컴포넌트

- 부품, 구성요소
- components폴더 생성
    - Welcome.js파일 생성
        - Welcome.js > App.js > index.js > index.html
- 컴포넌트명 관례: 첫글자 대문자

종류

1. 클래스형: old style
    - class ~~~
    - 유지보수를 위해 학습해야 할 수도
2. 함수형: new style
    - function ~~~

컴포넌트 구현시 기억해야 할 것

1. 재사용성(생산성)
    - component
2. 재료(값): 데이터
    - props (properties)
3. 검증
    - props validation

사용자 인터페이스 구현시

=> 사용된 기술에 상관없이 각 부품을 만들어 조립 => 컴포넌트 단위

개념은 모두 똑같다!

- Vuejs vs Reactjs: 구현 방식의 차이
- 컴포넌트의 개념은 유사

- 붕어빵
    - 컴포넌트: 붕어빵 틀
    - 밀가루+앙코(팥,슈크림)
    - 시식? 재료에 흙과 같이 먹지 못하는 것인지 체크
- 유튜브 사이트
    - 동영상한개표시하는컴포넌트
    - 각 동영상 정보
    - 동영상 정보가 정당한지 체크

실습 1

- components 폴더내에서 작업
- 클래스형 컴포넌트 vs 함수형 컴포넌트
- 함수형 컴포넌트
- function 컴포넌트명(props){}
- const 컴포넌트명 = function(props){}
- const 컴포넌트명 = (props)=>{}

```
<Welcome.js>
// [1]
function Welcome{
  <p>Welcome to Korea</p> //---->에러 - 아무표시안됨
}
export default Welcome;

<App.js>
import './App.css';
import Welcome from './components/Welcome'; // 자동으로 입력됨

function App() {
  return (
    <div className="App">
        <Welcome />
    </div>
  );
}

export default App;
```

--- 실습

```
<Welcome.js>
// [1]
function Welcome{
  return <p>Welcome to Korea</p> // 수정 첫번째
  return( // 수정 두번째
    <p>
      Welcome to Korea
    </p> // 수정
  )
}
export default Welcome;
```

```
<App.js>
import './App.css';
import Welcome from './components/Welcome'; // 자동으로 입력됨
```

```
function App() {
  return (
    <div className="App">
        <Welcome />
        <Welcome></Welcome>
        <Welcome />
    </div>
  );
}

export default App;
```

컴포넌트, props

- props: 컴포넌트 생성의 재료

```
<p>Welcome to Seoul (인구 수 :  940 만명)</p>
             재료1            재료2
```

- App.js
    - `<Welcome city="Seoul" population="940"/>`
    - `<Welcome city="Seoul" population="940">환영합니다. </Welcome>`

```
<Welcome.js>
function Welcome( props ) {
     console.log( typeof( props ))
     console.log( props )
     return <p>{props.children} Welcome to { props.city } (인구 수 : {
props.population } 만명)</p>
    }
```

- App.js
    - `<Welcome city="Jeju" population="60"/>`
    - `<Welcome city="Suwon" population="120"/>`

실습 2

```
<Welcome.js>
// [2]
const Welcome = function( props ) {
    return <p>Welcome to { props.city } (인구 수 : { props.population } 만명)</p>
}
export default Welcome;
```

```
<Welcome.js>
// [3] Arrow Function 선호
const Welcome = props => {//const Welcome = (props) => {
    return <p>Welcome to { props.city } (인구 수 : { props.population } 만명)</p>
}
export default Welcome;
```

```
<Welcome.js>
import { Fragment } from "react";
// [4] 선호
export default function Welcome( props ) {
    return (
        <Fragment> // or <>, div태그는 렌더링 되지만, Fragment 등은 렌더링에는 사용하지 않음
            <p>Welcome to { props.city } (인구 수 : { props.population } 만명)</p>
        </Fragment> //or </>
        // ---> 더바람직, 의미없는 div태그 추가 안함
    )
}
```

- Fragment사용하면 의미없는 태그의 삽입 방지
- import 필요
- 축약형: <></>

- Box 컴포넌트
  - inline style 적용
    - style={{backgroundColor:props.color}}

```
<Box.js>
const Box = ( props ) => {
  return (
    <>
      <div
      className="box"
      style={{ backgroundColor: props.color }}
      >
        { props.name }
      </div>
    </>
  )
}

export default Box;

<App.js>
import './App.css';
import Box from './components/Box';
```

```
import Welcome from './components/Welcome'; // 자동으로 입력됨

function App() {
  return (
    <div className="App">
        <Welcome city="Seoul" population="940"/>
        <Welcome city="Jeju" population="60"/>
        <Welcome city="Suwon" population="120"/>
        <Box name='blue box' color='skyblue'/>
        <Box name='blue box' color='rgb(125,221,13)'/> 가능?
    </div>
  )
}

<App.css> 추가
.box{
    border:1px solid;
    width:100px;
    height:100px;
    //background-color:skyblue;
}
```

props 디폴트 값

```
<Box2.js>
const Box2 = ( props ) => {
    return (
        <>
            <div
                className="box"
                style={{ backgroundColor: props.color }}
            >
                { props.name }
            </div>
        </>
    )
}

Box2.defaultProps = {
    color: 'yellow',
    name: '노랑 박스'
}

export default Box2;

<App.js>
- <Box2 name='blue box' color='skyblue'/>
- <Box2 />
```

props 값 구조분해할당 적용

- Destructuring Assignment
- App.js에서 사용시 -> 매개변수를 전달받을 때
- --> 객체로 처음부터 받음
- props대신 --> {color, name}변경

```
<Box3.js>
const Box3 = ( { color, name } ) => {
   return (
    <>
        <div className="box" style={{ backgroundColor : color, color }}>
                        // color:color
            { name }
        </div>
    </>
   )
}

export default Box3;

<Welcome2.js>
export default function Welcome2( { city, population } ) {
   return (
    <>
        <p>Welcome to { city } (인구 수 : { population })</p>
    </>
   )
}


<App.js>
<Welcome2 city="Seoul" population="940"/>
<Welcome2 city="Jeju" population="60"/>
<Welcome2 city="Suwon" population="120"/>
```

props의 children

```
<App.js>
import './App.css';
import Box2 from './components/Box2';
import Wrapbox from './components/Wrapbox';

function App() {
  return (
      <Wrapbox>
        <Box2 name='파랑박스' color='skyblue'/>
        <Box2 name='빨간박스' color='red'/>
        <Box2 name='노랑박스' color='yellow'/>
```

```
      </Wrapbox>

  )
}

<Wrapbox.js>
// props.children
// 1. 컴포넌트간 합성 가능
//2. 하위(자식) 컴포넌트나 HTML li와 같은
//   엘리먼트가 얼마나 있는지 모르는 상황에서
//   화면에 표시하기 위해 사용
const Wrapbox = ( {children} ) => {
    return (
        <div>
            { children }
        </div>
    )
}
/*
children은
        <Box2 name='파랑박스' color='skyblue'/>
        <Box2 name='빨간박스' color='red'/>
        <Box2 name='노랑박스' color='yellow'/>
내용포함
*/
export default Wrapbox;


<Wrapbox.js>

const Wrapbox = ( props ) => {
    return (
        <div>
            { props.children }
        </div>
    )
}
export default Wrapbox;

<App.js>
import './App.css';
import Box2 from './components/Box2';
import Wrapbox from './components/Wrapbox';

function App() {
  return (
      <Wrapbox>
        <Box2 name='파랑박스' color='skyblue'/>
        <Box2 name='빨간박스' color='red'/>
        <Box2 name='노랑박스' color='yellow'/>
        <li>aaa</li>
        <li>bbb</li>
        <li>ccc</li>
      </Wrapbox>
```

```
    )
}
```

컴포넌트 내부에서 스타일 적용하는 방법 (콧수염 2개 {{}}의 비밀?)

- 외부{}: JSX 문법 {}
- 내부{}: {객체표현}

```
<Box2.js>
const Box2 = ( props ) => {
  const = styleObj = { backgroundColor: props.color };
    return (
        <>
            <div
                className="box"
                style={styleObj}
            >
                { props.name }
            </div>
        </>
    )
}

<Box2.js>
const Box2 = ( props ) => {
  const = styleObj = { backgroundColor: props.color,
  border:'10px solid blue',};
    return (
        <>
            <div
                className="box"
                style={styleObj}
            >
                { props.name }
            </div>
        </>
    )
}
```

- 참고: styled-components
  - npm install styled-components
  - `export const 컴포넌트명 = styled.div` css 코드 `;`
  - `const LogoContainer = styled(자식컴포넌트?)` css코드 `