

## [React 기초 강좌]

### 1. React에서의 컴포넌트

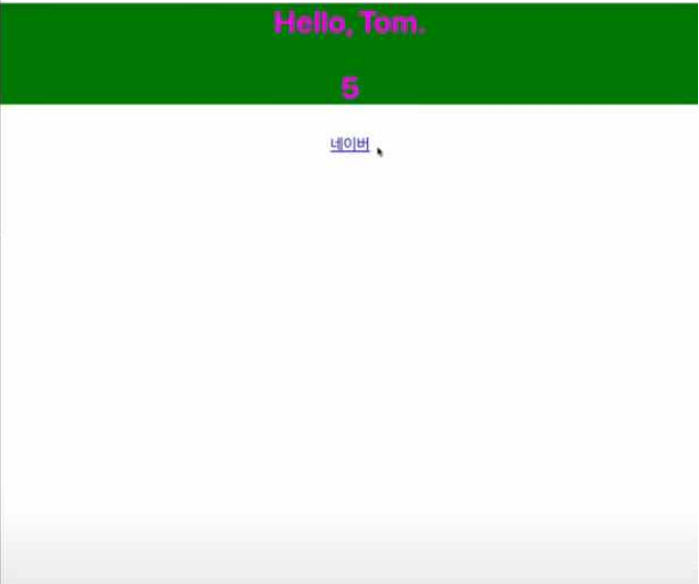
- 간단히 요약하자면 웹 사이트상에서 한 가지의 기능을 하는 모듈의 각 부분 부분을 말함
- 기본적으로 컴포넌트들의 '재사용'을 지향함

#### 1) 컴포넌트의 간단한 구조

- 기본적인 구조 : "xxx.js" / 기본 컴포넌트는 "App.js"
- 내용 : 파일이름과 동일한 함수명으로 구성 / default로 동일한 이름의 함수를 export
- 컴포넌트의 사용 : "index.js"에서 import하여 사용
- 특징 : 모든 컴포넌트는 대문자로 시작함

함수 안의 코드는 "JSX(JavaScript XML)"로 이루어져있음

```
src > js App.js > App
1  import './App.css';
2
3  function App() {
4    const name = "Tom";
5    const naver = {
6      name: "네이버",
7      url: "https://naver.com",
8    };
9    return (
10     <div className="App">
11       <h1
12         style={{
13           color: "#f0f",
14           backgroundColor: "green",
15         }}
16       >
17         Hello, {name}.<p>{2 + 3}</p>
18       </h1>
19       <a href={naver.url}>{naver.name}</a>
20     </div>
21   );
22 }
23
24 export default App;
```



[ ▲ 기본적인 컴포넌트의 사용, 객체는 {변수명} 안에 사용 ]

#### 2) 컴포넌트 만들기

- 기본적으로 component 폴더 하위에 컴포넌트를 모아서 구성
- 컴포넌트의 기본적인 사용방법

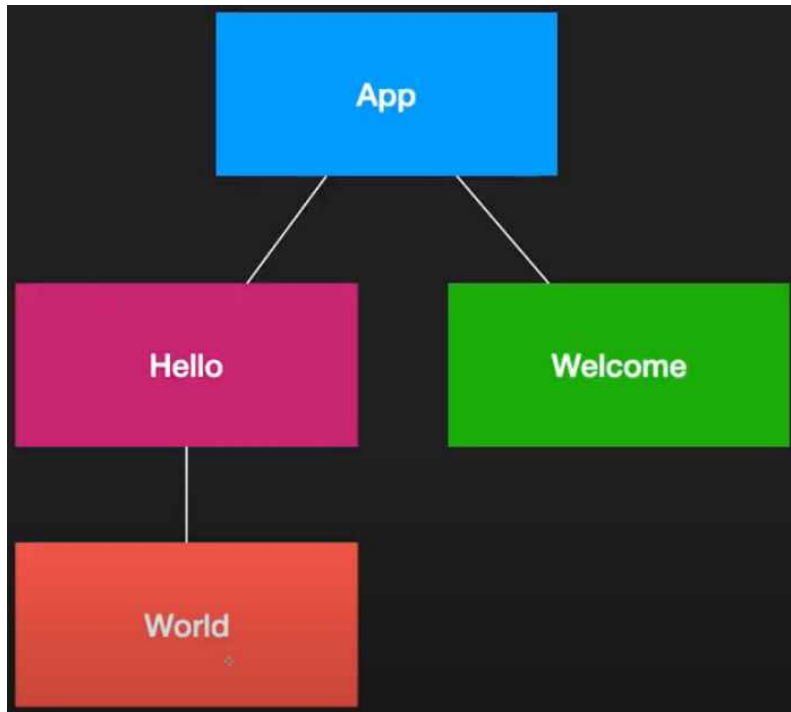
```
export default function Hello() {
  return <p>Hello</p>;
}

1  import './App.css';
2  import Hello from './component/Hello';
3
4  function App() {
5    return <div className="App">
6      <Hello />
7    </div>;
8  }
9
10 export default App;
```

<Hello></Hello>와 같이 사용해도 됨

index.js에 import된 컴포넌트 App.js  
App에서 Hello.js 컴포넌트를 import

- 컴포넌트를 재사용하여 다음과 같은 구조로 구성하는 것도 가능



index.js에서 App.js 하나만 import하면 하위 컴포넌트가 모두 사용이 된다.

- (주의사항) 각 컴포넌트별로 return되는 값은 하나의 태그만 가능하다.
  - ↳ 따라서 여러 태그를 포함하는 경우 `<></>` 빈 태그나 `<div></div>`로 감싸준다.

## 2. React에서의 CSS

### 1) 태그 안에 직접 명시하기

```
export default function Hello() {
  return (
    <div>
      <h1 style={
        color : '#f00',
        borderRight : '2px solid #000',
        marginBottom : '30px',
        opacity : 0.5
      }>Hello</h1>
      <World />
      <World />
    </div>
  );
}
```

## 2) xxx.css 파일 작성 후 import하기

```
.box {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
}
```

< App.css >

Hello.css >

```
.box {  
  width: 200px;  
  height: 50px;  
  background-color: blue;  
}
```

- 위와 같이 **css** 파일을 작성하고 각 컴포넌트마다 import 해준다.



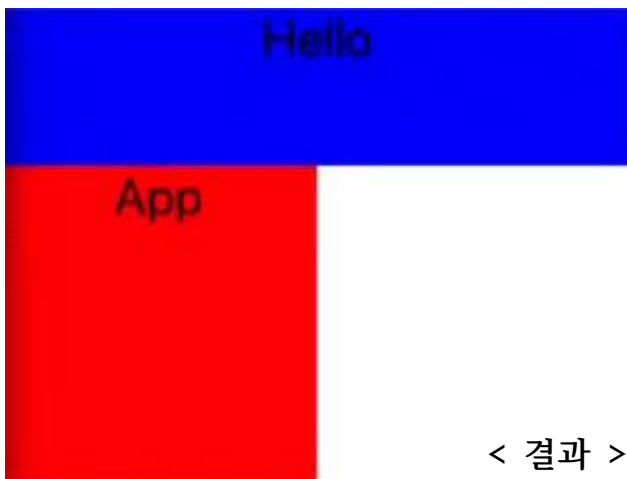
< 결과 >

css 파일은 html 문서 header에  
전체 적용되므로 컴포넌트마다  
각각의 css 파일이 적용되지 않는 모습

## 3) xxx.module.css 파일 작성 후 import하기

- 위의 파일 내용들을 'Hello.module.css'와 같이 새로운 파일을 만들어 작성
- 동일명의 컴포넌트에서 다음과 같이 추가

```
import styles from "./Hello.module.css";  
import styles from "./App.module.css";
```



< 결과 >

### 3. 이벤트 처리 (Handling Events)

- 한 컷 정리 : 이벤트 처리의 다양한 방법

```
export default function Hello() {  
  function showName() {  
    console.log("Mike");  
  }  
  function showAge(age) {  
    console.log(age);  
  }  
  function showText(txt) {  
    console.log(txt);  
  }  
  
  return (  
    <div>  
      <h1>Hello</h1>  
      <button onClick={showName}>Show name</button>  
      <button  
        onClick={() => {  
          showAge(10);  
        }}  
      >  
        Show age  
      </button>  
      <input  
        type="text"  
        onChange={e => {  
          const txt = e.target.value;  
          showText(txt);  
        }}  
      />  
    </div>  
  );  
}
```

## 4. state, useState

### 1) state란?

- 컴포넌트가 가지고 있는 속성값
- 속성값이 변하게 되면 React는 자동으로 설정해놓은 UI로 변화시켜줌
- (예시) 단순히 `let name = "Mike";` 와 같이 변수를 만들어두고,  
이 변수를 이벤트를 통해 변경시켰을 때, html 문서는 이것이 변경되지 않음.  
=> 이유 : DOM 구조가 변화되지 않음
- (해결방안) '`useState`'를 통해 state를 만들어주면 UI를 변경시킬 수 있다.

### 2) useState란?

- React에서 지원하는 '`Hook`'의 기본적인 종류 중 하나
  - ↳ Hook이란? 클래스의 사용없이 state를 만들어 사용하게 해주는 React의 도구
- 기본적인 사용방법

```
import { useState } from "react"; 2.9K (gzipped: 1.3K)

export default function Hello() {
  // let name = "Mike";
  const [name, setName] = useState("Mike");

  return (
    <div>
      <h1>state</h1>
      <h2 id="name">{name}</h2>
      <button
        onClick={() => {
          setName(name === "Mike" ? "Jane" : "Mike");
        }}
      >
        Change
      </button>
    </div>
  );
}
```

#### [ 사용방법 ]

```
const [state, function] = useState("initial state");
function("속성을 변화시키는 동작");
```

state : 일종의 변수 (변수명)

initial state : 위 변수의 초기값

## 5. props (=properties)

### 1) props란?

- 함수형 컴포넌트에서 매개변수의 역할을 하는 것
  - ↳ 그냥 프로그래밍 언어에서 매개변수와 동일함 pass

## 6. 더미 데이터 구현, map() 반복문

### 1) 더미 데이터

- json 파일로부터 데이터를 받아 오는 데이터 묶음

```
import dummy from "../db/data.json";
```

### 2) map() 반복문

- dummy 객체를 통해서 아래와 같이 데이터를 즉시 사용 가능
- map() 함수를 통해서 더미 데이터를 순회하는 반복문을 만들 수 있음

```
return (  
  <ul className="list_day">  
    {dummy.days.map(day => (  
      <li key={day.id}>Day {day.day}</li>  
    ))}  
  </ul>  
);
```

```
"days": [  
  { "id": 1, "day": 1 },  
  { "id": 2, "day": 2 },  
  { "id": 3, "day": 3 }  
],
```

< 위의 js 코드에 사용된  
json 파일 중 일부

- (주의사항) 반복되는 코드에서 **key에 대한 명시가 없으면 오류가 발생한다.**
  - ↳ json 파일에서 임의로 부여한 id값을 임의의 태그 안에 key로 부여하여 해결
- 아래는 위 파일들의 실행결과

Day 1

Day 2

Day 3

- filter() 함수를 통해 조건을 통과한 데이터들만 모아서 새로운 리스트를 만들 수 있다.

```
const wordList = dummy.words.filter(word => word.day === day);
```

## 7. 라우터 구현 (react-router-dom)

- npm install react-router-dom 설치

### 1) React에서 라우팅이란?

- 여러 페이지 간에 이동할 수 있도록 구현한 경로 혹은 그 과정
- react-router-dom 라이브러리에서 import할 목록

```
import { BrowserRouter, Route, Switch } from "react-router-dom";
```

- (App.js 예시)

```
function App() {  
  return (  
    <BrowserRouter>  
      <div className="App">  
        <Header />  
        <Switch>  
          <Route exact path="/">  
            <DayList />  
          </Route>  
          <Route path="/day">  
            <Day />  
          </Route>  
        </Switch>  
      </div>  
    </BrowserRouter>  
  );  
}
```

└ **<BrowserRouter> 태그** : 컴포넌트의 리턴값 전체에 적용

└ **<Switch> 태그** : 위 태그 안에서 페이지마다 개별적으로 노출되는 부분에만 적용

=> <BrowserRouter> 안의 내용 중 <Switch>에 적용되지 않는 부분은 전체 페이지에 공통적으로 노출이 되는 부분이다.

└ **<Route> 태그** : URL 경로 지정, 해당 경로일 때 태그 안의 내용을 실행 (=라우팅)

└ **exact** : 아래에 명시된 라우팅 경로와 겹치는 부분이 있을 때 사용하는 예약어

### 2) React에서 링크 연결

- (ex) import { Link } from "react-router-dom";

```
<Link to="/day">Day {day.day}</Link>
```

- **useParams** : URL 경로의 값을 파라미터로 받을 때 사용하는 Hook

└ (사용방법)

(1) <Route> 태그에서 파라미터 설정

[ex] <Route path="/day/:day"> </Route>

(2) 변수에 파라미터값 저장

[ex] const a = useParams();

< 변수 a에 파라미터 **:day**의 값을 저장  
**{day : "data"}**의 형태로 저장됨



## 8. json-server, REST API

- npm install -g json-server 설치

### 1) json-server를 이용한 API

- `json-server --watch [api로 만들 json파일의 경로] --port [임의의 포트번호]`

### 2) REST API

- Create : POST
- Read : GET
- Update : PUT
- Delete : DELETE

## 9. useEffect, fetch()

- 더미 데이터를 구현하는 부분에서 더미 데이터를 DB/json으로 바꿔주는 구간

### 1) useEffect

- 어떠한 상태값이 바뀌고 렌더링이 끝나는 순간 동작하는 Hook
- 형식 : `useEffect(function, array);`
- (예시)

(1) 

```
useEffect(() => {  
  console.log("Count change");  
});
```

< 렌더링이 끝나는 모든 순간마다 동작

(2) 

```
useEffect(() => {  
  console.log("Count change");  
}, [count]);
```

< 배열 내 count 값이 변할 때만 동작

(3) 

```
useEffect(() => {  
  console.log("Count change");  
}, []);
```

< 페이지 렌더링 직후 최초 1회 실행

### 2) fetch() : API 비동기 통신 시 사용

- 형식 : `fetch("api url") / fetch(`api url?value=${useParams data}`)`
- (예시)

(1) 

```
useEffect(() => {  
  fetch("http://localhost:3001/days")  
    .then(res => {  
      return res.json();  
    })  
    .then(data => {  
      setDays(data);  
    });  
}, []);
```



(2)

```
useEffect(() => {
  fetch(`http://localhost:3001/words?day=${day}`)
    .then(res => {
      return res.json();
    })
    .then(data => {
      setWords([data]);
    });
}, [day]);
```

#### - 부연설명

- (i) fetch() 함수까지
  - fetch()를 통해 가져온 데이터는 json 형식의 데이터가 아님
- (ii) 첫 번째 .then() 함수까지
  - 그 데이터를 .json() 함수를 통해 json 형식으로 변환시켜 반환
- (iii) 두 번째 .then() 함수까지
  - 가공된 최종 데이터를 이용하여 후속 처리
- (iv) 의존성 배열 부분까지
  - api를 받을 때 최초 한번만 실행하면 되기에 위 useEffect 파트 예시(3) 형식을 사용하면 되지만, fetch() 파트 예시(2)와 같이 url 경로에 `${value}` 형식처럼 사용될 경우, 의존성 배열 안에 해당 변수를 적어준다.

## 10. Custom Hooks

- 유사한 동작을 하는 코드를 분리하여 사용자가 새로운 hook을 만들 수가 있다.

### 1) hooks > useFetch.js

- "useXXX"의 형식으로 hook을 작성
- 타 프로그래밍 언어의 함수처럼 코드를 간결하게 표현 가능

## 11. PUT(수정), DELETE(삭제)

### - DB 사용할거임

## 12. POST(생성), useHistory()

### - DB 사용할거임

## 13. 타입스크립트

- 굳이 써야 되는 건가... 라는 생각?
- Java 실제 프로젝트에서도 문제 많다고 interface 잘 안 쓰단 말이지 물론 쓰는 곳이야 쓰긴 하겠지만