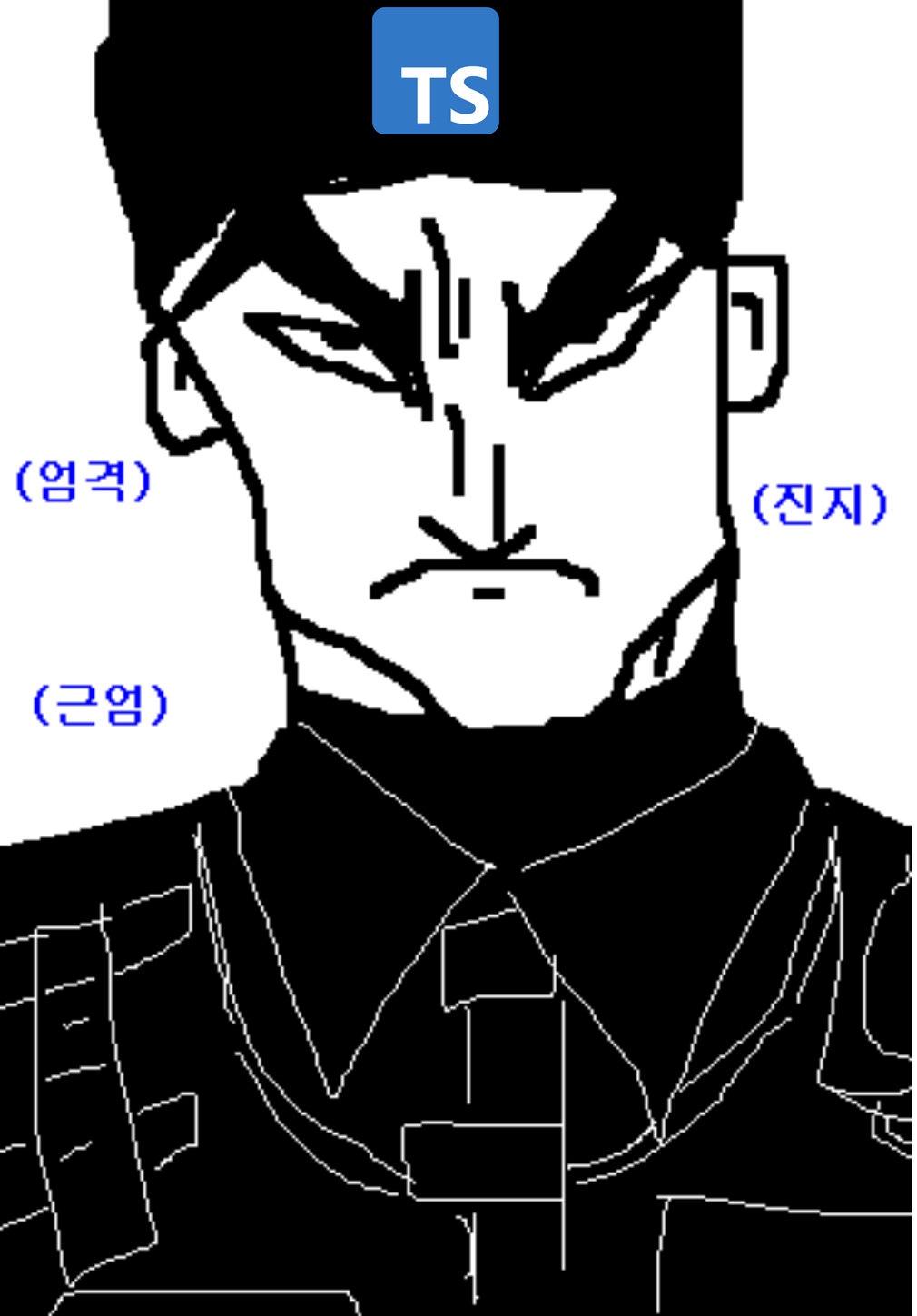


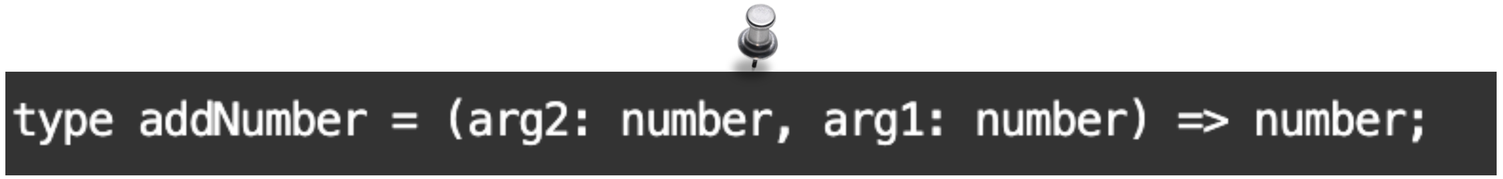
Item 29.

사용할 때는 너그럽게, 생성할 때는 엄격하게

'23. 06. 10. (SAT) 신현호 (@SWARVY)

포스텔의 법칙

TCP 구현체는 견고성의 일반적 원칙을 따라야 한다.
당신의 작업은 엄격하게 하고, 다른 사람의 작업은 너그럽게 받아들여야 한다.



```
type addNumber = (arg2: number, arg1: number) => number;
```

함수의 시그니쳐에도 비슷한 규칙을 적용해야해요.

함수의 매개변수는 타입의 범위가 넓어도 되지만,
결과를 반환할 때는 일반적으로 타입의 범위가 더 **구체적이어야** 해요

new naver.maps.AroundControl(AroundControlOptions)

Parameters

Name	Type	Description
`AroundControlOptions`	AroundControlOptions	UI 컨트롤 옵션

가능한 타입이 여러가지였다면?

함수의 **매개변수**는 타입의 범위가 넓으면 사용하기 **편리해요**
하지만, **반환 타입**은 범위가 넓으면 사용하기 **불편해요**

즉, 사용하기 쉬운 API일수록 반환 타입이 **엄격**하다는 거에요.

```ts
interface CameraOptions {
  center?: LngLat;
  zoom?: number;
  bearing?: number;
  pitch?: number;
}
type LngLat =
  { lng: number; lat: number; } |
  { lon: number; lat: number; } |
  [number, number];
```

수많은 **선택적 속성을 가지는 반환 타입**과 **유니온 타입**은
CameraOptions를 사용하기 어렵게 만듭니다.

이럴 때는 기본 형식을 구분하여
유니온 타입을 요소별로 **분기**시킬 수 있습니다.

이런식으로요,

```
interface Camera {
  center: LngLat;
  zoom: number;
  bearing: number;
  pitch: number;
}
interface CameraOptions extends Omit<Partial<Camera>, 'center'> {
  center?: LngLatLike;
}
```

Camera가 너무 엄격하니 조건을 완화하여 느슨한 **CameraOptions** 타입을 만들었습니다

혹은,

```
interface CameraOptions {
  center?: LngLatLike;
  zoom?: number;
  bearing?: number;
  pitch?: number;
}
```

명시적으로 타입을 추출해서 다음처럼 작성할 수도 있습니다

요약

보통 매개변수 타입은 반환 타입에 비해 범위가 넓은 경향이 있어요
선택적 속성과 **유니온 타입**은 반환 타입보다 **매개변수 타입에 더 일반적이에요**

매개변수와 반환 타입의 **재사용을 위해서**
기본 형태(반환 타입)와 **느슨한 형태(매개변수 타입)**를 도입하는 것이 좋아요