

# TypeScript

with React

# React with Typescript

- 프로젝트 만들기

```
npx create-react-app 프로젝트 이름 --template typescript
```

- (참고) 기존 프로젝트에 typescript 를 적용하고 싶다면

```
npm install typescript @types/node @types/react @types/react-dom @types/jest
```

- 모듈 설치! &
- js & jsx 파일을 ts, tsx 파일 변경
- tsconfig.json 파일 생성 (compilerOptions 안의 “jsx”: “react-jsx” 추가)


# props와 interface

- props를 이용해서 데이터를 상위 컴포넌트에게 받을 때는 props 가 어떤 형태로 넘어오는지 type을 미리 적어둬야 함.

```
function PropType1({ name }) {  
  return (  
    <>  
      <h2>Hello {name}</h2>  
    </>  
  );  
}
```

jsx를 사용한 react 에서 props를 받아오는 법

```
interface Props {  
  name: string;  
}  
  
function PropType1({ name }: Props) {  
  return (  
    <>  
      <h2>Hello {name}</h2>  
    </>  
  );  
}
```



typescript를 사용하는 react 에서 props를 받아오는 법

# props와 interface

- 상위 컴포넌트에서 prop를 넘겨줄 때, 정의된 props라면 반드시 넘겨줘야 함.

```
Default function Lecture() {  
  Property 'name' is missing in type '{}' but required in type  
  'Props'. ts(2741)  
  PropsType.tsx(5, 3): 'name' is declared here.  
  <(alias) function PropsType1({ name }: Props): JSX.Element  
  import PropsType1  
  View Problem (Alt+F8) Quick Fix... (Ctrl+.)  
  <PropsType1/>
```

하위 컴포넌트에서  
interface로 정의된 props들은  
사용되지 않더라도 넘겨주어야 합니다.

- 넘겨주고 싶지 않은 props가 있다면 → ? 를 이용!

```
interface Props {  
  name?: string;  
}
```



```
<PropsType1/>
```

name이라는 key에 물음표(?)를 붙여  
있어도 되고, 없어도 되는 optional한 요소임을 알려주었을 때만  
props를 전달하지 않아도 됩니다.

# useState generic <T>

- generic 함께 쓰기

```
const [count, setCount] = useState<number>(0);  
const [text, setText] = useState<string>>('');
```

- 초기값에 대한 type을 generic을 이용해서 설정
- 물론 setState 이용해서 state를 변경할 때에도 generic으로 정해진 type으로만 변경 가능!

- 하지만 typescript 가 타입 유추를 알아서 잘 합니다.

- 기본적으로는 useState를 사용할 때 generic을 쓰지 않아도 괜찮아요.

- state의 값이 null 일 수도 있고 아닐 수도 있을 때, 반드시 generic으로 union type 전달!

```
interface Data { name:string; age:number;}  
const [data, setData] = useState<null | Data>(null)
```

# useRef generic <T>

## 1. 값 저장

- 초기값에 대한 type을 generic으로 작성

```
const refVal = useRef<number>(0)
```

## 2. DOM 접근

- DOM 객체에 접근하기 위한 useRef의 generic에는, type에 HTMLElement 타입 이용

```
const ref = useRef<HTMLInputElement>(null);
```

# event 객체의 type

- click event의 e.target,
- keydown event의 e.key, e.code
- change event 의 e.target.value, ...

- html 요소에 이벤트를 적용 할 때, 이벤트가 발생한 요소에 대한 정보가 담겨져 있는 객체

```
<div onClick={(e) => console.log(e)}>onClick</div>
```



왼쪽의 div를 클릭하게 되면,  
div click 에 대한 정보가 콘솔창에  
출력됩니다.

- 매개변수인 e의 타입은 어떻게 지정해야 할까요?

# event 객체의 type

- onClick, onDrag, .. (click 과 관련된 event 객체)
    - `React.MouseEvent<HTMLElement>`
  - onChange
    - `React.ChangeEvent<HTMLInputElement>`
  - onKeyDown, onKeyUp, .. (keyboard 이벤트와 관련된 event 객체)
    - `React.KeyboardEvent<HTMLInputElement>`
-