Seoul Software ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

With. 팀 뤼쳐드

TypeScript

TypeScript란?

- *"Typescript is a **typed superset** of Javascript that compiles to plain Javascript"*

  타입스크립트는 자바스크립트로 컴파일 되는, 자바스크립트의 **타입이 있는 상위집합**이다.

- 2012년에 발표된 오픈 소스 프로그래밍 언어로 Microsoft에 의해 개발됨

https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html#a-typed-superset-of-javascript

TypeScript란?

- 타입이 있는 자바스크립트
- 즉, JavaScript 의 기본 문법에 자료형 체크하는 기능을 추가한 것
- 자바스크립트가 자의적으로 type 해석을 하고 코드를 실행시켰을 때, 의도와 다른 방식으로 쓰이지 않도록 방지
- 정적 파일 언어

  -> 실행하지 않고도 코드 상의 에러를 알려줌 (실시간 디버깅)

TypeScript는 JavaScript Superset!

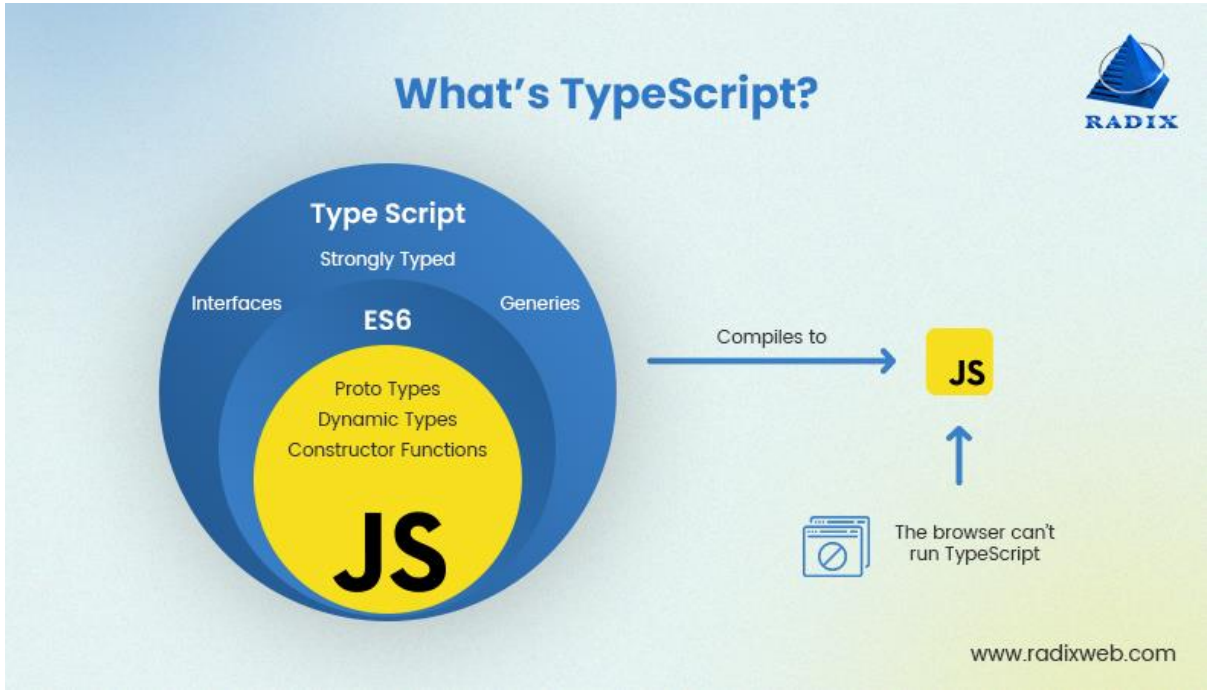
- JavaScript에서 지원하지 않는 기능을 지원!

- TypeScript 대표 기능
  - 엄격한 타입 관리 (Strongly Typed)
  - 제너릭 (Generics)
  - 인터페이스 (Interface)
  - …

트랜스파일러 (Transpiler)

- TS는 웹 브라우저에서 바로 해석될 수 없음
- TS가 JS로 변환되어야 브라우저가 해석 할 수 있음!

- TS가 JS로 출력되기에 트랜스파일러(Transpiler) 라고 부름



JavaScript 의 자료형

- number
- string
- boolean
- null
- undefined
- object

타입과 함께 선언하는 typescript

변수나 함수를 만들어줄 때 타입까지 명시해서 선언

```
let a: number = 1; // 명시적으로 number 타입 지정
let b: string = "안녕하세요"; // 명시적으로 string 타입 지정
let c: object = {
    name: “gildong",
    friends: null,
}; // 명시적으로 object 타입 지정
```

let 변수이름:타입; 으로 선언할 수 있어요! (물론 const 로도 가능하겠죠? ^^)

TS 사용

- 웹 브라우저는 ts 파일을 읽을 수 없기 때문에 ts →js 의 변환 과정이 필요합니다.

1. TypeScript 설치 (전역 설치를 희망한다면 –g 옵션)
   `npm install typescript` (Mac : `sudo npm install typescript`)
2. 설치가 잘 되었는지 version 확인
   `npx tsc -v`
3. tsconfig 파일 생성
   `npx tsc --init`
4. ts 파일을 만들고 js로 변환하고 싶을 때
   `npx tsc 파일이름.ts`

- 실제 사용은 변환된 js 파일을 사용하면 됩니다. (`node 파일이름.js`)

변환 + 실행 자동화

- ts 파일을 일일이 변환한 후에, js 파일을 실행하는게 귀찮아요!
- ts-node 모듈 설치

  `npm install ts-node`

  - package.json 에 ts-node 모듈이 잘 설치되어있는지 확인하기
- 실행은

  `npx ts-node 파일이름.ts`

Ts type 알아보기2 (JS 에서는 없던 type들)

- Tuple
- Enum
- never
- void
- any

Tuple

- JS에서는 배열과 같습니다. (단, 요소의 길이와 타입이 고정된 특수한 배열)
- 순서와 개수가 정해져 있는 배열 (요소의 길이와 타입 고정)
- 일반 배열과 다른 점은 배열의 각각의 타입에 모두 type을 지정해줘야 합니다!
- 순서와 규칙이 있는 배열이 있다면 Tuple을 이용!

```
// 튜플 타입 선언
let drink: [string, number];

// 튜플 초기화
drink = ["cola", 1 ];
```

배열의 element에게 개별적으로 type 선언

drink 라는 배열은 2개의 요소를 가지며
첫번째 요소는 string, 두번째 요소는 number

(즉, 요소들의 타입이 모두 같을 필요는 없음을 나타냄)

Tuple 과 readonly

- 길이와 데이터 타입이 정해진 배열인 tuple
- Readonly ? 읽기만 가능한 data type!

```
let drink3: readonly [string, number] = ["cola", 2500];
```

  - Readonly로 만들어진 tuple 은 데이터를 변경할 수 없음

Enum (열거형)

- 숫자 열거형과 문자 열거형 (특정 값들의 집합)
- 값들에 미리 이름을 정의하고 사용하는 타입

```
enum Auth {
    admin = 0, // 관리자를 0으로 표현
    user = 1,  // 회원은 1로 표현
    guest = 2  // 게스트는 2로 표현
}
```

- 위는 권한 (관리자/사용자/게스트) 별로 숫자값으로 관리를 하는것

Enum (열거형)

```
// 관리자 여부를 숫자로 체크한다.
if (userType !== 0) {
    alert("관리자 권한이 없습니다");
}
```

```
// 회원 권한을 enum으로 정의
enum Auth {
    admin = 0, // 관리자
    user = 1,  // 회원
    guest = 2  // 게스트
}

// Auth.admin(==0) 으로 의미있게 값 체크 가능
if (userType !== Auth.admin) {
    alert("관리자 권한이 없습니다");
}
```

- 0,1,2 로 비교하는 코드를 짜야하지만 개발자가 0,1,2 의 의미를 모두 외우고 있어야 가능함
- 문자열이나 숫자에 미리 의미를 지정해 두고 그룹화할 수 있는 속성
  - Enum으로 정의된 타입 Auth(그룹)
  - Auth에 정의된 0,1,2 를 사용할 때에는 점 접근으로 사용할 수 있음
    Auth.admin / Auth.user / Auth.guest

Enum (열거형) 문법

- JS 의 오브젝트와 유사하지만 선언 이후로는 내용을 추가하거나 삭제할 수 없음
- enum 의 value로는 문자열 혹은 숫자만 허용
- 값을 넣지 않고 선언한다면 숫자형 Enum. 가장 위의 요소부터 0으로 할당돼서 1씩 늘어남

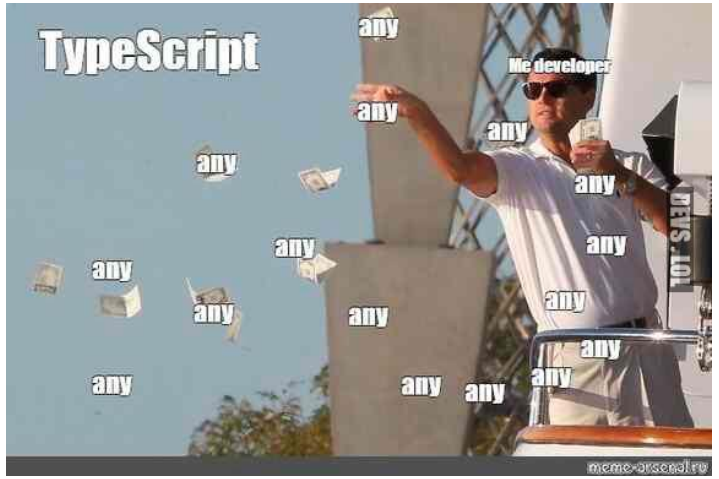
```
enum Cake {
    choco,
    vanilla,
    strawberry,
}
```

```
console.log(Cake.choco); //0
console.log(Cake.vanilla); //1
console.log(Cake.strawberry); //2
```

any (어떤 타입이든)

```
let val:any;
```

- 어떤 타입이든 상관 없이 오류가 나지 않아요!



- 빈 배열을 먼저 선언하고 싶을 때
- 받아오는 데이터 타입이 확실하지 않을 때
- 어떤 타입을 할당해야 할지 알지 못하는 경우 (외부 라이브러리를 사용하거나 동적 콘텐츠를 사용하는 경우)

하지만 정말 어쩔 수 없는 경우가 아니라면 지양!

사용자 정의 type

type

- type 키워드로 복잡한 타입을 type alias (타입 별칭)을 정의
  - 사용자 정의 타입을 만들어줌
- 오브젝트 뿐만 아니라 문자열이나 숫자로 제한을 둘 수 있음

```
type Gender = "Female" | "male";
```

Type 만드는 법

```
type Gender = "Female" | "male";
const gender:Gender="Female"
const gender2:Gender="female"
```

Type을 이용해서 만들어둔 Gender 형으로 변수 선언,
Type에서 설정한 것 이외의 값이 들어오면
코드에서 빨간줄로 틀렸음을 알려줌

interface

interface

- 여러 오브젝트의 타입을 정의하는 규칙

```
interface Student {
    name: string;
    grade: number;
    isPassed: boolean;
}
```

Interface 만드는 법

```
const student1:Student ={
    name:'jh',
    grade:2,
    isPassed:false,
}
```

오브젝트를 선언할 때,
:object 로 쓰는 것이 아닌
:interface로 만든 type을 써준다면,
내부에 있는 key의 type까지 지정할 수 있다.

내가 만들어준 type인 Student! 가 되는 것
(student1이라는 오브젝트를 Student 형으로 만들겠다.)

type vs. interface

type	interface
기존 타입 또는 새 타입을 생성하는데 사용	객체 타입의 구조를 정의하는데 사용
다른 타입 또는 인터페이스를 상속하거나 구현 불가	다른 인터페이스를 상속하거나 구현 가능
리터럴 타입, 유티온 타입, 인터섹션 타입 사용 가능	extends 키워드로 인터페이스 확장 가능

간단한 타입을 정할 때 적합!

객체 구조가 잘 정의되어 있는 경우 적합!

함수에서의 type 선언

함수 선언과 typescript

- 선언시에 타입설정, 호출할 땐 기존처럼

  1. 매개변수 타입설정

  2. 함수의 return 타입에 따라 함수 전체 타입 설정

  (리턴 타입을 보고 타입을 추론할 수 있으므로 생략 가능)

    이미 알고 있는 타입 외에도 never 과 void 를 함수의 리턴 타입으로 설정 가능

선언하는 방식

- 파라미터와 리턴 타입을 선언하는 기본 형식

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

- 화살표 함수로 타입을 선언

```
const sum = (a: number, b: number): number => {
  return a + b;
}
```

- 리턴 생략한 형태로도 선언

```
const sum = (a: number, b: number): number => a + b
```

왼쪽의 sum 함수는 number 형 a, b를 매개변수로 받아 합을 return 하는 함수입니다.

매개변수 a, b의 type 설정

함수의 return type 설정

함수와 매개변수의 개수

JS vs.TS

- JS: 매개변수 선언하고, 호출 시 parameter 전달하지 않아도 오류 x
- TS: 매개변수를 2개 선언했다면, 호출했을 때 반드시 모든 값 전달해줘야 함

```
function sum(a, b, c) {
    return a + b + c;
}
console.log(sum(1, 2));
```

```
function sum(a:number, b:number, c:number) {
    return a + b + c;
}
console.log(sum(1, 2));
```

세 개의 매개변수를 가진 함수 sum 을 만들었지만
함수 호출 시 2개의 parameter 만 전달했기 때문에 오류!

들어오지 않을 변수를 처리하기 위해서는 ? 이용

함수와 매개변수의 개수

- 세번째 매개변수인 c에게 값을 전달하지 않는 경우가 생긴다면 ?를 이용해서 undefined가 될 수도 있음을 정의

```
function print(a: number, b: number, c?: number) {
    console.log(a);
    console.log(b);
    console.log(c);
}
print(1, 2);
```

함수의 리턴 타입이 없을 때 void

- void 란 비어있다는 의미입니다.
- 리턴이 없는 함수는 void로 설정할 수 있습니다.

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

끝이 없는 함수 never

- 어떤 조건에서도 함수의 끝에 도달할 수 없을 때 사용

```
function goingOn(): never {
    while (true) {
        console.log("go");
    }
}
```

함수와 generic 〈T〉

Generic

- 클래스, 함수, 인터페이스에서 다양한 타입으로 재사용 가능
- 선언할 때는 타입 파라미터만 명시하고, 생성 시점에 사용할 타입을 결정!
- 예를들면, 타입을 특정할 수 없는 함수가 있다면?
  - 배열을 매개변수로 받아 배열의 길이를 리턴하는 함수라면

```
function arrLength(arr: (number | string | boolean | object | null)[]): number {
  return arrLength.length;
}
```

```
function arrLength(arr: any[]): number {
  return arrLength.length;
}
```

  - 모든 타입의 데이터 타입이 들어올 수 있으므로 위처럼 사용할 수 있음
  - 하지만 모든 데이터타입을 쓰거나 any를 쓰는 것은 typescript 를 사용하는 이유가 사라짐.

Generic

- 함수를 호출할 때 데이터 타입을 지정할 수 있는 문법 Generic!

- 선언

```
function arrLength2<T>(arr:T[]):number{
    return arr.length
}
```

- 함수 호출

```
arrLength2<string>(["a"]);
arrLength2<number>([1, 2, 3, 4]);
```

  - **함수를 호출할 때** 매개변수로 들어갈 데이터 **타입을 설정**
  - **타입**을 함수의 **파라미터**처럼 사용할 수 있음