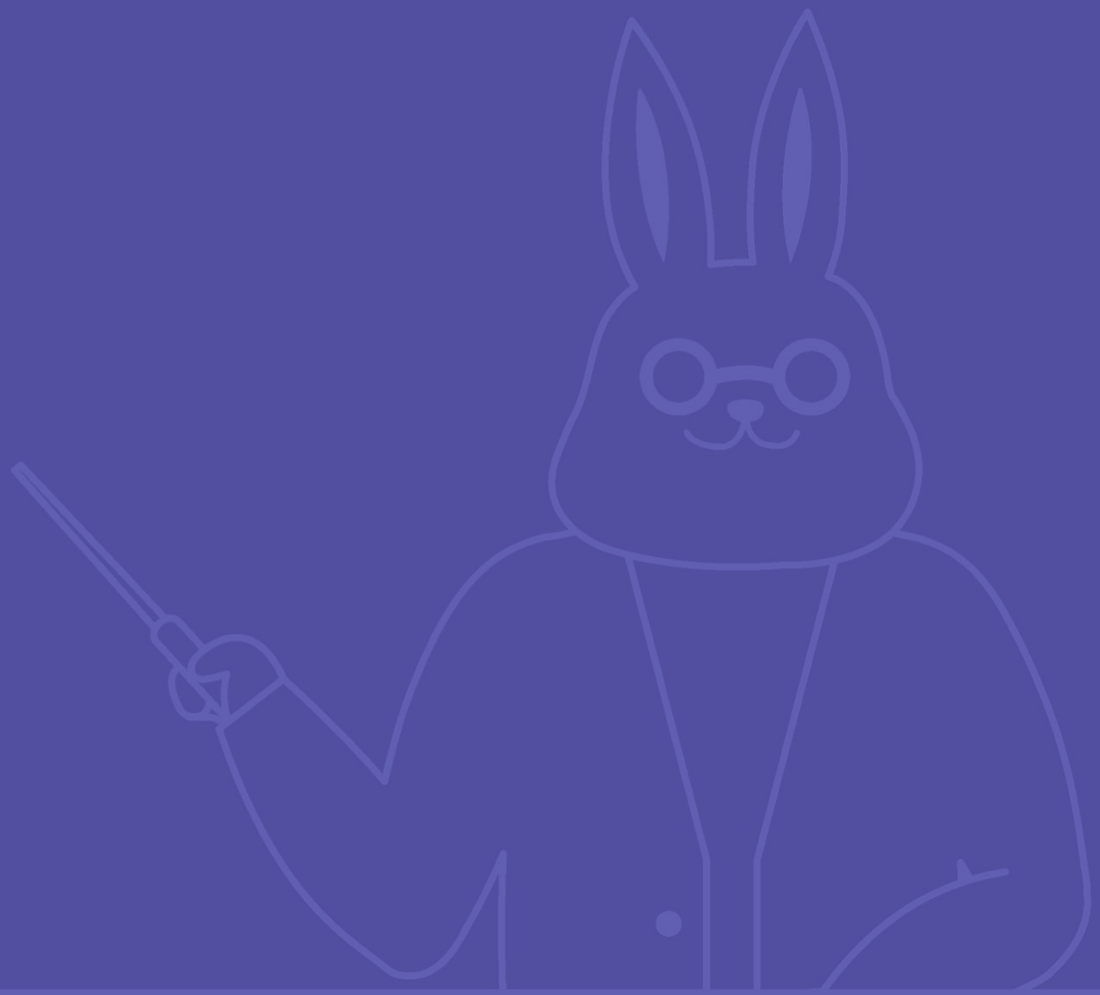
Component

01 인스타그램 클론코딩

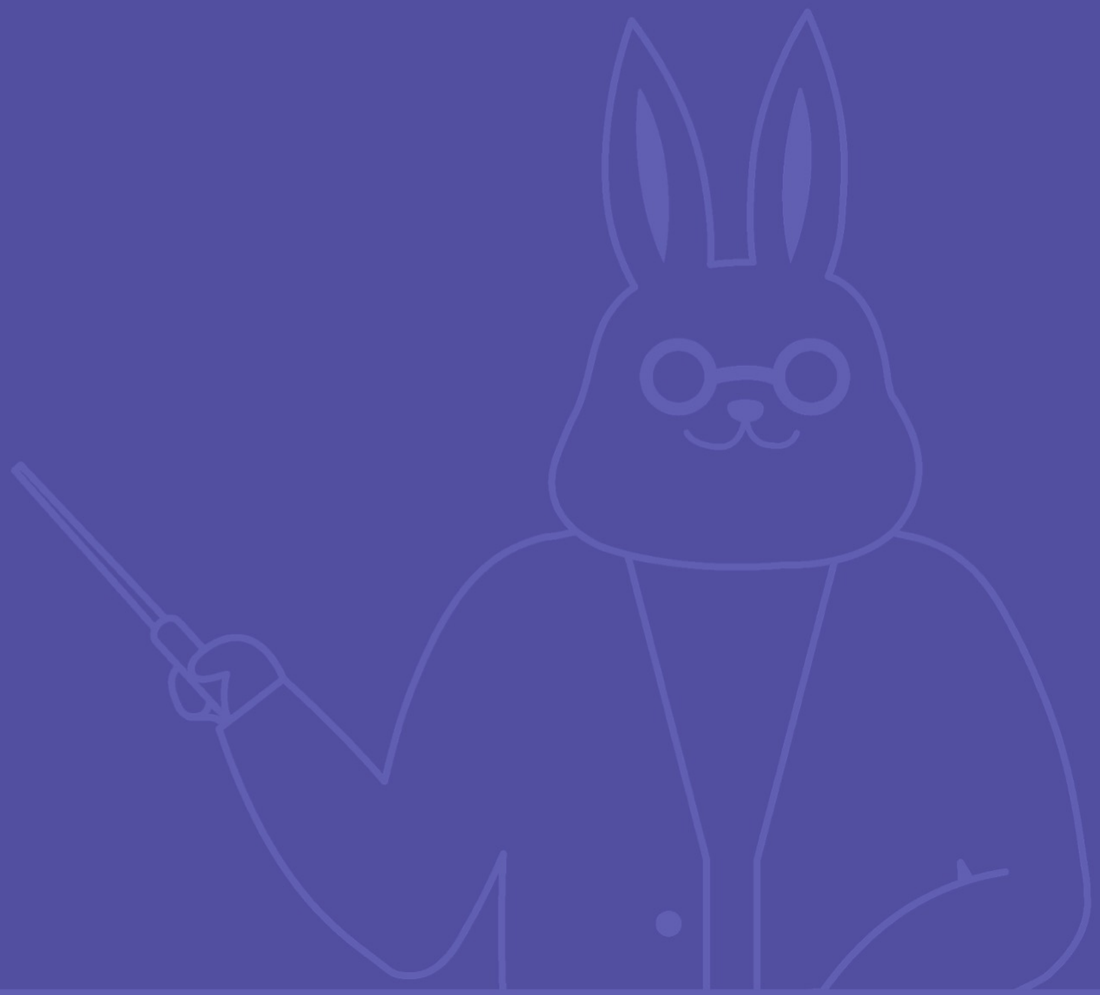


/* elice */

목차

01

컴포넌트 정의



컴포넌트의 개념

컴포넌트의 개념

반복, 재사용 불가

덩어리 일 경우 복잡하고 규모가 큰 설계 불가능

컴포넌트의 개념

분리해서 설계

컴포넌트를 활용하여 엘리먼트를 독립적으로 만들어줄 수 있습니다.
= 엘리먼트의 재사용성을 높여줍니다

컴포넌트란?

컴포넌트 =

render()를 통해 HTML요소를 반환하는 함수

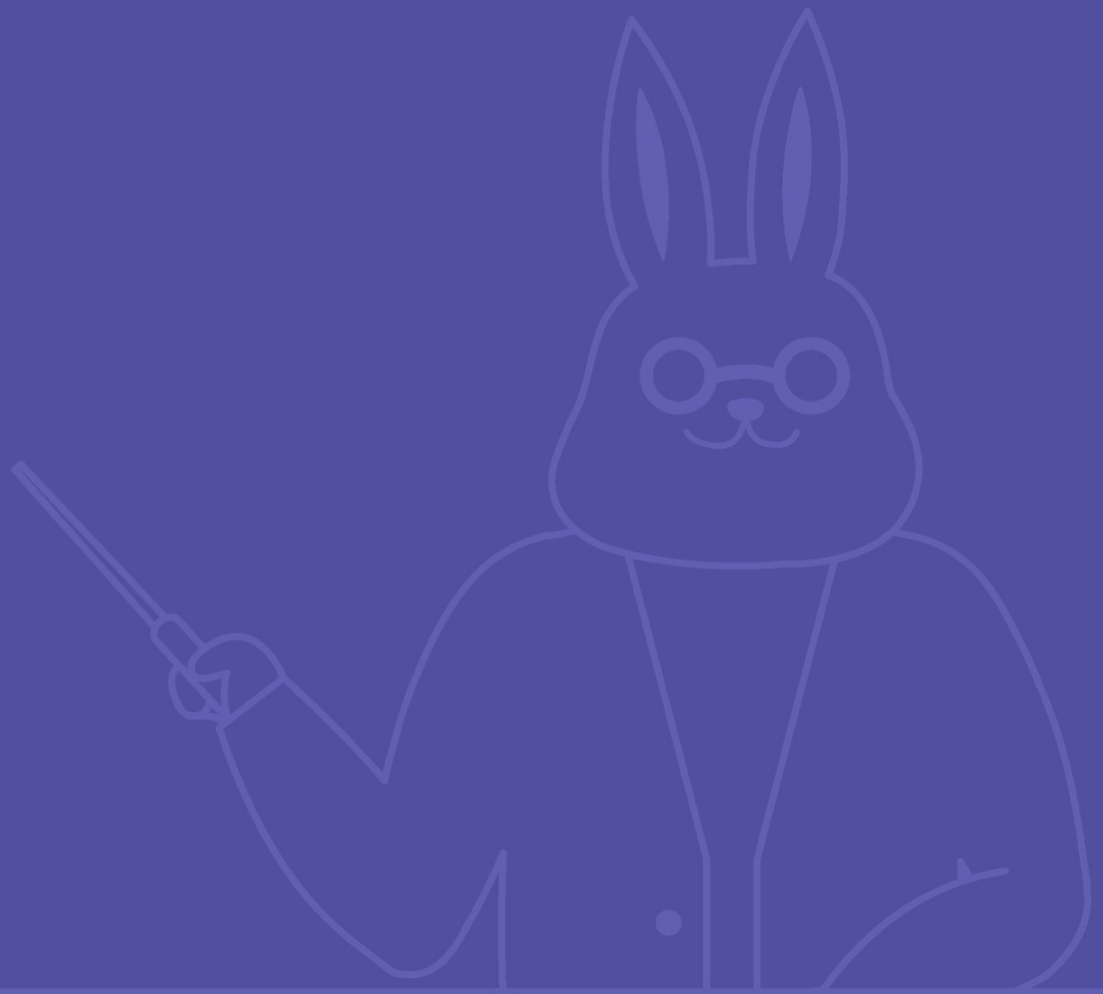
독립적이고, 재사용이 가능한 작은 UI 조각

Javascript 함수와 동일한 용도로 사용

복잡한 웹을 작게 컴포넌트로 쪼개면 -> **재사용**도 쉽고 **효율적으로 관리** 가능!

02

컴포넌트 사용 규칙



컴포넌트 이름은 대문자

코드

```
//App.js
const name = "Elice";
//컴포넌트 선언
function Hello() {
   return <h1> Hello,
      {name} </h1>;
}
export default Hello;
```

코드

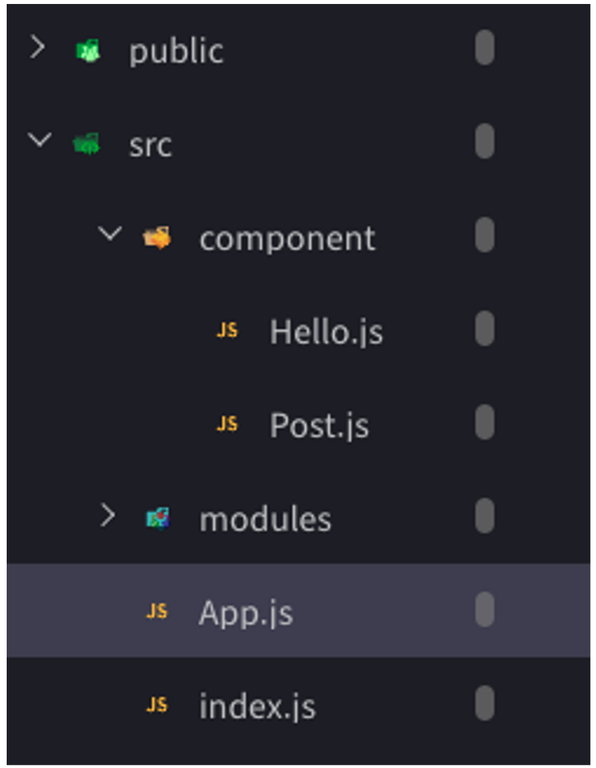
```
//index.js
import Hello from './App';

ReactDOM.render(<Hello/>,
document.getElementById('root'));
```

컴포넌트의 **이름은 항상 대문자**로 시작

Component를 만들고(왼쪽) 다른 Component에서 자유롭게 활용(오른쪽)할 수 있습니다.

return() 내에 있는 건 **태그 하나로 묶기**



코드

```
//App.js
import Hello from './component/Hello.js';
import Post from './component/Post.js';

const name = "Elice";
//컴포넌트 선언
function App() {
    return (
    <div className = "App">
    <Hello/>
    <Post/>
    </div>
    );
}
export default App;
```

03

함수형 컴포넌트 vs 클래스형 컴포넌트

코드

```
const name = "Elice"

const Hello = () => {
    return <h1> Hello, {name} </h1>;
        }

//컴포넌트 호출
const element = <Hello/>
    ReactDOM.render(element,
    document.getElementById('root'));
```

코드

```
import Component from 'react'
const name = "Elice"

class Hello extends Component {
    render() {
    return <h1> Hello, {name} </h1>
    }
}
const element = <Hello/>
    ReactDOM.render(element,
    document.getElementById('root'));
```

function

```
const name = "Elice"

const App = () => {
const name = '함수형 컴포넌트'
   return <div>{name}</div>
}
export default App
```

- 덜 복잡한 UI 로직
- Component 선언이 편함
- 클래스형보다 메모리 자원 덜 사용

class

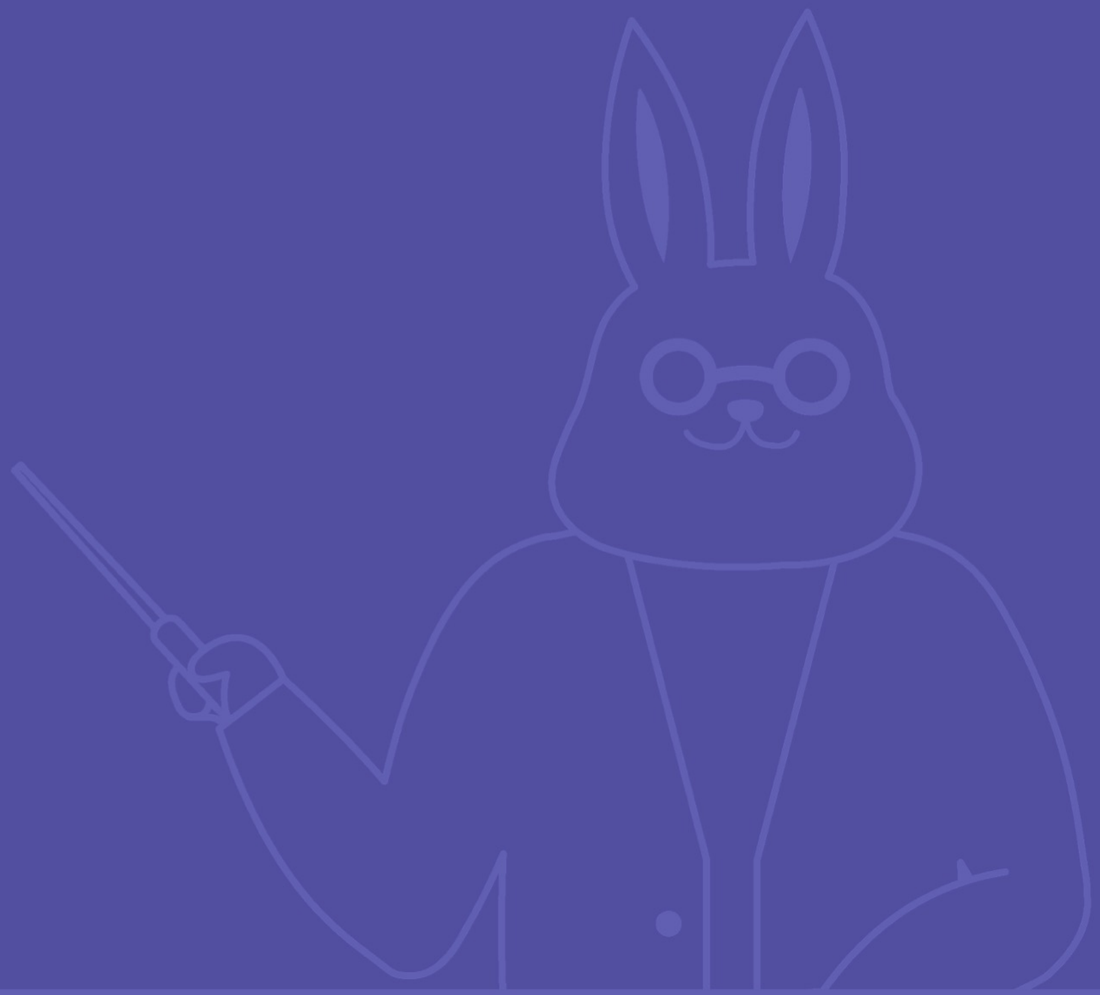
```
import Component from 'react'
const name = "Elice"

class App extends Component {
   render() {
   const name = '클래스형 컴포넌트’
   return <div>{name}</div>
   }
}
export default App
```

- Class 키워드 필요
- Component를 상속받아야 함
- Render() 메소드 반드시 필요
- 함수형보다 메모리 자원 더 사용

04

Template Literal



Template Literal이란?

역따옴표 (백틱 : ` `) 로 감싸는 문자열

문자열을 템플릿화해서, **배열이나 객체 데이터들을 갈아끼울 수 있는 구조**

- 데이터 기반의 HTML 페이지를 만드는데 활용
- 반복되는 HTML 의 태그나 목록을 동적으로 생성 가능

가볍고 간결한 데이터 기반 페이지 기능!

백틱키: option key + esc키 밑에 있는 ₩ key

문자열을 변수와 함께 활용할 때

```
var name = "엘리스";
var job = "개발자";
var age = "29";
console.log( "안녕하세요 제 이름은 \" + name + "\" 입니다. \n" +
"직업은 \"" + job + "\" 이고 \n" +
" 나이는 \"" + age + "\"입니다. \n")
```

Template Literal을 활용할 때

```
console.log(
`안녕하세요 제 이름은 "${name}"입니다.
직업은 "${job}" 이구요
나이는 "${age}" 입니다.` )
```

`${변수명}` **의 형태로 특정 변수를 전달**

템플릿 리터럴의 선언

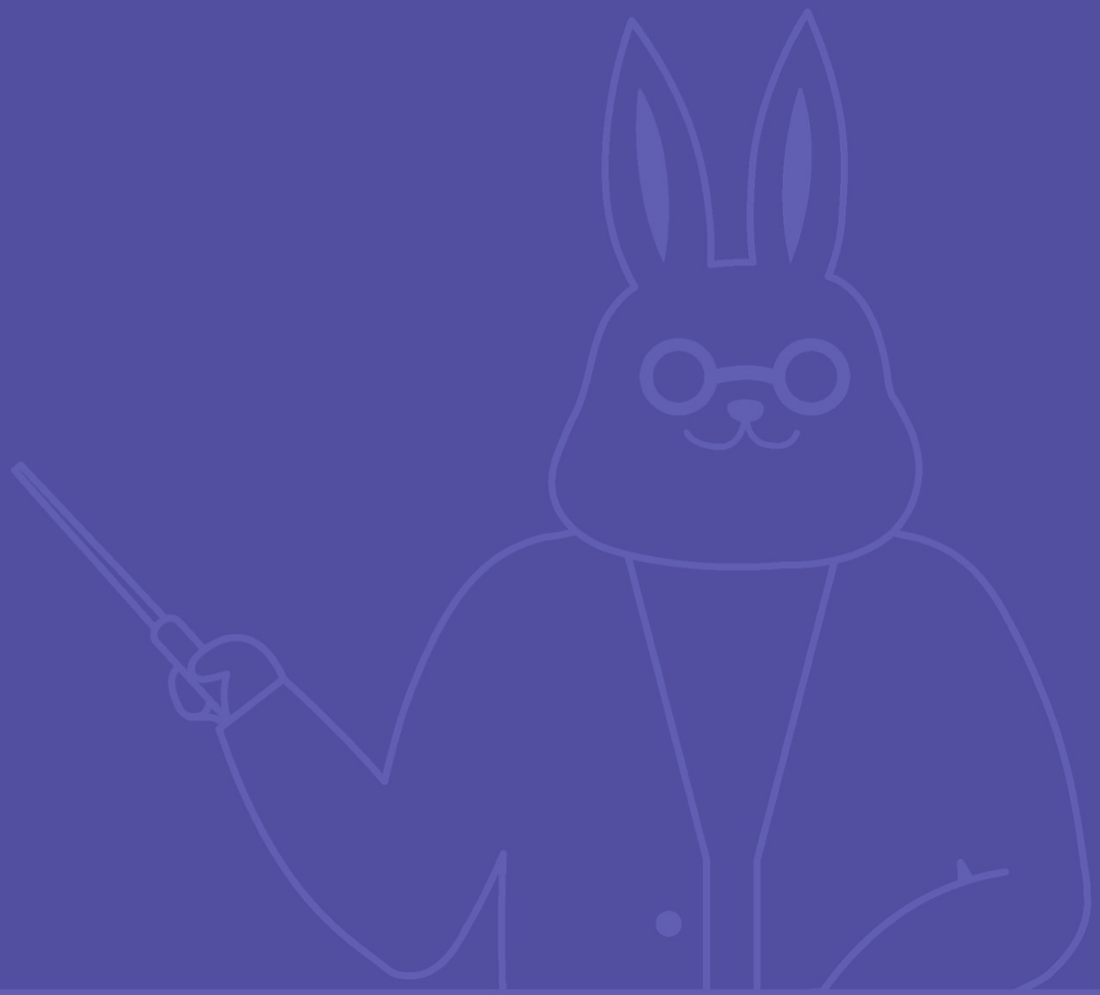
코드

```
let word = "Elice";
console.log(`문자열 ${word}`);
```

문자열에 변수 역할을 하는 템플릿 코드를 추가
-> 자바스크립트 데이터를 대입해 새로운 문자열을 생성

05

컴포넌트 반환하기



/* elice */

컴포넌트 반환

코드

```
//App.js
import Nav from './component/nav/Nav.js'
import Posts from './component/post/Posts.js'

const App = () => {
return `${Nav()}
        <div class="container">
        ${Posts()}
        </div>`
}
```

Hook과 state 관리는 리엑트 강좌에서!

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
안조현

강사
안조현

감수자
이재성

디자이너
강혜정

연락처

TEL

070-4633-2015

WEB

https://elice.io

E-MAIL

contact@elice.io