TypeScript

Why TypeScript?

정적 타입 검사

실행 전 버그 발견

자동 완성

생산성 극대화

안정성

유지보수성 증대

TS 셋팅

설치 프로그램

- Node.js (18 ver 이상) : node –version
- pnpm (또는 npm)

이미 Node.js와 pnpm이 있다면, 아래 명령어를 복사해서 터미널에 붙여넣으세요.

```
# 1. 초기화 및 설치

pnpm init
pnpm add -D typescript ts-node @types/node

# 2. 설정 파일 및 폴더 생성

pnpm tsc --init
mkdir –p  src/exercises

# 3. 테스트 파일 생성 및 실행

echo 'console.log("Hello, TS!");' > index.ts
pnpm exec ts-node index.ts
```

환경 설정 (tsconfig.json)

TypeScript 프로젝트의 루트에 위치하며 컴파일러 옵션을 설정하는 파일입니다.

•**생성**: tsc --init 명령어로 기본 설정 파일 생성

•**주요 옵션**:

- target: 컴파일된 JavaScript의 버전 설정 (예: es6, es2020)
- module: 모듈 시스템 설정 (예: commonjs, esnext)
- strict: 모든 엄격한 타입 검사 옵션 활성화 (true 권장)
- outDir: 컴파일된 JavaScript 파일이 저장될 경로 (예: ./dist)
- rootDir: 소스 파일이 위치한 경로 (예: ./src)

기본 타입 (Basic Types)

string

문자열 데이터

```
let name: string = "Alice";
```

number

정수 및 부동소수점

```
let age: number = 30;
```

boolean

참/거짓 논리값

```
let isDone: boolean = true;
```

null / undefined

값이 없거나 미할당

```
let n: null = null;
```

any

모든 타입 허용 (주의)

```
let data: any = 123;
```

void

반환 값이 없음

```
function log(): void {}
```

배열과 튜플 (Array & Tuple)

Array (배열)

동일한 타입의 요소만으로 구성된 리스트입니다.
길이가 가변적입니다.

```
let list: number[] = [1, 2, 3];

// 제네릭 방식
let list2: Array<string> =
 ["A", "B"];
```

Tuple (튜플)

정해진 개수와 순서에 따라 타입이 지정된 배열입니다.

```
let user: [string, number];

 user = ["John", 30]; // OK

// Error: 순서/타입 불일치
// user = [30, "John"];
```

인터페이스 (Interface)

객체 구조 정의

객체가 가져야 할 속성과 타입을
명확하게 정의하는 계약(Contract)입니다.

확장성 (Extends)

기존 인터페이스를 상속받아
새로운 기능을 쉽게 추가할 수 있습니다.

선택적 속성 (?)

필수가 아닌 속성은 물음표(?)를
사용하여 유연하게 처리합니다.

```
Interface User {
name: string;
age: number;
email?: string; // Optional
 }

// 인터페이스 확장 (상속)
interface Admin extends User {
role: string;
 }

const admin: Admin = {
name: "Alice",
age: 30,
role: "Manager"
 };
```

타입 별칭 (Type Alias)

유연한 타입 정의

객체뿐만 아니라 원시 값, 유니온, 튜플 등
모든 타입에 새로운 이름을 부여할 수 있습니다.

복잡한 타입의 단순화

길고 복잡한 유니온 타입이나 함수 시그니처를
간결한 이름으로 추상화하여 가독성을 높입니다.

Interface와의 차이

extends 키워드를 사용할 수 없으며,
선언 병합(Declaration Merging)이 불가능합니다.

```
// 1. 원시 타입 & 유니온
type ID = string | number;
type Status = "Ready" | "Waiting";
// 2. 객체 타입
type User = {
 id: ID;
 name: string;
 status: Status;
 };
// 3. 함수 타입
type Callback = (msg: string) => void;
const user: User = {
 id: "u-1",
 name: "Alice",
 status: "Ready"
 };
```

함수 (Functions)

기본 선언

매개변수와 반환 값에
타입을 명시합니다.

```
function add (x: number, y: number): number {
return x + y;
 }
```

화살표 함수

간결한 문법으로
타입을 정의합니다.

```
const sub = (x: number, y: number): number=> x - y;
```

선택적 매개변수

? 기호를 사용하여
생략 가능한 인자를 만듭니다.

```
function greet (name: string, age?: number): void {
// age는 number | undefined
 }
```

유니온과 열거형 (Union & Enum)

| Union Types

"OR" 연산자처럼 동작하여
여러 타입 중 하나를 허용합니다.

```
let id: string|number;

id = "user-123"; // OK
id = 123; // OK
// id = true; // Error
```

{} Enums

관련된 상수들의 집합을
이름으로 정의하여 관리합니다.

```
enum Role {
 ADMIN,
 USER,
 GUEST
 }
let role: Role = Role.ADMIN;
```

정리 (Summary)

01 JavaScript + Type

TypeScript는 JavaScript에 정적 타입을 더한 것입니다.
기존 JS 지식을 그대로 활용할 수 있습니다.

02 Compile-time Safety

코드를 실행하기 전에 오류를 발견하여
버그를 획기적으로 줄여줍니다.

03 Start Simple

모든 기능을 알 필요는 없습니다.
기본 타입, 인터페이스, 함수부터