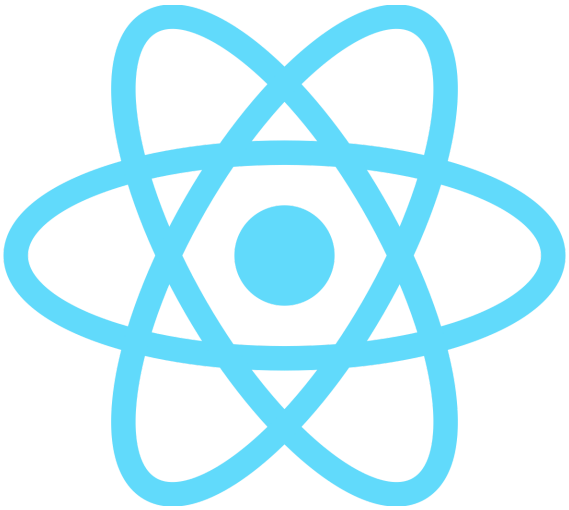
230109_2반_실습



JSX

JSX 소개

JSX란?

```
const element = <h1>Hello, world!</h1>;j
```

```
// 실제 작성할 JSX 예시
function App() {
  return (
      <h1>Hello, GodDaeHee!</h1>
    );
}

// 위와 같이 작성하면, 바벨이 다음과 같이 자바스크립트로 해석하여 준다.
function App() {
  return React.createElement("h1", null, "Hello, GodDaeHee!");
}
```

- JSX(JavaScript XML)는 Javascript에 XML을 추가하여 확장한 문법이다.
- JSX는 리액트로 프로젝트를 개발할 때 사용되므로 공식적인 자바스크립트 문법은 아니다.

- JSX는 **React “엘리먼트(element)”** 를 생성한다. React 엘리먼트는 브라우저 DOM 엘리먼트와 달리 일반 **객체**이다.
- React는 JSX 사용이 필수가 아니지만, JS 코드 안에서 UI관련 작업을 할 수 있기 때문에 시각적으로 더 도움이 된다. 또한 JSX를 사용하면 React가 더욱 도움이 되는 에러 및 경고 메시지를 표시할 수 있게 해준다.
- 장점
  1. 보기 쉽고 익숙하다.
     - JSX는 HTML 코드와 비슷하기 때문에 일반 자바스크립만 사용한 코드보다 더 익숙하며 가독성이 좋다.
  2. 높은 활용도
     - JSX에는 div, span 같은 HTML 태그를 사용할 수 있으며, 개발자가 만든 컴포넌트도 JSX 안에서 작성할 수 있다.

> JSX란? - React 공식웹사이트
>
> React에서는 본질적으로 렌더링 로직이 UI 로직(이벤트가 처리되는 방식, 시간에 따라 state가 변하는 방식, 화면에 표시하기 위해 데이터가 준비되는 방식 등)과 연결된다는 사실을 받아들입니다. React는 별도의 파일에 마크업과 로직을 넣어 기술을 인위적으로 분리하는 대신, 둘 다 포함하는 “컴포넌트”라고 부르는 느슨하게 연결된 유닛으로 관심사를 분리합니다. 이후 섹션에서 다시 컴포넌트로 돌아오겠지만, JS에 마크업을 넣는 게 익숙해지지 않는다면 이 이야기가 확신을 줄 것입니다. React는 JSX 사용이 필수가 아니지만, 대부분의 사람은 JavaScript 코드 안에서 UI 관련 작업을 할 때 시각적으로 더 도움이 된다고 생각합니다. 또한 React가 더욱 도움이 되는 에러 및 경고 메시지를 표시할 수 있게 해줍니다.

JSX 사용법

표현식

- JSX 안에 자바스크립트 표현식 을 중괄호로 묶어서 포함시킬 수 있습니다.
- 자바스크립트 표현식을 작성하려면 JSX내부에서 코드를 { }로 감싸주면 된다.

```
function App() {
  const name = 'Snoopy';
  return (
    <div>
      <div>Hello</div>
      <div>{name}!</div>
    </div>
  );
}
```

```
const name = "홍길동";
const nameDetail = {
    firstName: "홍",
    lastName: "길동",
};

const element = <h1>성은 {nameDetail.firstName}이요,
이름은 {nameDetail.lastName}, 전체 이름은 {name}</h1>;

ReactDOM.render(element, document.getElementById("root"));
```

닫힌 코드

- **반드시 부모 요소 하나가 감싸는 형태여야 한다.**
  - Virtual DOM에서 컴포넌트 변화를 감지할 때 때 효율적으로 비교할 수 있도록 컴포넌트 내부는 하나의 DOM 트리 구조로 이루어져야 한다는 규칙이 있기 때문이다.
  - 잘못된 코드
    - 예제1

```
function App() {
  return (
    <div>Hello</div>
    <div>World!</div>
  );
}
```

    - 예제2

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <Hello />
```

```
    <div>안녕히계세요.</div>
  );
}

export default App;
```

- 바른 코드
  - 예제1

```
// div를 사용 하였기 때문에 스타일 적용시 작성한 코드를 div로 한번 더 감쌌다는 부분을 고려해야
 한다.
function App() {
  return (
    <div>
      <div>Hello</div>
      <div>World!</div>
    </div>
  );
}
```

```
function App() {
  return (
    <Fragment>
      <div>Hello</div>
      <div>World!</div>
    </Fragment>
  );
}
```

```
function App() {
  return (
    <>
      <div>Hello</div>
      <div>World!</div>
    </>
  );
}
```

  - 예제2

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
```

```
      <Hello />
      <div>안녕히계세요</div>
    </div>
  );
}

export default App;
```

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <>
      <Hello />
      <div>안녕히계세요</div>
    </>
  );
}

export default App;
```

- `<div>`, `<p>`, `<span>`, `<a>` 같이 짝이 있는 태그의 경우 반드시 닫는 태그가 존재해야 한다. 그렇지 않을 경우 에러가 발생한다.
- `<img/>`, `<input/>`, `<br/>` 같은 단독 태그(self-closing tag)의 경우에는 반드시 태그를 닫아줘야 한다. 그렇지 않을 경우 에러가 발생한다.
  - 잘못된 코드

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
      <div>
    </div>
  );
}

export default App;
```

```
import React from 'react';
import Hello from './Hello';
```

```
function App() {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
      <input>
      <br>
    </div>
  );
}

export default App;
```

- 바른 코드

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
      <input />
      <br />
    </div>
  );
}

export default App;
```

style과 className

- 일반 HTML에서 CSS 클래스를 사용할 때에는 class 라는 속성을 사용한다.
- JSX에서는 class가 아닌 className 을 사용한다.
- jsx에서 style를 작성할 때로 "-"가 아니라 camelCase를 사용해야 합니다
  - 예제1

```
function App() {
    return (
        <>
            <h1 class="abc">{title}</h1> <--// X 오류 발생
            <h1 className="abc">{title}</h1> <--// O 정상 작성
            <Desc />
            <Body />
        </>
```

```
    );
}
```

- 예제2

App.css

```
.gray-box {
  background: gray;
  width: 64px;
  height: 64px;
}
```

App.js

```
import React from 'react';
import Hello from './Hello';
import './App.css';


function App() {
  const name = 'react';
  const style = {
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: 24, // 기본 단위 px
    padding: '1rem' // 다른 단위 사용 시 문자열로 설정
  }

  return (
    <>
      <Hello />
      <div style={style}>{name}</div>
      <div className="gray-box"></div>
    </>
  );
}

export default App;
```

- style 속성을 사용한다.
  - 예제

```
import React from 'react';
import Hello from './Hello';

function App() {
  const name = 'react';
```

```
  const style = {
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: 24, // 기본 단위 px
    padding: '1rem' // 다른 단위 사용 시 문자열로 설정
  }

  return (
    <>
      <Hello />
      <div style={style}>{name}</div>
    </>
  );
}

export default App;
```

주석

- JSX 내에서 {/…/} 와 같은 형식을 사용 한다.

```
function App() {
  return (
    <>
      {/* 주석사용방법 */}
      <div>Hello, GodDaeHee!</div>
    </>
  );
}
```

- 시작태그를 여러줄 작성시에는, 내부에서 // 의 형식을 사용할 수 있다.

```
function App() {
  return (
    <>
      <div
      // 주석사용방법
      >Hello, GodDaeHee!</div>
    </>
  );
}
```

Function Component

- 다양한 Function 형태로 컴포넌트화할 수 있습니다.

```
const title = "기능 컴포넌트화하기";
function Desc() {
```

```
    return (
        <h2>Function 형태로 컴포넌트화할 수 있습니다.</h2>
    );
}
const Body = () => <p> 본문 내용이 나오는 영역입니다. </p>;
const Footer = () => {return <p> Footer 영역입니다.</p>;};

// 함수형 컴포넌트
function App() {
    return (
        <>
            <h1>{title}</h1>
            <Desc></Desc>
            <Body />
            <Footer />
        </>
    );
}

ReactDOM.render(<App />, document.getElementById("root"));
```

Class Component

- Class 형태로도 컴포넌트화할 수 있습니다.

```
const title = "기능 컴포넌트화하기";
function Desc() {
    return (
        <h2>Function 형태로 컴포넌트화할 수 있습니다.</h2>
    );
}
const Body = () => <p> 본문 내용이 나오는 영역입니다. </p>;
const Footer = () => {return <p> Footer 영역입니다.</p>;};

// 클래스 형 컴포넌트
class App extends React.Component {
    render() {
        return (
          <>
            <h1>{title}</h1>
            <Desc></Desc>
            <Body />
            <Footer />
          </>
        );
    }
}

ReactDOM.render(<App />, document.getElementById("root"));
```

Props 사용하기

- Props를 이용하여 컴포넌트에 값을 넘길 수 있습니다.

```
const Body = (props) => {
    return
        <>
          <p> 제목 : {props.title}</p>;
          <p> 내용 : {props.desc}</p>;
        </>
};

function App() {
    return (
        <Body title="제목입니다." desc="설명입니다." />
    );
}
```

구조 분해 할당 (튜플 분해)

- props를 넘길 때 구조를 분해해서 할당할 수도 있습니다.

```
const Body = ({ title, desc }) => {
    return
        <>
            <p> 제목 : {title}</p>;
            <p> 내용 : {desc}</p>;
        </>
};

function App() {
    return (
        <>
          <Body title="제목입니다." desc="설명입니다." />
        </>
    );
}
```

조건(3항) 연산자 사용하기

- jsx의 표현식을 활용할 때 if 문은 사용할 수 없고 대신에 3항 연산자로 사용할 수 있습니다. (3항 연산자란 일반적인 if 문을 축약한 형태입니다.)
- JSX 내부의 JS 표현식에서는 if문을 사용할 수 없다. 때문에 조건에 따라 다른 내용을 렌더링 하고자 할 경우 JSX 밖에서 if 문을 사용하거나, 중괄호 안에서 삼항 연산자를 사용하면 된다.

```
// 일반 if 문.
if (yn === true) {
    return <Body title="제목입니다." desc="설명입니다." />;
```

```
} else {
    return <Body title="없습니다." desc="아닙니다." />;
}

// 3항 연산자
{
    yn === true ? <Body title="제목입니다." desc="설명입니다." /> : <Body title="없습니다." d
esc="아닙니다." />;
}

// AND 연산자 : true 면 실행, false 면 아무것도 없음.
{
    yn === true && <Body title="제목입니다." desc="설명입니다." />;
```

```
class App extends Component {
    render() {
        let name = 'React';
        return (
            <div>
                {
                    name === 'React' ? (
                        <h1>This is React.</h1>
                    ) : (
                        <h1>This is not React.</h1>
                    )
                }
            </div>
        );
    }
}
```

- AND 연산자(&&)를 사용한 조건부 렌더링
    - 특정 조건을 만족할 때만 내용을 보여주고 싶을 때 사용

```
class App extends Component {
    render() {
        let name = 'React';
        return (
            <div>
                {
                    name === 'React' && <h1>This is React.</h1>
                }
            </div>
        );
    }
}
```

- OR 연산자(||)를 사용한 조건부 렌더링

- 리액트 컴포넌트에서는 함수에서 undefined나 null을 반환하면 렌더링을 하려하면 오류가 발생한다. 반면 JSX 내부에서 undefined나 null을 렌더링하는 것은 괜찮다.
- JSX 내부에서 undefined나 null을 렌더링하면 아무것도 보여주지 않는다.
- OR 연산자는 AND 연산자와 다르게 특정 값이 undefined나 null일 경우 보여주고 싶은 문구가 있을 때 주로 사용한다.

```
class App extends Component {
    render() {
        let name = undefined;
        return (
            <div>
                {
                    name === 'React' || <h1>This is React.</h1>
                }
            </div>
        );
    }
}
```