A. class vs functional component

> - 컴퍼넌트를 선언하는 방식은 `함수형 컴퍼넌트와 클래스형 컴퍼넌트` 가 있다.
> - 차이점은 클래스형일 경워 `state, life cyle기능 사용 및 임의의 method()를 정의할 수 있다`
> - ES6이전에는 자바스크립트에는 class가 없었다. 그 것을 구현하려면 prototype이라는 문법을 사용
> - 클래스형에는 `반드시 render()함수가 있어야 한다`

1. 함수형의 장점
   - 클래스형보다 선언하기가 편하다.
   - 메모리자원도 적게 사용
   - 프로젝트를 빌드한 후에 배포시 파일크기가 작다.
2. 함수형의 단점
   - state와 life cycle API사용이 불가
   - 이 단점은 V16.8이후에 `Hooks기능이 도입되면서 해결`
   - 리액트 공식문서에서는 `함수형컴퍼넌트와 Hooks를 사용하도록 권고`

B. Class Componet작성

1. Sample Code

```
src/MyComponent.js
const MyComponent = () => {
    return <div>새로운 컴퍼넌트!!!</div>
}

export default MyComponent;
```

- `export default` 이 코드는 다른 파일에서 이 파일을 import할 때 불러올 수 있도록 정의

```
src/App.js
import MyComponent from './MyComponent'

const App = () {
    return <MyComponent />
}

export default App;
```

2. props

- props는 properties의 약어
- 컴퍼넌트 속성을 설정할 때 사용하는 요소

- props값은 해당 컴포넌트를 불러와 사용하는 부모 컴퍼넌트에서 설정할 수 있다.

2.1 컴퍼넌트를 사용할 때 props값 지정하기

```
src/App.js
import MyComponent from './MyComponent'

const App = () {
    return <MyComponent name="React" />
}

export default App;

src/MyComponent.js
const MyComponent = props => {
    return <div>제 이름은 {props.name} 입니다.</div>
}

export default MyComponent;
```

2.2 defaultProp

```
src/App.js
import MyComponent from './MyComponent'

const App = () {
    return <MyComponent />
}

MyComponent.defaultProps = {
  name: "홍길동",
};

export default App;

src/MyComponent.js
const MyComponent = props => {
    return <div>제 이름은 {props.name} 입니다.</div>
}

export default MyComponent;
```

2.3 children : tag사이의 내용을 조회

```
src/App.js
import MyComponent from './MyComponent'

const App = () {
     return <MyComponent>리액트 입니다!!!</MyComponent>;
}

MyComponent.defaultProps = {
  name: "홍길동",
};
```

```
export default App;

src/MyComponent.js
const MyComponent = (props) => {
  return (
    <div>
      내 이름은 {props.name} 입니다.!!! <br />
      children 값 = {props.children}
    </div>
  );
};
MyComponent.defaultProps = {
  name: "홍길동",
};

export default MyComponent;
```

2.4 비구조화 할당 문법

```
src/MyComponent.js
const MyComponent = (props) => {

  const { name, children } = props;
  return (
    <div>
      내 이름은 {props.name} 입니다.!!! <br />
      children 값 = {props.children}
    </div>
  );
};
MyComponent.defaultProps = {
  name: "홍길동",
};

export default MyComponent;
```

- 객체에서 값을 추출하는 문법을 `비구조화 할당(destructuring assignment)`라고 한다.
- 이 문법은 `구조분해문법`디라고도 하며 함수의 파라미터 부분에서도 사용할 수 있다.
- 만약, 함수 파마리터가 객체라면 그 값을 비구조화해서 사용할 수 있다.

```
src/MyComponent.js

 아래 부분 생략...
```

2.5 propTypes를 통한 props 검증

- 컴퍼넌트의 필수 props를 지정하거나 `props의 타입을 지정할 때는 propsType를 사용`
- 문자열로 지정할 경우 문자열이 아닌 값이 전달되면 console에 경고창 표시
  - version에 따라 경고표시 안 나올 수도 있음

```
src/App.js
import MyComponent from './MyComponent'

const App = () {
  return <MyComponent name={3}>propTyps : 값을 문자타입 아닌 것으로
전달하면 에러!!!</MyComponent>;

}

src/MyComponent.js
import PropTypes from 'prop-types'
const MyComponent = ({ name, children }) => {

  return (
    <div>
      내 이름은 {name} 입니다.!!! <br />
      children 값 = {children}
    </div>
  );
};

 MyComponent.prototypes = {
  name: PropTypes.string
 }
```

2.6 isRequred

- 참고사이트 :
  - https://www.npmjs.com/package/prop-types
  - https://github.com/facebook/prop-types
    - npm repo prop-types

```
src/App.js
import MyComponent from './MyComponent'

const App = () {

 return <MyComponent name={'손흥민'}>children</MyComponent>;

}

src/MyComponent.js
import PropTypes from 'prop-types'
const MyComponent = ({ name, children }) => {

  return (
    <div>
      내 이름은 {name} 입니다.!!! <br />
      children 값 = {children}
    </div>
  );
};

 MyComponent.prototypes = {
  name: PropTypes.string,
```

```
 addr: PropTypes.string.isRequred
}
```