리액트 동작 원리

**학습목표**

DOM, Virtual DOM 개념

JSX 문법

함수형 컴포넌트

key prop

**목차**

[1. 가상 DOM (Virtual DOM) 2](#_Toc145230381)

[1) ex02 프로젝트 생성 2](#_Toc145230382)

[2) index.html 파일 살펴보기 3](#_Toc145230383)

[3) index.tsx 파일 살펴보기 4](#_Toc145230384)

[4) DOM 5](#_Toc145230385)

[5) Virtual DOM 5](#_Toc145230386)

[2. JSX 7](#_Toc145230387)

[1) 순수 자바스크립트 문법으로 구현 7](#_Toc145230388)

[2) JSX 문법으로 구현 8](#_Toc145230389)

[3) JSX 문법 오류 #1 11](#_Toc145230390)

[4) JSX 문법 오류 #2 12](#_Toc145230391)

[5) CSS Style 13](#_Toc145230392)

[3. 함수형 컴포넌트 16](#_Toc145230393)

[1) App 컴포넌트 16](#_Toc145230394)

[2) MyTable 컴포넌트 구현 17](#_Toc145230395)

[4. 정리 19](#_Toc145230396)

[1) 요약 19](#_Toc145230397)

[2) 연습문제 19](#_Toc145230398)

# 가상 DOM (Virtual DOM)

## ex02 프로젝트 생성

프로젝트를 만들 적당한 디렉토리에서 다음 명령을 실행하자.

|  |
| --- |
| npx create-react-app ex02 --template typescript |

|  |
| --- |
|  |

Visual Studio Code에서 ex02 프로젝트 디렉토리를 열고

다음 명령을 실행하여 프로젝트를 개발 모드로 실행하자.

|  |
| --- |
| npm start |

## index.html 파일 살펴보기

### public/index.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">    <head>      <meta charset="utf-8" />      <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />      <meta name="viewport" content="width=device-width, initial-scale=1" />      <meta name="theme-color" content="#000000" />      <meta        name="description"        content="Web site created using create-react-app"      />      <link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />      <!-- 생략 -->      <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />      <!-- 생략 -->      <title>React App</title>    </head>    <body>      <noscript>You need to enable JavaScript to run this app.</noscript>      <div id="root"></div>      <!-- 생략 -->    </body>  </html> |

### Single Page App

리액트 앱을 포함하여 프론트엔드 앱들은 싱글 페이지 앱이다. (Single Page App)

하나의 웹 페이지 내에서만 실행된다는 뜻이다.

그 웹페이지를 벗어나면 프론트엔드 앱은 종료된다.

그 웹 페이지를 새로고침하면 프론트엔드 앱은 재시작 하게 된다.

html 웹페이지에 구현된 자바스크립트 코드가 그 웹페이지를 벗어나면 종료되기 때문이다.

### public/index.html 파일

이 프로젝트에 구현된 리액트 앱은, index.html 페이지 내에서 실행된다.

<div id="root"></div>

리액트 앱의 화면은 이 div 태그 내부에 출력된다. (렌더링 된다)

### 랜더링(rendering)

리액트 앱에서 생성한 화면은 Html 태그들로 구성되어 있다

그 html 태그들을 웹 페이지에 보여지도록 하는 작업을 렌더링이라고 부른다.

## index.tsx 파일 살펴보기

### src/index.tsx

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import './index.css';  import App from './App';  import reportWebVitals from './reportWebVitals';  const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <App />    </React.StrictMode>  );  // If you want to start measuring performance in your app, pass a function  // to log results (for example: reportWebVitals(console.log))  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  reportWebVitals(); |

import React from 'react';

import ReactDOM from 'react-dom/client';

React 라이브러리 패키지와 ReactDOM 라이브러리 패키지를 import 한다.

import './index.css';

index.css 파일의 CSS 서식을 적용한다.

import App from './App';

App.tsx 파일에 구현된 App 컴포넌트를 import 한다.

리액트 앱의 구성요소는 컴포넌트이다.

import reportWebVitals from './reportWebVitals';

reportWebVitals 라이브러리 패키지를 import 한다.

이것은 리액트 앱의 성능을 측정하기 위한 패키지이다.

|  |
| --- |
| const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <App />    </React.StrictMode>  ); |

위 코드의 기능은 다음과 같다.

<div id="root"></div> 태그에 App 컴포넌트를 렌더링 한다.

## DOM

웹브라우저 내부의 html 태그 객체를 DOM (Document Object Model) 이라고 한다.

대부분의 웹브라우저가 C++로 구현되었기 때문에,

DOM 객체도 C++ 클래스로 구현된 객체이다.

웹브라우저가 HTML 문서를 읽으면,

HTML 태그 각각에 대해서 웹브라우저 내부에 DOM 객체가 생성되고,

그 DOM 객체들이 웹브라우저 창에 그려진다.

### 프론트엔드 앱의 화면

(1) 프론트엔드 앱의 화면에 해당하는 html 태그 DOM 객체들을 처음부터 전부 새로 생성하기

(2) 데이터가 변경되면 화면에서 그 데이터가 출력된 부분의 html 태그 DOM 객체 수정하기

(1) 구현보다 (2) 구현이 훨씬 복잡하고 어렵다.

그래서 (2) 부분을 구현하지 않고 (1) 부분만 구현하고,

데이터가 변경될 때 마다 화면 전체를 새로 생성하는 방식이 훨씬 구현하기 쉽다.

구현하기는 쉽지만 너무 비효율적이고 느리다.

이 비효율을 개선하는 방법은??

## Virtual DOM

웹브라우저 내부의 DOM 객체들에는 많은 기능이 구현되어 있어서 무겁고 느리다.

기능이 많아서 무겁고 느린 DOM 객체들을,

데이터가 변경될 때 마다 전부 새로 만드는 것은 너무 비효율적이다.

그래서 등장한 아이디어가 Virtual DOM 객체를 활용하는 것이다.

Virtual DOM은 DOM을 흉내내서 만든 평범한 javascript 객체이다.

DOM과는 달리 별 기능이 없어서 가볍고 빠르다.

데이터가 수정될 때마다,

무거운 DOM 객체들 전체를 다시 생성하는 것은 너무 느리지만,

가벼운 Virtual DOM 객체들 전체를 다시 생성하는 것은 빠르다.

### React와 개발자의 역할 분담

개발자는

데이터를 웹브라우저 창에 출력하기 위해, 웹페이지 전체 Virtual DOM 객체를 생성하는 코드만 구현한다.

이 구현은 비교적 쉽다.

데이터가 수정되거나 삭제될 때, 해당 Virtual DOM 객체를 찾아서 수정하거나 삭제하는 코드는

구현하지 않는다. 이 구현이 꽤 어려운데 안 해도 되니 다행이다.

React는

데이터가 수정되거나 삭제될 때 마다, 개발자가 구현한 코드를 실행해서,

웹페이지 전체 Virtual DOM 객체들을 다시 생성한다.

데이터가 수정되기 전, 웹페이지 전체 Virtual DOM 객체들과

데이터가 수정된 후, 웹페이지 전체 Virtual DOM 객체들을 비교한다.

이 비교로 어느 부분이 수정되었고 삭제되었는지를 React가 파악한다.

그 수정되고 삭제된 부분에 해당하는 웹브라우저 내부 DOM 객체들을 React가 수정한다.

장점

React가 딱 필요한 DOM 객체만 수정 삭제하기 때문에 빠르다.

개발자는, 웹페이지 전체 Virtual DOM 객체 생성 코드만 구현하면 되고,

데이터가 수정 삭제될 때, 해당 Virtual DOM 객체를 찾아서 수정 삭제하는 코드까지 구현할 필요는 없다.

그래서 구현이 쉽다.

### 리액트 컴포넌트

리액트 개발자는 리액트 컴포넌트를 구현해야 한다.

리액트 컴포넌트를 클래스 형태로 구현하거나 함수 형태로 구현할 수 있는데,

함수 형태로 구현하는 것이 바람직하다.

함수형 리액트 컴포넌트는 Virtual DOM 객체를 생성해서 리턴하는 자바스크립트 함수이다.

리액트 앱은

화면을 구성하는 html 태그 Virtual DOM 객체들을 생성해서 리턴하는 함수들로 구성된다.

(함수형 컴포넌트들로 구성된다)

# JSX

## 순수 자바스크립트 문법으로 구현

### src/App.css 수정

|  |
| --- |
| table { border-collapse: collapse; margin: 5px; }  td { border: 1px solid gray; padding: 4px; } |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let td1 = React.createElement("td", null, "이름");    let td2 = React.createElement("td", null, "나이");    let tr = React.createElement("tr", null, td1, td2);    let trlist = [tr];    for (let person of persons) {      td1 = React.createElement("td", null, person.name);      td2 = React.createElement("td", null, person.age);      tr = React.createElement("tr", null, td1, td2);      trlist.push(tr);    }    let table = React.createElement("table", null, trlist);    return table;  }  export default App; |

React.createElement(name, attribute, children) 메소드

Virtual DOM 객체를 생성하는 메소드

name: 태그 이름

attribute: 태그 애트리뷰트

children: 태그 자식 (예: <td>태그 자식</td>, <tr>태그 자식</tr>)

Graphical user interface, text, application

Description automatically generated

위 App.tsx의 App() 함수는 리액트 함수형 컴포넌트이다.

이 함수는 Virtual DOM 객체를 생성해서 리턴한다.

React.createElement("td", null, "이름")

이 코드는 <td>이름</td> 형태의 Virtual DOM 태그 객체를 생성한다.

위와 같은 순수 자바스크립트 코드로 Virtual DOM 객체를 생성 것은 귀찮다.

좀더 편한 구현 방법이 필요하다.

## JSX 문법으로 구현

Virtual DOM 개체 생성 코드를 쉽게 구현할 수 있도록 리액트는 자바스크립트 문법을 확장하였다.

그 문법이 JSX 이다.

표준 자바스크립트/타입스크립트 문법으로 구현된 소스코드 파일은 \*.js/\*.ts 이고,

확장된 JSX 문법으로 구현된 자바스크립트/타입스크립트 소스코드 파일은 \*.jsx/\*.tsx 이다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let td1 = <td>이름</td>;    let td2 = <td>나이</td>;    let tr = React.createElement("tr", null, td1, td2);    let trlist = [tr];    for (let person of persons) {      td1 = <td>{ person.name }</td>      td2 = <td>{ person.age }</td>      tr = React.createElement("tr", null, td1, td2);      trlist.push(tr);    }    let table = React.createElement("table", null, trlist);    return table;  }  export default App; |

노란색으로 칠한 부분이 확장된 JSX 문법으로 구현된 Virtual DOM 객체 생성코드이다.

<td>이름</td>

'이름' 부분은 그냥 그대로 출력할 텍스트이므로, 위와 같이 구현한다.

<td>{ person.name }</td>

person.name 부분은 자바스크립트 표현식이라서, 위와 같이 { } 괄호가 필요하다.

{ } 괄호가 없으면 'person.name' 텍스트가 그냥 그대로 출력된다.

위와 같이 JSX 문법으로 구현된 코드는, 빌드단계에서 순수 자바스크립트 코드로 변환된다.

즉 React.createElement 메소드 호출 코드로 변환된다.

### 참고 - 표현식(expression)

어떤 값이 있는 소스 코드 부분은 표현식이다.

숫자 값, 문자열 값, true false 값, 객체, 배열 등

if 문, for 문 그 자체에는 값이 없으므로 표현식이 아니다.

함수를 호출하는 코드는 그 함수의 리턴값이 값이기 때문에 표현식이다.

tr 태그 생성 부분에도 JSX 문법을 적용하면 다음과 같다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let tr = <tr><td>이름</td><td>나이</td></tr>;    let trlist = [tr];    for (let person of persons) {      tr = <tr><td>{ person.name }</td><td>{ person.age }</td></tr>;      trlist.push(tr);    }    let table = React.createElement("table", null, trlist);    return table;  }  export default App; |

table 태그 생성 부분에도 JSX 문법을 적용하면 다음과 같다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let tr = <tr><td>이름</td><td>나이</td></tr>;    let trlist = [tr];    for (let person of persons) {      tr = <tr><td>{ person.name }</td><td>{ person.age }</td></tr>;      trlist.push(tr);    }    let table = <table>{ trlist }</table>;    return table;  }  export default App; |

<table>{ trlist }</table>

trlist는 자바스크립트 변수이므로 { } 괄호로 묶어야 한다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let trlist = [];    for (let person of persons) {      let tr = <tr><td>{ person.name }</td><td>{ person.age }</td></tr>;      trlist.push(tr);    }    let table = <table>                   <tr><td>이름</td><td>나이</td></tr>                   { trlist }                </table>;    return table;  }  export default App; |

위와 같이 좀 더 간결하게 구현할 수 있다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let trlist = persons.map(person =>  <tr><td>{ person.name }</td><td>{ person.age }</td></tr>);    return <table>              <tr><td>이름</td><td>나이</td></tr>              { trlist }           </table>;  }  export default App; |

리액트 개발자들 대부분 위와 같은 형태로 구현한다.

리액트 개발자는 자바스크립트 배열의 map, filter, reduce 메소드를 능숙하게 활용할 수 있어야 한다.

## JSX 문법 오류 #1

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let trlist = persons.map(person =>                      <tr><td>{ person.name }</td><td>{ person.age }</td></tr>);    return      <table>        <tr><td>이름</td><td>나이</td></tr>        { trlist }      </table>;  }  export default App; |

위와 같이 구현하면, JSX 문법 오류이다.

직전 예제와 차이는 단지 return 문 다음에 줄 바꿈만 했을 뿐인데 오류라니...

위와 같은 JSX 문법 오류를 피하려면,

JSX 문법으로 구현된 태그 부분을 ( ) 괄호로 묶어주면 된다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let trlist = persons.map(person =>                      <tr><td>{ person.name }</td><td>{ person.age }</td></tr>);    return (      <table>        <tr><td>이름</td><td>나이</td></tr>        { trlist }      </table>    );  }  export default App; |

- 여러 줄로 구현된 JSX 태그 부분은 ( ) 괄호로 묶어주자.

- JSX 태그 부분에서 자바스크립트 표현식은 { } 괄호로 묶어주자.

## JSX 문법 오류 #2

리액트 컴포넌트가 리턴하는 Virtual DOM 객체는 루트가 있는 트리 구조이어야 한다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    return (      <h1>hello world</h1>    );  }  export default App; |

루트가 h1 태그이므로 OK

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    return (      <h1>hello</h1>      <h1>world</h1>    );  }  export default App; |

루트가 없으므로 에러 (루트가 여러 개 이므로 에러)

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    return (       <div>        <h1>hello</h1>        <h1>world</h1>       </div>    );  }  export default App; |

루트가 div 태그이므로 OK

## CSS Style

### src/App.css 수정

|  |
| --- |
| div { margin: 10px; font-size: 20pt; }  p.blue { color: blue; }  p.green { color: green; } |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    return (      <div>        <p className="blue">hello world</p>        <p className="green">안녕하세요</p>      </div>    );  }  export default App; |

표준 HTML 태그라면 아래와 같이 구현해야 하지만

      <p class="blue">hello world</p>

      <p class="green">안녕하세요</p>

JSX에서 class 단어는 자바스크립트 키워드라서 사용할 수 없다.

JSX에서는 class 대신 className을 사용해야 한다.

Graphical user interface, text, application, email

Description automatically generated

### 인라인 서식 (inline style)

표준 HTML 태그에서 인라인 서식을 아래와 같이 구현한다.

      <p style="color: blue; font-size: 20pt;">hello world</p>

      <p style="color: green; font-size: 25pt;">안녕하세요</p>

HTML에서 p는 태그이고 style은 애트리뷰트이다.

JSX 태그에서 인라인 서식은 아래와 같이 구현해야 한다.

      <p style={{ color: "blue", fontSize: "20pt" }}>hello world</p>

      <p style={{ color: "green", fontSize: "25pt" }}>안녕하세요</p>

JSX에서 p는 자바스크립트 객체이고 style은 속성(property)이다.

속성(property)을 짧게 프롭(prop)이라고 부른다. (properties == props)

style 프롭스의 값은 문자열이 아니고, 자바스크립트 객체이어야 한다.

<p style={ 자바스크립트\_객체 }>

JSX 태그 부분에 자바스크립트 표현식은 { } 괄호로 묶어야 한다.

{ color: "blue", fontSize: "20pt" }

이것은 인라인 서식을 표현한 자바스크립트 객체이다.

자바스크립트 변수 이름이나 속성 이름에 - 문자를 사용할 수 없다.

따라서 인라인 서식을 자바스크립트 객체로 구현할 때,

font-size 형태의 서식명을 fontSize 형태로 구현해야 한다.

### 참고

여러 단어로 구성된 이름을 표현하는 방식

|  |  |
| --- | --- |
| 카멜 케이스(camel case) | multiWordsId |
| 파스칼 케이스(pascal case) | MultiWordsId |
| 스내이크 케이스(snake case) | multi\_words\_id |
| 케밥 케이스 (kebab case) | multi-words-id |

HTML에서 CSS 서식 이름은 케밥 케이스 방식이지만

JSX에서 CSS 서식 이름은 카멜 케이스 방식을 사용해야 한다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    return (      <div>        <p style={{ color: "blue", fontSize: "20pt" }}>hello world</p>        <p style={{ color: "green", fontSize: "25pt" }}>안녕하세요</p>      </div>    );  }  export default App; |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let s1 = { color: "blue", fontSize: "20pt" };    let s2 = { color: "green", fontSize: "25pt" };    return (      <div>        <p style={ s1 }>hello world</p>        <p style={ s2 }>안녕하세요</p>      </div>    );  }  export default App; |

위와 같이 구현할 수도 있다.

s1, s2 변수의 값이 자바스크립트 객체이다.

인라인 서식의 값이 자바스크립트 객체이어야 한다.

# 함수형 컴포넌트

## App 컴포넌트

### src/App.tsx

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let s1 = { color: "blue", fontSize: "20pt" };    let s2 = { color: "green", fontSize: "25pt" };    return (      <div>        <p style={ s1 }>hello world</p>        <p style={ s2 }>안녕하세요</p>      </div>    );  }  export default App; |

src/App.tsx 파일에 구현된 App 컴포넌트를 호출하는 곳은 src/index.tsx 이다.

### src/index.tsx

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import './index.css';  import App from './App';  import reportWebVitals from './reportWebVitals';  const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <App />    </React.StrictMode>  );  // If you want to start measuring performance in your app, pass a function  // to log results (for example: reportWebVitals(console.log))  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  reportWebVitals(); |

App 함수형 컴포넌트를 호출하는 부분이 <App /> 이다.

App 함수가 생성해서 리턴하는 Virtual DOM 객체들이 여기서 렌더링 된다.

## MyTable 컴포넌트 구현

### src/MyTable.css 생성

|  |
| --- |
| table.MyTable { border-collapse: collapse; margin: 5px; }  table.MyTable td { border: 1px solid gray; padding: 4px; } |

### src/MyTable.tsx 생성

|  |
| --- |
| import React from 'react';  import './MyTable.css';  function MyTable() {    let persons = [{ name: '홍길동', age: 16 },                   { name: '임꺽정', age: 19 },                   { name: '전우치', age: 20 }];    let trlist = persons.map(person =>         <tr><td>{ person.name }</td><td>{ person.age }</td></tr>);    return (      <table className="MyTable">        <tr><td>이름</td><td>나이</td></tr>        { trlist }      </table>    );  }  export default MyTable; |

MyTable 함수 컴포넌트가 리턴하는 Virtual DOM 객체들은 다음과 같은 트리 구조이다.

루트 객체는 <table> 객체이고 4개의 자식 <tr> 객체들이 있다.

그 <tr> 객체들 각각은 2개의 자식 <td> 객체들이 있다.

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import MyTable from './MyTable';  function App() {    return (      <div>        <MyTable />        <hr />        <MyTable />      </div>    );  }  export default App; |

MyTable 함수 컴포넌트를 두 번 호출하고 있다.

App 함수 컴포넌트가 리턴하는 Virtual DOM 객체들은 다음과 같은 트리 구조이다.

루트 객체는 <div> 객체이고 3개의 자식 객체들이 있다.

MyTable 함수 컴포넌트가 생성해서 리턴한, <table> 객체가 루트인 트리

<hr /> 객체

MyTable 함수 컴포넌트가 생성해서 리턴한, <table> 객체가 루트인 트리

Text

Description automatically generated with medium confidence

# 정리

## 요약

함수형 리액트 컴포넌트는 Virtual DOM 객체를 생성해서 리턴하는 자바스크립트 함수이다.

JSX 문법으로 구현한 태그는 Virtual DOM 자바스크립트 객체이다. HTML 태그처럼 보이지만 HTML 태그가 아니다. 따라서 JSX 작성 규칙은 HTML 문서 작성 규칙과 다르다.

여러 줄로 구현된 JSX 태그 부분은 ( ) 괄호로 묶어주자.

JSX 태그 부분에서 자바스크립트 표현식은 { } 괄호로 묶어주자.

JSX 태그 부분에서는 class 대신 className을 사용해야 한다.

JSX 태그 부분에서 style 값은 문자열이 아니고, 자바스크립트 객체이어야 한다.

JSX에서 CSS 서식 이름은 카멜 케이스 방식을 사용해야 한다.

## 연습문제

구구단 table 태그 출력 구현하기