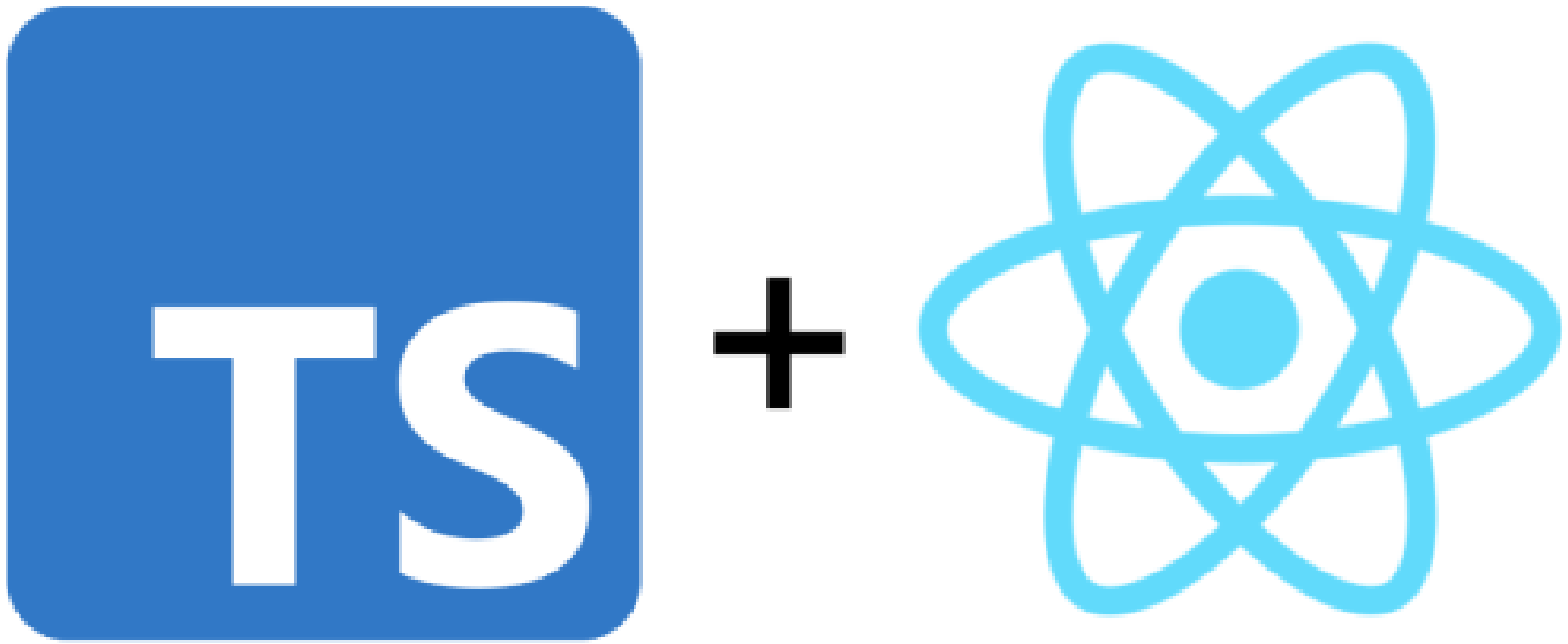
# What is React.FC



# 다양한 컴포넌트 선언 방법

```tsx
import { FC } from 'react';

type TestProps = {
  name: string;
}

const TestComponent
  return (
    <div>
      <div>Hello!
      {children}
    </div>
```

```tsx
const App = () =>
  const { theme }

  return (
    <div classNam
      <Routes />
    </div>
  );
}
```

## React.FC vs JSX.Element

Discussion

So another dev and I have both been assigned to a team that has recently lost all of it's devs and we're to clean up the leftover react code.

We've both agreed to get rid of unnecessary class components and move more towards a functional approach, but i've always used

React.fc and he's always used JSX.element. Example below:

```
const HelloWorld: React.FC<Iprops> = ({name}) => <div>`hi ${name}`</div>
```

vs

```
function HelloWorld({name}: Iprops): JSX.Element {
     return (<div>`hi ${name}`</div>)
}
```

I get it that React.FC is returning a ReactElement, but isn't JSX.Element also just a ReactElement without types? There would be no functional or performance difference between the two approaches correct? are there any pitfalls of using one over the other?

```tsx
t => {


orySingleton}>
{<ProgressLoading />}>
```

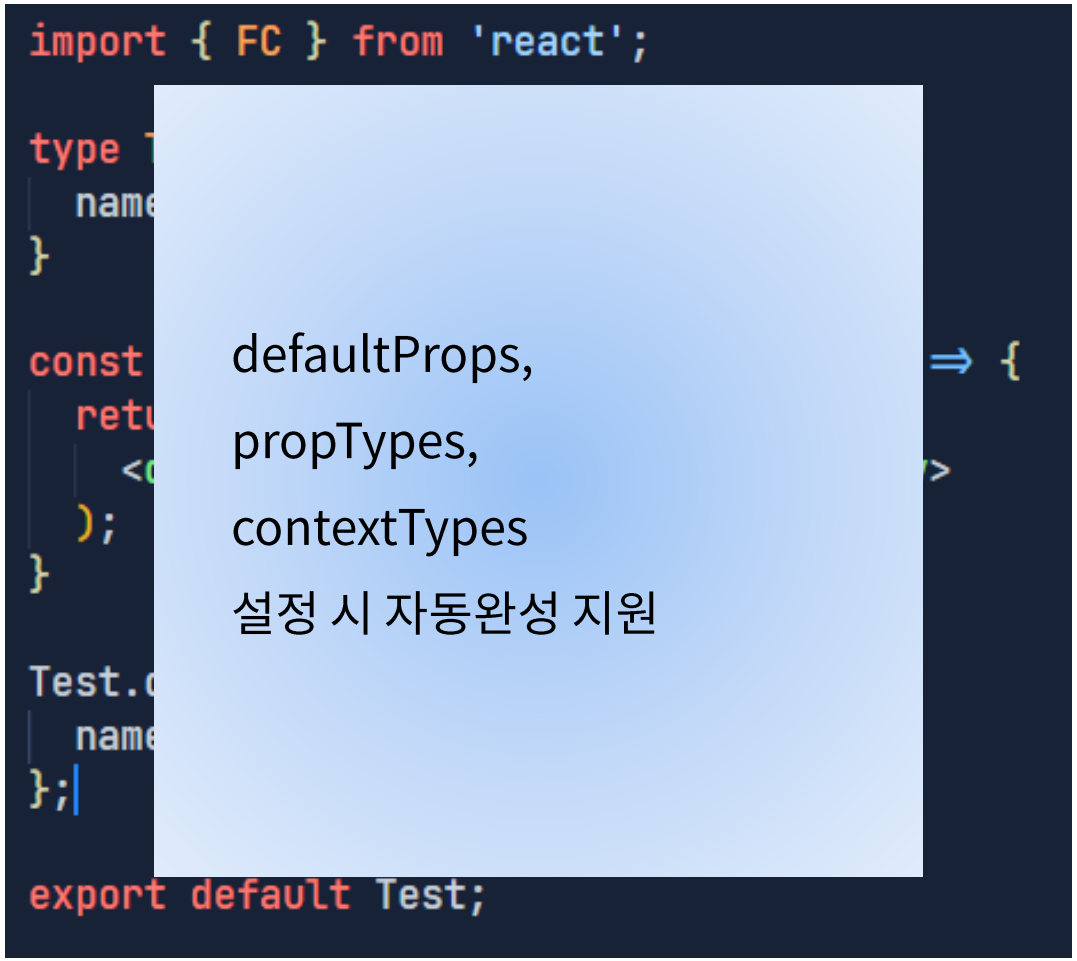
# React.FC 너의 특색이 뭐야?

```tsx
type TestProps = {
  name: string;
}

const Test = ({ name }: TestProps) => {
  return (
    <div>Hello! My name is ${name}</div>
  );
}

Test.defaultProps = {
  name: 'yiyb0603',
};

export default Test;
```




React.FC를 붙이지 않으면, 그냥 하나의 JSX Element를 return하는 함수로 타입 추론.

React.FC를 붙이면, 하나의 함수형 컴포넌트 (Functional Component)로 타입이 추론된다.

# React.FC 기본 사용법!

```tsx
import { FC } from 'react';

type TestProps = {
  name: string;
}

const TestComponent: FC<TestProps> = ({ name, children }) => {
  return (
    <div>
      <div>Hello! My name is {name}</div>
      {children}
    </div>
  );
}

export default TestComponent;
```

Props Type을 제네릭으로 넣어줍니다.

기존에 Props의 타입을 지정해주는 곳 대신, FC<제네릭>에 넣어줌으로써 안 넣어줘도 된다.

기존에는 그저 JSX를 반환하는 함수 형태였지만, FC를 붙임으로써 하나의 함수형 컴포넌트 타입이 추론된다.

하지만 문제점은 존재한다.

# 1. defaultProps 이슈

```tsx
type TestProps = {
  name: string;
}

const Test = ({ name }: T
  return (
    <div>Hello! My name i
  );
}

Test.defaultProps = {
  name: 'yiyb0603',
};

export default Test;
```


```tsx
import Test from './Test';

Render = () => {

/>

ault TestRender;
```

# 1. defaultProps 이슈

```tsx
import { FC } from 'react';

type TestProps = {
  name: string;
}

const Test: FC<TestP
  return (
    <div>Hello! My n
  );
}

Test.defaultProps =
  name: 'yiyb0603',
};

export default Test;
```

```tsx
import Test from './Test';

r = () => {

TestRender;
```

React.FC를 이용해서 선언한 컴포넌트의 defaultProps가 있음에도 불구하고,
Props를 무조건 넘겨주라고 요구함.

# 2. children은 기본이라구~

```
type PropsWithChildren<P> = P & { children?: ReactNode };
```

React.FC를 이용해서 선언한 컴포넌트의 props중, 기본적으로 children이 Optional로 명시되어 있다.
(이는 장점이 될 수도 있고, 단

장점: Component Props 타 적용되어 있기 때문)

단점: children이 필요하지 않 속성이 들어간다. (이는 TS 철학과 모

위의 단점처럼 children prop 을 지정해주는 것이 좋다.

```
type TestProps = {
  name: string;
}

const TestComponent = ({ name }: TestProps) => {
  return (
    <div>
      <div>Hello! My name is {name}</div>
    </div>
  );
}

export default TestComponent;
```

# 번외. 함수의 선언 일관성

React.FC는 함수 그 자체에 타입을 지정하며, Functional Component 그 자체가 된다.
매개변수에 꼭 타입을 지정 해 줘야하는 일반적인 함수의 경우에는 함수 그 자체에 타입을 지정하기 번거롭다.
만약, Props에만 타입을 지정해주면 다른 형태의 함수들과 선언 형식이 통일 된다는 의견.

> 이에 대한 자세한 사항은 아래의 글을 참고하면 더 도움이 돼요!
> https://vo.la/mENqv