number(this.h), m = Number(this.m), s = Number(this.s);            let milisec = ((h \* 60 + m) \* 60 + s + 1) \* 1000;            this.stopTime = new Date().getTime() + milisec;            let callback = () => {              let milisec = this.stopTime - new Date().getTime();              this.h = Math.floor(milisec / HH);              this.m = Math.floor(milisec % HH / MM);              this.s = Math.floor(milisec % MM / SS);              if (this.h < 10) this.h = "0" + this.h;              if (this.m < 10) this.m = "0" + this.m;              if (this.s < 10) this.s = "0" + this.s;              if (milisec < 1000) {                clearInterval(this.timerId);                this.run = false;                audio.play();              }            };            this.timerId = setInterval(callback, 200);            this.run = true;          },          stop() {            clearInterval(this.timerId);            this.run = false;          }        }      })     </script>  </body>  </html> |

타이머가 완료되면 mp3 파일을 플레이 하는 기능을 추가함.

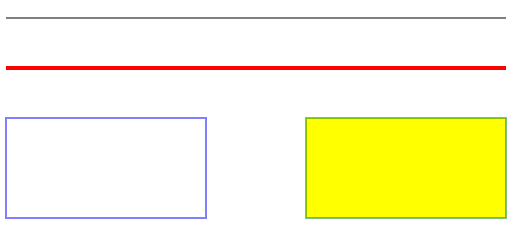
# canvas 그래픽

아날로그 시계를 그리기 위해 필요한 canvas 그래픽 기능을 먼저 학습하자.

## 선과 사각형

### canvas1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />  </head>  <body>    <canvas id="clock" width="600" height="600"></canvas>      <script type="text/javascript">      let canvas = document.getElementById("clock");      let context = canvas.getContext("2d");      // 선그리기      context.beginPath();      context.moveTo(50, 50);      context.lineTo(550, 50);      context.stroke();      // 선그리기      context.beginPath();      context.lineWidth = 4;      context.moveTo(50, 100);      context.lineTo(550, 100);      context.strokeStyle = "red";      context.stroke();      // 사각형      context.lineWidth = 1;      context.strokeStyle = "blue";      context.strokeRect(50, 150, 200, 100);      // 사각형      context.strokeStyle = "green";      context.fillStyle = "yellow";      context.fillRect(350, 150, 200, 100);      context.strokeRect(350, 150, 200, 100);    </script>  </body>  </html> |



<canvas id="clock" width="600" height="600"></canvas>

웹브라우저에 그림을 그리기 위한 canvase 태그를 가로 세로 600px 크기로 생성한다.

let canvas = document.getElementById("clock");

canvas 태그 객체에 대한 참조를 얻는다.

context = canvas.getContext("2d");

그림을 그리기 위한 context 객체에 대한 참조를 얻는다.

// 선그리기

context.beginPath();

context.moveTo(50, 50);

context.lineTo(550, 50);

context.stroke();

(50, 50) 좌표에서 (550, 50) 좌표까지 직선을 그린다.

### 좌표

좌표의 원점은 왼쪽 위 이다.

x 축의 증가방향은 오른쪽, y 축의 증가 방향은 아래쪽이다.

// 선그리기

context.beginPath();

context.lineWidth = 4;

context.moveTo(50, 100);

context.lineTo(550, 100);

context.strokeStyle = "red";

context.stroke();

(50, 100) 좌표에서 (550, 100) 좌표까지 직선을 그린다.

선의 굵기는 4, 선의 색은 red

beginPath 메소드 = 경로 만들기 시작

moveTo 메소드 = 경로에 직선을 추가하지 않고 경로의 현재 위치를 이동한다.

lineTo 메소드 = 경로에 직선을 추가하면서 경로의 현재 위치를 이동한다.

stroke 메소드 = 지금까지 만든 경로의 선을 화면에 그린다.

// 사각형

context.lineWidth = 1;

context.strokeStyle = "blue";

context.strokeRect(50, 150, 200, 100);

strokeRect 메소드의 파라미터는 (x1, y1, x2, y2) 이다.

(50, 150, 200, 100) 좌표의 사각형의 경계선을 그린다.

선의 굵기는 1, 선의 색은 blue

// 사각형

context.strokeStyle = "green";

context.fillStyle = "yellow";

context.fillRect(350, 150, 200, 100);

context.strokeRect(350, 150, 200, 100);

(350, 150, 200, 100) 좌표의 사각형의

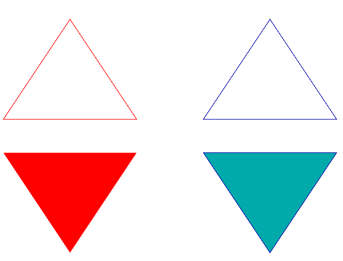
내부를 노란색으로 칠하고 (fillRect)

경계선을 녹색으로 그린다. (strokeRect)

## 경로 그리기

### canvas2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>    </style>  </head>  <body>    <canvas id="clock" width="600" height="600"></canvas>      <script type="text/javascript">      let canvas = document.getElementById("clock");      context = canvas.getContext("2d");      // 삼각형 그리기      context.beginPath();      context.moveTo(50, 200);      context.lineTo(150, 50);      context.lineTo(250, 200);      context.lineTo(50, 200);      context.strokeStyle = "#f00"      context.stroke();      // 삼각형 그리기      context.beginPath();      context.moveTo(350, 200);      context.lineTo(450, 50);      context.lineTo(550, 200);      context.closePath();      context.strokeStyle = "#00a"      context.stroke();      // 삼각형 그리기      context.beginPath();      context.moveTo(50, 250);      context.lineTo(150, 400);      context.lineTo(250, 250);      context.closePath();      context.fillStyle = "#f00"      context.fill();      // 삼각형 그리기      context.beginPath();      context.moveTo(350, 250);      context.lineTo(450, 400);      context.lineTo(550, 250);      context.closePath();      context.fillStyle = "#0aa"      context.fill();      context.strokeStyle = "#00a"      context.stroke();    </script>  </body>  </html> |



|  |  |
| --- | --- |
| // 삼각형 그리기  context.beginPath();  context.moveTo(50, 200);  context.lineTo(150, 50);  context.lineTo(250, 200);  context.lineTo(50, 200);  context.strokeStyle = "#f00"  context.stroke(); |  |

lintTo 메소드를 3 번 호출하여, 삼각형의 세 직선을 그리는 경로를 생성함..

|  |  |
| --- | --- |
| // 삼각형 그리기  context.beginPath();  context.moveTo(350, 200);  context.lineTo(450, 50);  context.lineTo(550, 200);  context.closePath();  context.strokeStyle = "#00a"  context.stroke(); |  |

lineTo 메소드를 2 번 호출하여, 삼각형의 두 직선을 그리는 경로를 생성하고,

마지막으로 closePath 메소드를 호출하여 경로의 현재 위치부터 경로의 시작점까지의 직선을 경로에 추가함.

|  |  |
| --- | --- |
| // 삼각형 그리기  context.beginPath();  context.moveTo(50, 250);  context.lineTo(150, 400);  context.lineTo(250, 250);  context.closePath();  context.fillStyle = "#f00"  context.fill(); |  |

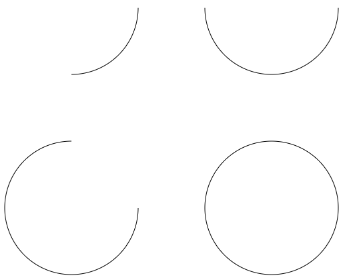
fill 메소드는 경로(path)의 내부를 칠한다.

|  |  |
| --- | --- |
| // 삼각형 그리기  context.beginPath();  context.moveTo(350, 250);  context.lineTo(450, 400);  context.lineTo(550, 250);  context.closePath();  context.fillStyle = "#0aa"  context.fill();  context.strokeStyle = "#00a"  context.stroke(); |  |

## 원둘레 그리기

### canvas3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>    </style>  </head>  <body>    <canvas id="clock" width="700" height="700"></canvas>      <script type="text/javascript">      let canvas = document.getElementById("clock");      let context = canvas.getContext("2d");      var x = 200, y = 100, radius = 100, radian1 = 0, radian2 = 0.5 \* Math.PI;      context.beginPath();      context.arc(x, y, radius, radian1, radian2);      context.stroke();      var x = 500, y = 100, radius = 100, radian1 = 0, radian2 = 1 \* Math.PI;      context.beginPath();      context.arc(x, y, radius, radian1, radian2);      context.stroke();      var x = 200, y = 400, radius = 100, radian1 = 0, radian2 = 1.5 \* Math.PI;      context.beginPath();      context.arc(x, y, radius, radian1, radian2);      context.stroke();      var x = 500, y = 400, radius = 100, radian1 = 0, radian2 = 2 \* Math.PI;      context.beginPath();      context.arc(x, y, radius, radian1, radian2);      context.stroke();    </script>  </body>  </html> |



#### context.arc(x, y, radius, startingAngle, endingAngle)

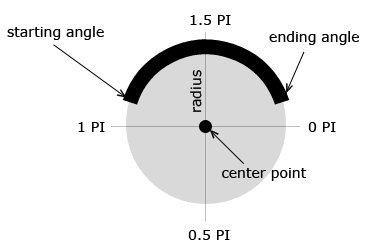
경로에 원둘레(arc)를 추가함

x, y => 원둘레의 중심 좌표

radius => 원의 반지름

starting angle => 원둘레의 시작 각도 (radian 값)

ending angle => 원둘레의 끝 각도 (radian 값)

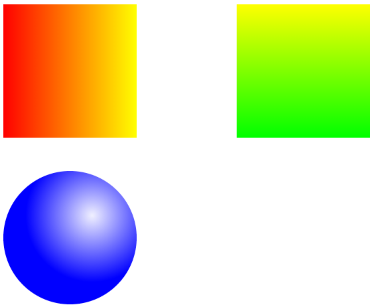


|  |  |  |
| --- | --- | --- |
| degree | radian | 방향 |
| 0 | 0 | 3시 |
| 90 | 0.5π | 6시 |
| 180 | 1π | 9시 |
| 270 | 1.5π | 12시 |
| 360 | 2π | 3시 |

## 그라디언트 칠하기

### canvas4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>    </style>  </head>  <body>    <canvas id="clock" width="600" height="600"></canvas>      <script type="text/javascript">      let canvas = document.getElementById("clock");      context = canvas.getContext("2d");      // 사각형, 수평 방향 linear gradient      var x = 50, y = 50, width = 200, height = 200;      var gradient = context.createLinearGradient(x, 0, x + width, 0);      gradient.addColorStop(0,"#F00");      gradient.addColorStop(1,"#FF0");      context.fillStyle = gradient;      context.fillRect(x, y, width, height);      // 사각형, 수직 방향 linear gradient      var x = 400, y = 50, width = 200, height = 200;      var gradient = context.createLinearGradient(0, y, 0, y + height);      gradient.addColorStop(0,"#FF0");      gradient.addColorStop(1,"#0F0");      context.fillStyle = gradient;      context.fillRect(x, y, width, height);      // 원, radial gradient      var x = 150, y = 400;      var radius = 100, gx = x + radius/3, gy = y - radius/3;      var gradient = context.createRadialGradient(gx, gy, 1, gx, gy, radius);      gradient.addColorStop(0,"#EEF");      gradient.addColorStop(1,"#00F");      context.fillStyle = gradient;      context.beginPath();      context.arc(x, y, radius, 0,  2 \* Math.PI);      context.fill();    </script>  </body>  </html> |



#### context.createLinearGradient(x1, y1, x2, y2)

(x1, y1) 좌표에서 (x2, y2) 좌표로 방향으로,

한 색에서 다른 색으로 점점 변해가는 칠하기 객체를 생성한다.

#### context.createRadialGradient(x1, y1, radius1, x2, y2, radius2);

한 원의 경계선에서 다른 원의 경계선 방향으로

한 색에서 다른 색으로 점점 변해가는 칠하기 객체를 생성한다.

첫째 원의 중심 좌표 (x1, y1) 반지름 radius1

둘째 원의 중심 좌표 (x2, y2) 반지름 radius2

#### gradient.addColorStop(n, color);

n 번째 color를 지정한다.

|  |  |
| --- | --- |
| // 사각형, 수평 방향 linear gradient  var x = 50, y = 50, width = 200, height = 200;  var gradient = context.createLinearGradient(x, 0, x + width, 0);  gradient.addColorStop(0,"#F00");  gradient.addColorStop(1,"#FF0");  context.fillStyle = gradient;  context.fillRect(x, y, width, height); |  |

|  |  |
| --- | --- |
| // 사각형, 수직 방향 linear gradient  var x = 400, y = 50, width = 200, height = 200;  var gradient = context.createLinearGradient(0, y, 0, y + height);  gradient.addColorStop(0,"#FF0");  gradient.addColorStop(1,"#0F0");  context.fillStyle = gradient;  context.fillRect(x, y, width, height); |  |

|  |  |
| --- | --- |
| // 원, radial gradient  var x = 150, y = 400;  var radius = 100, gx = x + radius/3, gy = y - radius/3;  var gradient = context.createRadialGradient(gx, gy, 1, gx, gy, radius);  gradient.addColorStop(0,"#EEF");  gradient.addColorStop(1,"#00F");  context.fillStyle = gradient;  context.beginPath();  context.arc(x, y, radius, 0,  2 \* Math.PI);  context.fill(); |  |

## 문자열 그리기

### canvas5.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>    </style>  </head>  <body>    <canvas id="clock" width="600" height="600"></canvas>      <script type="text/javascript">      let canvas = document.getElementById("clock");      context = canvas.getContext("2d");      var x = 50, y = 100;      context.fillStyle = "#F00";      context.font = "100px 궁서체";      context.fillText("안녕하세요", x, y);        var x = 50, y = 250;      context.strokeStyle = "#00F";      context.strokeText("안녕하세요", x, y);      var x = 50, y = 400;      var gradient = context.createLinearGradient(x, 0, x + 400, 0);      gradient.addColorStop(0,"#0F0");      gradient.addColorStop(1,"#00F");      context.fillStyle = gradient;      context.fillText("안녕하세요", x, y);    </script>  </body>  </html> |



#### context.fillText(문자열, x, y)

x, y 위치에 문자열을 그린다. 문자열의 내부를 칠하는 방식으로 그린다.

#### context.strokeText(문자열, x, y)

x, y 위치에 문자열을 그린다. 문자열의 경계선을 그리는 방식으로 그린다.

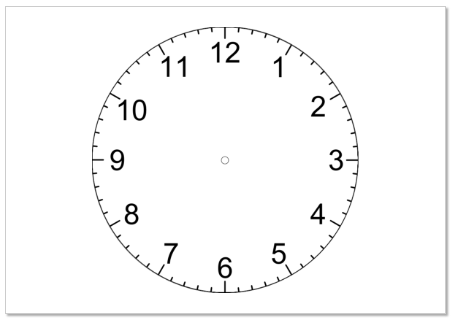
# 아날로그 시계

## 배경 이미지

강의자료 폴더에서 clock.png 파일을 다운로드 받아서 소스코드 폴더에 넣자.

### analog\_clock1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      canvas {        background-image: url(clock.png);        background-size: 400px; background-position: center center;        background-repeat: no-repeat;      }    </style>  </head>  <body>    <div id="app">      <canvas id="clock" width="400" height="400"></canvas>    </div>  </body>  </html> |
|  |



background-image: url(clock.png);

태그의 배경에 clock.png 이미지를 그린다.

background-size: 400px;

배경 이미지를 축소 확대하여 400px 크기로 그린다.

background-position: center center;

배경 이미지를 태그의 가로 세로 가운데 그린다.

background-repeat: no-repeat;

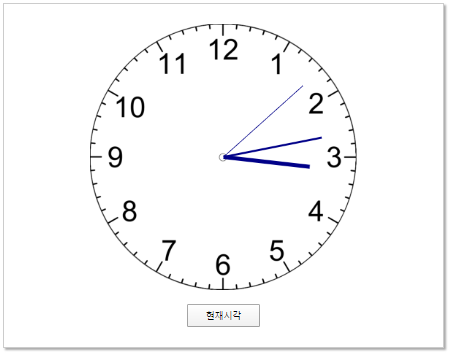
배경 이미지를 한 번만 그린다.

이 속성의 디폴트 값은 repeat = 마치 타일처럼 가로 세로 방향으로 연속해서 여러 번 그리는 방식이다.

## 현재 시각 그리기

### analog\_clock2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      canvas {        background-image: url(clock.png);        background-size: 400px; background-position: center center;        background-repeat: no-repeat;      }      button { padding: 0.5em 2em; margin-top: 20px; }    </style>  </head>  <body>    <div id="app">      <canvas id="clock" width="400" height="400"></canvas>      <div>        <button type="button" onclick="drawClock()">현재시각</button>      </div>    </div>    <script type="text/javascript">      const centerX = 200, centerY = 200;      var context;      function drawClock() {        let now = new Date();        let h = now.getHours(), m = now.getMinutes(), s = now.getSeconds();        let h\_angle = h \* 360 / 12 + m \* 360 / 12 / 60;        drawClockHand(h\_angle - 90, 6, 130);        let m\_angle = m \* 360 / 60 + s \* 360 / 60 / 60;        drawClockHand(m\_angle - 90, 3, 150);        let s\_angle = s \* 360 / 60;        drawClockHand(s\_angle - 90, 1, 160);      }      function drawClockHand(angle, width, radius) {        let radian = Math.PI / 180 \* angle;        let x = centerX + Math.cos(radian) \* radius;        let y = centerY + Math.sin(radian) \* radius;        context.beginPath();        context.lineWidth = width;        context.moveTo(centerX, centerY);        context.lineTo(x, y);        context.strokeStyle = "#008";        context.stroke();      }      window.onload = function() {        let canvas = document.getElementById("clock");        context = canvas.getContext("2d");      }     </script>  </body>  </html> |



현재 시각 버튼을 클릭하면, 현재 시각 시 분 초 바늘을 그린다.

Vue 의 기능이 필요 없기 때문에, 평범한 javascript 기능만으로 구현함.

|  |
| --- |
| const centerX = 200, centerY = 200; |

시계의 중심 좌표

|  |
| --- |
| var context; |

convas context 객체에 대한 참조를 저장할 전역 변수

|  |
| --- |
| function drawClock() {    let now = new Date();    let h = now.getHours(), m = now.getMinutes(), s = now.getSeconds();    let h\_angle = h \* 360 / 12 + m \* 360 / 12 / 60;    drawClockHand(h\_angle - 90, 6, 130);  let m\_angle = m \* 360 / 60 + s \* 360 / 60 / 60;    drawClockHand(m\_angle - 90, 3, 150);    let s\_angle = s \* 360 / 60;    drawClockHand(s\_angle - 90, 1, 160);  } |

let now = new Date();

let h = now.getHours(), m = now.getMinutes(), s = now.getSeconds();

현재 시각의 시, 분, 초 성분을 각각 h, m, s 지역 변수에 대입한다.

let h\_angle = h \* 360 / 12 + m \* 360 / 12 / 60;

시 바늘의 각도 계산

h \* 360 / 12 => 시 바늘이 0시에서 12시까지 이동하면 360도를 이동한다.

따라서 0시를 0도라고 할 때, 1시는 (360 / 12)도 이다.

m \* 360 / 12 / 60 => 1시 30분이라면 시 바늘은 1시와 2시 사이의 가운데에 있어야 한다.

즉 30분 동안 시 바늘은 (360 / 12) / 2 도 움직여야 하고

1분 동안 시 바늘은 (360 / 12) / 60 도 움직여야 한다.

drawClockHand(h\_angle - 90, 6, 130);

사계 바늘을 그리는 함수 호출

이 함수의의 파라미터는 (각도, 바늘 굵기, 바늘 길이)

h\_angle - 90도 각도에 바늘을 그려야 한다.

시계에서 3시 방향은 90도이고, 9시 방향은 180도 이지만,

canvase에서 3시 방향은 0도이다, 9시 방향은 90도 이다.

그래서 h\_gangle - 90 도에 바늘을 그려야 한다.

let m\_angle = m \* 360 / 60 + s \* 360 / 60 / 60;

분 바늘의 각도 계산

m \* 360 / 60 => 분 바늘이 0분에서 60분까지 이동하면 360도를 이동한다.

따라서 0분을 0도라고 할 때, 1분은 (360 / 60)도 이다.

s \* 360 / 60 / 60 => 1분 30초라면 분 바늘은 1분과 2분 사이의 가운데에 있어야 한다.

즉 30초 동안 분 바늘은 (360 / 60) / 2 도 움직여야 하고

1초 동안 분 바늘은 (360 / 60) / 60 도 움직여야 한다.

let s\_angle = s \* 360 / 60;

초 바늘의 각도 계산

s \* 360 / 60 => 초 바늘이 0초에서 60초까지 이동하면 360도를 이동한다.

따라서 0초를 0도라고 할 때, 1초는 (360 / 60)도 이다.

|  |
| --- |
| function drawClockHand(angle, width, radius) {    let radian = Math.PI / 180 \* angle;    let x = centerX + Math.cos(radian) \* radius;    let y = centerY + Math.sin(radian) \* radius;    context.beginPath();    context.lineWidth = width;    context.moveTo(centerX, centerY);    context.lineTo(x, y);    context.strokeStyle = "#008";    context.stroke();  } |

let radian = Math.PI / 180 \* angle;

degree 각도를 radian 값으로 변환한다.

(예: 180도 = 1π)

let x = centerX + Math.cos(radian) \* radius;

let y = centerY + Math.sin(radian) \* radius;

바늘 끝의 x, y 좌표를 계산한다.

context.moveTo(centerX, centerY);

context.lineTo(x, y);

시계 중심에서 바늘끝까지 직선

|  |
| --- |
| window.onload = function() {  let canvas = document.getElementById("clock");    app.context = canvas.getContext("2d");  } |

window.onload = function() { ... }

현재 페이지의 모든 내용이 웹브라우저에 로드(load)된 후 이 함수가 호출된다.

canvas 배경에 clock.png 그림이 그려져야 한다.

clock.png 파일이 웹브라우저에 로드된 후에야 canvas 객체를 사용할 수 있고,

context 객체에 대한 참조를 구할 수도 있다.

app.context = canvas.getContext("2d");

canvas context 객체에 대한 참조를, Vue 앱 객체의 context 속성에 대입한다.

## 시계

### analog\_clock3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 600px; text-align: center;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#time { font-size: 25pt; padding: 5px; margin: 0px auto; background: #ccc;        max-width: 200px; border: 1px solid black; }      canvas {        background-image: url(clock.png);        background-size: 400px; background-position: center center;        background-repeat: no-repeat;      }    </style>  </head>  <body>    <div id="app">      <canvas id="clock" width="400" height="400"></canvas>    </div>    <script type="text/javascript">      const centerX = 200, centerY = 200;      var context;      function drawClock() {        clearCanvas();        let now = new Date();        let h = now.getHours(), m = now.getMinutes(), s = now.getSeconds();        let h\_angle = h \* 360 / 12 + m \* 360 / 12 / 60;        drawClockHand(h\_angle - 90, 6, 130);        let m\_angle = m \* 360 / 60 + s \* 360 / 60 / 60;        drawClockHand(m\_angle - 90, 3, 150);        let s\_angle = s \* 360 / 60;        drawClockHand(s\_angle - 90, 1, 160);      }      function clearCanvas() {        context.clearRect(0, 0, centerX \* 2, centerY \* 2);      }      function drawClockHand(angle, width, radius) {        let radian = Math.PI / 180 \* angle;        let x = centerX + Math.cos(radian) \* radius;        let y = centerY + Math.sin(radian) \* radius;        context.beginPath();        context.lineWidth = width;        context.moveTo(centerX, centerY);        context.lineTo(x, y);        context.strokeStyle = "#008";        context.stroke();      }      window.onload = function() {        let canvas = document.getElementById("clock");        context = canvas.getContext("2d");        setInterval(() => drawClock(), 1000);      }     </script>  </body>  </html> |

노란색으로 칠한 코드가 추가되었다.

context.clearRect(0, 0, centerX \* 2, centerY \* 2);

시계 바늘이 그려지는 영역을 지운다.

setInterval(() => drawClock(), 1000);

1초마다 한 번씩 drawClock 함수를 호출하여 시계 바늘을 그린다.