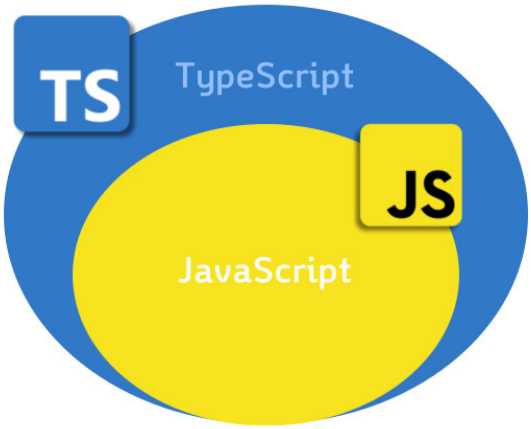
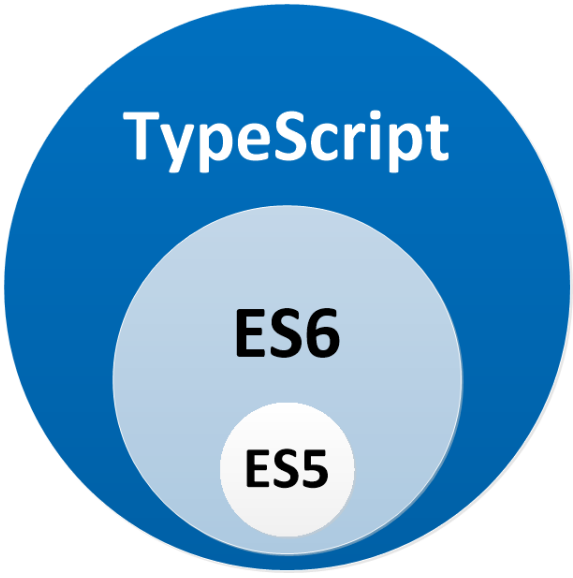
TypeScript

1. 타입스크립트란?

마이크로소프트에서 구현한 자바스크립트의 슈퍼셋 프로그래밍언어로 확장자는 .ts를 사용한다. 간단하게 말하면 자바스크립트를 기반으로 정적 타입 문법을 더한 언어이다. 덕분에 개발 도구에게 개발자가 의도한 변수나 함수 등의 목적을 더욱 명확하게 전달할 수 있고, 그렇게 전달된 정보를 기반으로 코드 자동 완성이나 잘못된 변수/함수 사용에 대한 에러 알림 같은 풍부한 피드백을 받을 수 있게 되므로 순수 자바스크립트에 비해 어마어마한 생산성 향상을 꾀할 수 있다. 즉, '자바스크립트를 실제로 사용하기 전에 있을만한 타입 에러들을 미리 잡는 것' 이 타입스크립트의 사용 목적이다. 그래서 자바스크립트 유저라면 간단하게 배울 수 있고 자바스크립트의 타입으로부터 오는 문제점들은 해결해주어서 갈수록 시장에서 한 자리씩 차지하고 있으며 React.js나 Vue.js에서도 타입스크립트를 사용하는 것을 심심치 않게 볼 수 있다.





2. 타입스크립트의 특징

1. 타입스크립트는 컴파일, 정적 타입 언어이다.

   자바스크립트는 동적 타입의 인터프리터 언어로 런타임에서 오류를 발견할 수 있다. 이에 반해 타입스크립트는 정적 타입의 컴파일 언어이며 타입스크립트 컴파일러 또는 바

벨(Babel)을 통해 자바스크립트 코드로 변환된다. 코드 작성 단계에서 타입을 체크해 오류를 확인할 수 있고 미리 타입을 결정하기 때문에 실행 속도가 매우 빠르다는 장점이 있다. 하지만 코드 작성 시 매번 타입을 결정해야 하기 때문에 번거롭고 코드량이 증가하며 컴파일 시간이 오래 걸린다는 단점이 있다.

2. 자바스크립트 슈퍼셋(Superset)

   타입스크립트는 자바스크립트의 슈퍼셋, 즉 자바스크립트 기본 문법에 타입스크립트의 문법을 추가한 언어이다. 따라서 유효한 자바스크립트로 작성한 코드는 확장자를 .js에서 .ts로 변경하고 타입스크립트로 컴파일해 변환할 수 있다.

3. 객체 지향 프로그래밍 지원

   타입스크립트는 ES6(ECMAScript 6)에서 새롭게 사용된 문법을 포함하고 있으며 클래스, 인터페이스, 상속, 모듈 등과 같은 객체 지향 프로그래밍 패턴을 제공한다.

4. 높은 수준의 코드 탐색과 디버깅

   타입스크립트는 코드에 목적을 명시하고 목적에 맞지 않는 타입의 변수나 함수들에서 에러를 발생시켜 버그를 사전에 제거한다. 또한 코드 자동완성이나 실행 전 피드백을 제공하여 작업과 동시에 디버깅이 가능해 생산성을 높일 수 있다.

5. 자바스크립트 호환

   타입스크립트는 자바스크립트와 100% 호환된다. 따라서 프론트엔드 또는 백엔드 어디든 자바스크립트를 사용할 수 있는 곳이라면 타입스크립트도 쓸 수 있다. 타입스크립트는 앱과 웹을 구현하는 자바스크립트와 동일한 용도로 사용 가능하며 서버 단에서 개발이 이루어지는 복잡한 대형 프로젝트에서도 빛을 발한다.

6. 강력한 생태계

   타입스크립트는 그리 오래되지 않은 언어임에도 불구하고 강력한 생태계를 가지고 있다. 대부분의 라이브러리들이 타입스크립트를 지원하며 마이크로소프트의 비주얼 스튜디오 코드(VSCode)를 비롯해 각종 에디터가 타입스크립트 관련 기능과 플러그인을 지원한다.

3. 자바스크립트의 문제점

```
if ("" == 0) {
  // 참입니다!
}
if (1 < x < 3) {
  // 어떤 x 값이던 참
}
```

```
const obj = { width: 10, height: 15 };
const area = obj.width * obj.heigth;
```

자바스크립트의 동일 연산자는 인수를 강제로 변환하여 예기치 않은 동작을 유발하며 존재하지 않은 프로퍼티의 접근을 허용한다.

대부분의 프로그래밍 언어는 이런 종류의 오류들이 발생하면 오류를 표출해주고, 일부는 코드가 실행되기 전인 컴파일 중에 오류를 표출해준다. 작은 프로그램을 작성할 때에는 관리가 가능하지만 큰 규모의 프로그램을 작성할 때에는 심각한 문제를 야기할 수 있다.

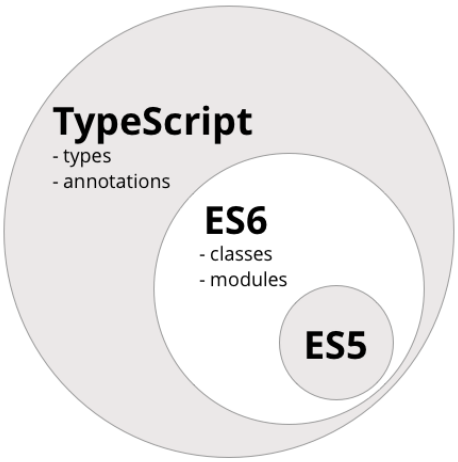
4. 자바스크립트와 타입스크립트의 차이점

	자바스크립트	타입스크립트
개요	웹 페이지에 상호 작용을 추가하는 인터프리터 기반 언어	일반 자바스크립트로 컴파일되는 자바스크립트의 상위 집합
언어 범주	스크립팅 언어	객체 지향 프로그래밍 언어
편집	컴파일러가 필요 없음, 웹 브라우저에서 실행	자바스크립트 파일로 변환하기 위해 타입스크립트 컴파일러가 필요
객체 지향 기능	순수하게 객체 지향적이지 않으며, 프로토 타입 기반이고 인터페이스가 없다.	객체 지향 프로그래밍 언어이며 클래스 기반이다. 클래스, 상속, 인터페이스 및 수정자를 사용할 수 있다.
실행 방법	클라이언트 측에서 실행	클라이언트 측과 서버 측에서 실행
정적 검사	정적 유형 검사가 없다	정적 유형 검사가 있다
모듈성	지원 모듈을 허용하지 않음	파일과 모듈을 가져올 수 있다

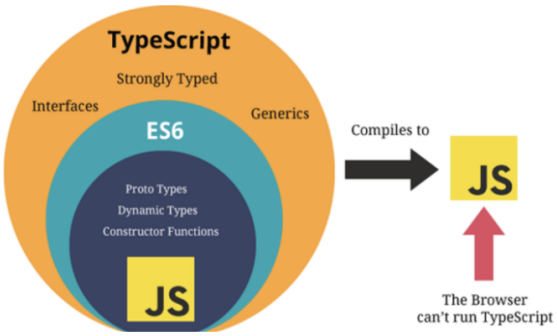
자바스크립트는 동적 웹 페이지를 만드는 언어이다. HTML 및 CSS와 쉽게 통합 할 수있는 경량 해석 언어이다. 양식 유효성 검사, 애니메이션 및 웹 페이지에 멀티미디어 기능 추가에 유용하다. 타입스크립트는 기능이 추가 된 자바스크립트이다. 자바스크립트와 타입스크립트의 차이점은 자바스크립트는 클라이언트 측 스크립팅 언어이고 타입스크립트는 객체 지향 컴파일 언어라는 것이다.

인터프리터 언어인 자바스크립트와는 다르게 타입스크립트는 컴파일 언어인데, 전통적인 C 계열의 컴파일 언어와는 차이가 있어 브라우저에서 이해할 수 있도록 자바스크립트 코드로 변환하는 트랜스파일 언어라고도 부른다.

동작 방식은 TS -> (컴파일(트랜스파일)) -> JS -> (실행) 과 같이 이루어지는데, TS가 JS로 변환될 때 helper 코드가 위아래로 추가되기 때문에, 절대적인 코드의 양 자체는 타입스크립트가 더 많다. 물론 개발자는 타입스크립트만 작성하고, 변환은 알아서 해주므로 개발할 때 크게 문제되는 부분은 아니다.



TypeScript Superset



ts config파일을 통해 js파일 변환시에 설정(경로,버전 등)을 할 수 있다.

---

1. 개발 환경 구성

개발환경 : vsCode, Chrome, Node

타입스크립트 컴파일러 설치

```
> npm i -g typescript
> tsc --version
```

```
PS C:\Users\YM\OneDrive\바탕 화면\ts> npm i -g typescript

added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
PS C:\Users\YM\OneDrive\바탕 화면\ts> tsc --version
Version 4.6.4
```

타입스크립트를 전역으로 설치하고, 설치가 되었는지 확인하는 명령어

typescript 패키지는 서버와 클라이언트로 동작하는 두 개의 프로그램을 포함한다. 따라서 컴파일을 하는 명령어는 tsc이고, 이는 타입스크립트 컴파일러 + 클라이언트의 의미가 있다.

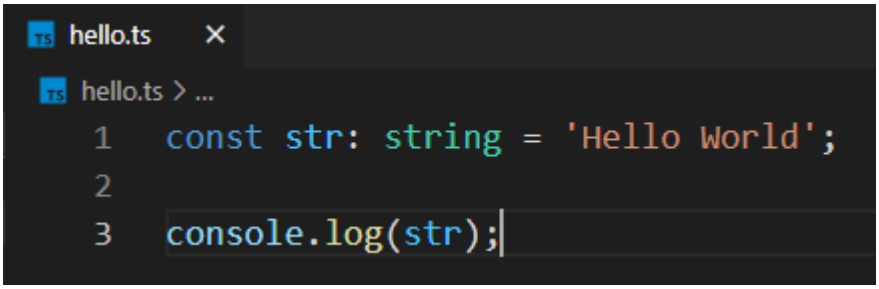
컴파일과 실행을 동시에

```
> npm i -g ts-node
> ts-node hello.ts # example
```

```
PS C:\Users\YM\OneDrive\바탕 화면\ts> npm i -g ts-node

changed 17 packages, and audited 18 packages in 1s

found 0 vulnerabilities
PS C:\Users\YM\OneDrive\바탕 화면\ts> ts-node hello.ts
Hello World
```



tsc는 타입스크립트 코드를 ES5 형식의 자바스크립트 코드로 변환만 할 뿐이다.
변환과 실행을 동시에 하려면 ts-node라는 프로그램을 설치해야 한다.

2. 프로젝트 생성 및 관리

```
> mkdir ts-example
> cd ts-example
> npm init --y
```

위와 같이 프로젝트를 위한 폴더를 생성하고 예시로 Node.js 프로젝트 시작을 위해 npm init 을 실행한다.

```
> npm i -D @types/node
```

타입스크립트는 자바스크립트로 개발된 라이브러리를 사용할 때 문법에 맞게 사용했는지 검증하기가 어렵다. 따라서 라이브러리들은 추가로 타입 라이브러리들을 제공해야 한다.

타입스크립트는 또한 웹 브라우저나 Node.js가 기본으로제공하는 타입들의 존재도 그냥은 알지 못한다. 예를 들어, Promise와 같은 타입을 사용하려면 위와 같이 @types/node라는 패키지를 설치해야 한다.

```
> tsc --init
```

위 명령어를 통해 타입스크립트 컴파일러의 설정 파일(ts 파일을 js 파일로 변환 할때 옵션을 주기 위하여) 인 tsconfig.json 파일이 생성된다.

```
{
  "name": "ts-example",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "scripts": {
    "dev" : "ts-node src",
```

```
    "build" : "tsc && node dist"
  },
  ...생략
}
```

위에서 npm init으로 생성한 pakcage.json 또한 기본 구성만 되어 있을 것이다. 개발을 위해서 타입스크립트 소스코드를 ES5로 변환해 node로 실행할 수 있도록 아래와 같이 script와 main을 수정하도록 한다.

3. 리액트에 타입스크립트 구성

위 예시는 기본적인 타입스크립트 환경 구성 예시이고, 내가 프로젝트에서 담당한 부분은 프론트앤드, 리액트 부분을 중점적으로 맡았기 때문에 리액트에 타입스크립트를 구성하는 방법도 알아보았다.

```
> npx create-react-app my-app --template typescript
```

리액트를 타입스크립트와 함께 설치하는 경우 사용하는 명령어

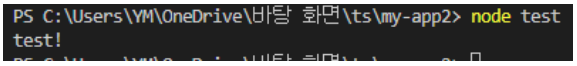
```
> npx typescript --init
```

tsconfig.json 파일 생성

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "baseUrl": "./src",
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
```

```
  "include": [
    "src"
  ]
}
```

tsconfig.json 파일 수정



test.ts 파일 생성 후 실행

4. 타입스크립트의 문법

- 자바스크립트

```
function sum(a, b) {
  return a + b;
}
```

- 타입스크립트

```
function sum(a: number, b: number) {
  return a + b;
}
```

타입스크립트는 자바스크립트 함수에 타입을 지정해서 예기치 않은 문제점을 방지한다 !

기본타입

타입스크립트는 다양한 기본 타입을 제공한다.
-> Boolean, Number, String, Object, Array, Tuple, Enum, Any, Void, Null, Undefined, Never

- 변수에 타입 설정

```
let str: string = 'hi';
let num: number = 100;

let arr: Array = [1, 2, 3];
let arr2: number[] = [1, 2, 3];

let obj: object = {};
```

```
let obj2: { name: string, age: number} = {
  name: 'hoho',
  age: 22
};
```

- 함수에 타입 설정

```
function add(a: number, b: number): number {
  return a+b;
}

function log(a: string, b?: string, c?: string) {
    console.log(a);
}
```

기본 타입 중 자바스크립트에 존재하지 않는 타입은 다음과 같다.

- Tuple: 배열의 타입 순서와 배열 길이를 지정할 수 있는 타입이다.

```
var arr: [string, number] = ['aa', 100];
```

- Enum: Number 또는 String 값 집합에 고정된 이름을 부여할 수 있는 타입이다. 값의 종류가 일정한 범위로 정해져 있는 경우에 유용하다. 기본적으로 0부터 시작하며 값은 1씩 증가한다.

```
enum Shoes {
  Nike = '나이키',
  Adidas= '아디다스'
}
```

- Any: 모든 데이터 타입을 허용한다.
- Void: 변수에는 undefined와 null만 할당하고 함수에는 리턴 값을 설정할 수 없는 타입이다.
- Never: 특정 값이 절대 발생할 수 없을 때 사용한다.

인터페이스

인터페이스는 타입을 정의한 규칙을 의미한다.

```
interface User {
  age: number;
```

```
  name: string;
}
```

- 변수와 함수에 활용한 인터페이스

```
var person: User = {
  age: 30,
  name: 'aa'
}

function getUser(user: User) {
  console.log(user);
}
```

- 인덱싱

```
interface StringArray {
  [index: number]: string;
}

var arr2: StringArray = ['a', 'b', 'c'];
arr[0] = 10 //Error;
```

- 딕셔너리 패턴

```
interface StringRegexDictionary {
  [key: string]: RegExp
}

var obj: StringRegexDictionary = {
  cssFile: /\.css$/,
  jsFile: 'a' //Error
}

obj['cssFile'] = /\.css$/;
obj['jsFile'] = 'a' //Error
```

- 인터페이스 확장

```
interface Person{
  name: string;
  age: number;
}

interface User extends Person{
  language: string;
}
```

오퍼레이터

- Union 타입: 자바스크립트의 OR 연산자와 같은 의미의 타입이다. Union 타입으로 지정하면 각 타입의 공통된(보장된) 속성에만 접근 가능하다.

```
function askSomeone(someone: Developer2 | Person) {
  console.log(someone);
}
```

- Intersection 타입: 자바스크립트의 AND 연산자와 같은 의미의 타입이다. 각각의 모든 타입이 포함된 객체를 넘기지 않으면 에러가 발생한다.

```
function askSomeone(someone: Developer & Person) {
  console.log(someone);
}
```

제네릭

한 가지 타입보다 여러 가지 타입에서 동작하는 컴포넌트를 생성하는데 사용된다. 제네릭이란 타입을 마치 함수의 파라미터처럼 사용하는 것을 의미한다.

```
function logText <T> (text: T):T {
  return text;
}

logText<string>('aa');
logText<number>(100);
```

타입 추론

타입 추론이란 타입스크립트가 코드를 해석하는 과정을 뜻한다.

```
var a = true;

a = 100; //Error
```

해당 코드는 a 변수를 Boolean 타입으로 추론했기 때문에 Number타입을 할당하면 에러가 발생한다.

- 가장 적절한 타입(Best Common Type): 배열에 담긴 값들을 추론하여 Union타입으로 묶어 나가는 것을 말한다.

```
var arr = [1, 2, true];
```

타입스크립트는 해당 코드의 타입을 Number | Boolean 으로 정의한다.

- 인터페이스와 제네릭을 이용한 타입 추론 방식

```
interface Dropdown<T>{
  value: T,
  text: 'String'
}

var items: Dropdown<boolean> {
  value: true,
  text: 'aa'
}
```

타입 단언

타입 단언이란 타입스크립트가 해석하는 것보다 더 확실한 목적을 가지고 개발자가 해당 코드에 타입을 직접 지정하는 것을 의미한다.

```
var a;
a = 10;
a = 'string';
var b = a as string;
```

Dom API 조작에서 많이 사용한다.

```
//타입추론시 HTMLDivElement | null로 반환
var div = document.querySelector('div') as HTMLDivElement;
div.innerText;
```

위의 타입 단언으로 null을 대비한 분기문을 작성하지 않아도 된다.

타입 호환

타입 호환이란 특정 타입이 다른 타입에 잘 호환되는지를 의미한다.

- 구조적 타이핑: 코드 구조 관점에서 타입이 서로 호환되는지를 판단하는 것이다. 구조적으로 더 큰 타입은 작은 타입을 호환할 수 없다.

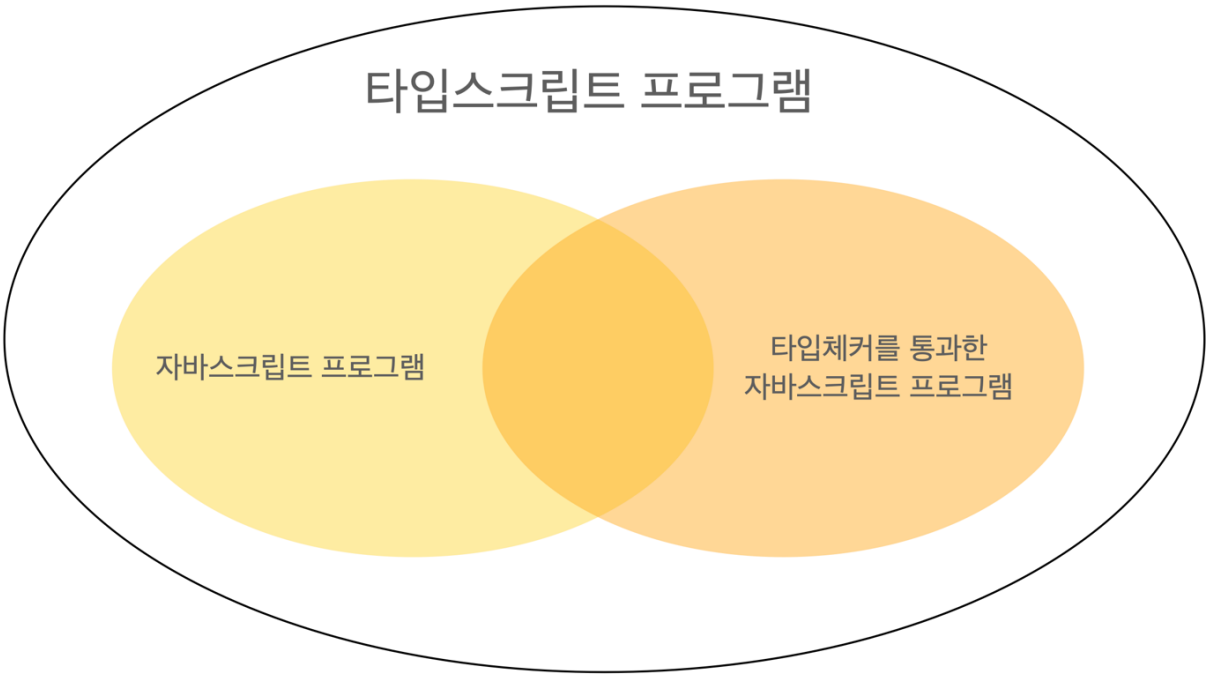
```
interface Developer {
  name: string;
  age: string;
}
interface Person {
  name: string;
}

var developer: Developer;
var person: Person;

developer = person; //Error
person = developer;
```

5. 자바스크립트와 타입스크립트의 관계

모든 자바스크립트는 타입스크립트이지만, 일부 자바스크립트(그리고 타입스크립트) 만이 타입 체크를 통과한다.




```
const a = null + 7; // 자바스크립트에서는 a값이 7이 되지만, 타입체커는 통과하지 못함.
const b = [] + 12; // 자바스크립트에서는 b값이 12가 되지만, 타입체커는 통과하지 못함.
```

자바스크립트 런타임으로 돌리면 정상 동작하지만, 타입체커는 통과 못해서 문제점이 표시되는 코드 예시

코드 생성과 타입은 관계가 없다.

타입 스크립트 컴파일러는 크게 두 가지 역할을 수행한다

1. 신 타입스크립트 / 자바스크립트를 브라우저에서 동작할 수 있도록 구 버전의 자바스크립트로 트랜스파일한다.
2. 코드의 타입 오류를 체크한다.

이 두 가지는 서로 완벽히 독립적으로 일어난다. 이로 인해서,

1. 타입 오류가 있는 코드도 컴파일이 가능하다.
2. 런타임에는 타입 체크가 불가능하다. (타입 체크를 하려면 태그된 유니온, 속성 체크, 클래스를 사용하면 된다.)
3. 런타임의 타입과 선언된 타입이 다를 수 있다.
4. 타입스크립트 타입으로는 함수를 오버로드 할 수 없다.
5. 타입스크립트 타입은 런타임 성능에 영향을 주지 않는다. (타입과 타입 연산자는 트랜스파일 시점에 제거되기 때문)

타입스크립트는 자바스크립트를 모델링하기 위해 *구조적 타이핑을 사용한다.

자바스크립트는 본질적으로 *덕 타이핑 기반이다.

- 덕 타이핑 : 만약 어떤 함수의 매개변수 값이 모두 제대로 주어진다면, 자바스크립트는 그 값이 어떻게 만들어 졌는지 신경쓰지 않고 그대로 사용한다.
- 구조적 타이핑 : 타입스크립트는 이런 동작, 즉 매개변수 값이 요구사항을 만족한다면 타입이 무언인지 신경 쓰지 않는 동작을 그대로 모델링 하는데, 이를 구조적 타이핑을 사용한다.

즉, 구조적 타이핑은 호환의 개념이다

```
interface Vector2D {
  x: number;
  y: number;
}

interface Vector3D {
  x: number;
  y: number;
```

```
  z: number;
}
```

위에서 Vector3D 타입은 Vector2D의 속성을 모두 가지고 있기 때문에 Vector2D타입으로도 인식된다. 타입스크립트는 모두 정확히 같은 속성만을 가지는 타입만을 같은 타입으로 인식할 것이라 생각하지만, 그렇지 않다. 타입스크립트의 타입 시스템에서 타입은 봉인되어 있지 않다.