Component

1. 컴포넌트 만들기

함수 컴포넌트와 클래스 컴포넌트

리액트에서는 컴포넌트를 클래스형과 함수형 두 가지로 정의할 수 있습니다. 클래스형의 경우, ES6 class를 사용하여 컴포넌트를 정의할 수 있습니다.

`React.Component` 클래스를 상속받는 `App` 클래스를 다음과 같이 만들 수 있습니다. 클래스형 컴포넌트의 경우 반드시 `render()` 함수가 있어야 합니다. 그리고 `render()` 함수에서는 반드시 JSX를 리턴해야 합니다.

```
import React from "react";

class App extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
export default App;
```

다음은 JavaScript 함수를 이용한 함수형 컴포넌트입니다.

```
function App() {
  return <h1>Hello, React!</h1>;
}
export default App;
```

React의 관점에서 볼 때 함수형/클래스형 컴포넌트는 동일합니다. 하지만 최신 리액트에서는

- 클래스 사용법이 어렵고 복잡하기 때문에
- 함수형 컴포넌트와 리액트 Hook의 조합이 더 사용하기 간편하기 때문에
- 코드가 훨씬 간결하기 때문에

점차 인기가 많아지고 있는 추세입니다. 본 강의에서도 함수형 컴포넌트를 사용할 것입니다.

> 💡 컴포넌트의 이름은 항상 대문자로 정의해야 합니다. 예를 들어 `<div />`는 HTML div 태그로 인식되지만, `<Welcome />`은 컴포넌트로 인식됩니다.

export

해당 모듈에서 기본적으로 내보낼 값 하나를 정합니다. 만일 다른 모듈에서 해당 모듈을 import한다면 `export default` 에 정의된 객체가 참조됩니다. 이때 import 되는 객체의 이름은 임의로 정할 수 있습니다.

```
export default App;

-->

import App from "App.js"
import MyApp from "App.js"
```

만일 한 개의 객체가 아닌, 여러 개를 내보내고 싶은 경우 아래와 같이 할 수 있습니다. 이때 다른 모듈에서는 원하는 값만 가져올 수 있습니다.

```
export {Var1, Var2};


-->

import {Var1} from "App.js"
import {Var1, Var2} from "App.js"
```

2. props

props는 properties의 줄임말로, 어떤 값을 컴포넌트에 전달할 때 사용합니다. `Hello` 라는 컴포넌트에 `name` 이라는 변수를 전달해주고 싶다면 다음과 같이 하면 됩니다. `name` 이라는 props에 `"indo"` 라는 값을 전달하고 있습니다.

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

function App() {
  return (
    <Hello name="indo" />
  );
}

export default App;
```

`Hello` 컴포넌트에서는 다음과 같이 props에 전달된 값을 사용할 수 있습니다.

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

function Hello(props) {
  return <h1>Hello, {props.name}</h1>
}

export default Hello;
```

`props` 라는 파라미터에 속한 변수를 `props.name` 으로 참조하고 있는 것을 볼 수 있습니다.

비구조화 할당과 기본값

파라미터를 참조할 때마다 `props.` 를 붙이지 않도록 "비구조화 할당"을 사용하면 더 직관적으로 파라미터를 이해할 수 있어서 편리합니다. 만일 특정 파라미터의 기본값을 설정하고 싶다면 `defaultProps`를 사용하면 됩니다. `파라미터: 값` 과 같이 사용합니다.

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

function Hello({name}) {
  return <h1>Hello, {name}</h1>
}
```

```
export default Hello;
```

```
Hello.defaultProps = {
  name: 'NoName'
}
```

실수하기 쉬운 Props 사용법

props 방향

props를 사용할 때 반드시 기억해야 하는 점은, props는 “부모 컴포넌트”에서 “자식 컴포넌트”로 보내주는 값이라는 점입니다. 따라서 부모 컴포넌트가 보내준 값을 자식 컴포넌트에서 변경하더라도, 원래 부모에서 보내준 값이 바뀌지 않습니다.

감싸는 컴포넌트를 만드는 경우, props.children 사용하기

다음과 같이 `Hello` 컴포넌트를 감싸는 `Wrapper` 컴포넌트를 만들고 싶을 때가 있습니다.

```
// App.js
import React from 'react';
import Hello from './Hello';
import Wrapper from './Wrapper';

function App() {
  return (
    <Wrapper>
      <Hello name="react" color="red"/>
      <Hello color="pink"/>
    </Wrapper>
  );
}

export default App;
```

`Wrapper` 컴포넌트는 아래와 같이 설계합니다.

```
import React from 'react';

function Wrapper() {
  const style = {
```

```
    border: '2px solid black',
    padding: '16px',
  };
  return (
    <div style={style}></div>
  )
}

export default Wrapper;
```

그러면 화면에 `Wrapper` 컴포넌트는 보이는데, 하위 엘리먼트들이 나타나지 않습니다.

`Wrapper` 컴포넌트가 사이에 오는 엘리먼트들을 렌더링하도록 하려면 `{children}` 을 넣어줘야 합니다. 자주 실수하는 부분이기 때문에 내가 정의한 컴포넌트로 다른 컴포넌트를 감싸는 경우 `props.children` 이 필요하다는 사실을 꼭 기억하세요.

```
import React from 'react';

function Wrapper({ children }) {
  const style = {
    border: '2px solid black',
    padding: '16px',
  };
  return (
    <div style={style}>
      {children}
    </div>
  )
}

export default Wrapper;
```