TypeScript

TypeScript

- JavaScript 의 기본 문법에 자료형 체크하는 기능을 추가한 것
- 자바스크립트가 자의적으로 type 해석을 하고 코드를 실행시켰을 때,
  의도와 다른 방식으로 쓰이지 않도록 방지
- 정적 파일언어
  -> 실행하지 않고도 코드 상의 에러를 알려줌 (실시간 디버깅)

JavaScript 의 자료형

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

타입과 함께 선언하는 typescript

변수나 함수를 만들어줄 때 타입까지 명시해서 선언

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

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
   `npm install -g typescript` (Mac : `sudo npm install typescript -g`)
2. 설치가 잘 되었는지 version 확인
   `tsc -v`
3. tsconfig 파일 생성
   `tsc --init`
4. ts 파일을 만들고 js로 변환하고 싶을 때
   `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 -g ts-node
  ```

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

  ```
  ts-node 파일이름.ts
  ```

Ts type 알아보기2 (JS 에서는 없던 type들)

- Tuple
- Enum
- never
- void
- any

Tuple

- Js에서는 배열과 같습니다.
- 순서와 개수가 정해져 있는 배열 (요소의 길이와 타입 고정)
- 일반 배열과 다른 점은 배열의 각각의 타입에 모두 type을 지정 해줘야 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

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
- 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
```

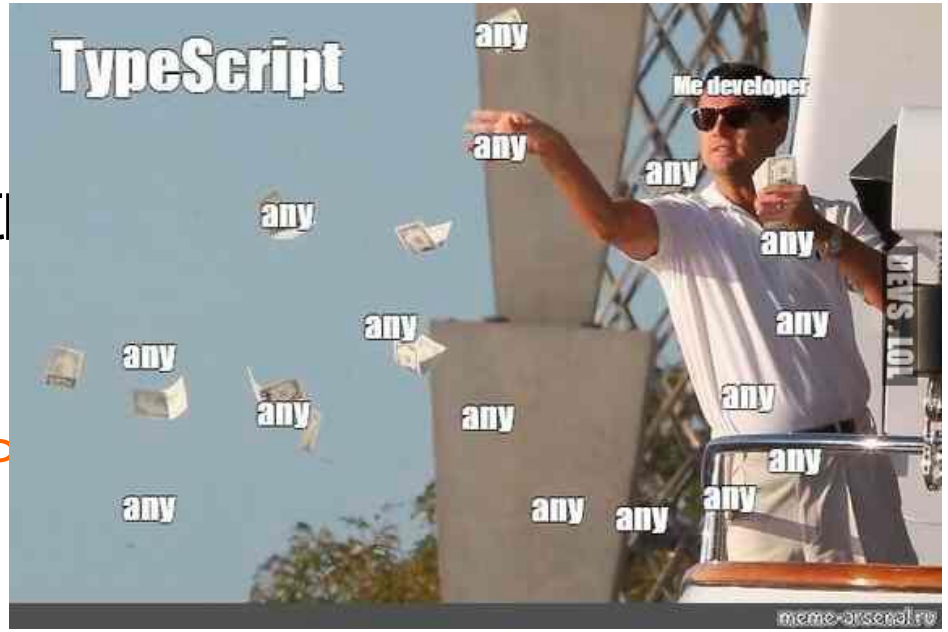
any (어떤 타입이든)

```
let val:any;
```

- 어떤 타입이든 상관 없이 오류가 나지 않아요!

빈 배열을 먼저 선언하고 싶을 때,
받아오는 데이터 타입이 확실하지 않을 때

하지만 정말 어쩔 수 없는 경우가 아니라면 지양



실습

- 오브젝트, 불리언(boolean) 데이터 타입 순으로 설정하는 튜플 만들어보기

```
olimpic_newgame = [
  {
    name: "쇼트트랙",
    type: "혼성 계주",
  },
  true,
];
```

- olimpic_newgame[1]=false; 를 했을 때 변경되지 않도록 수정할 수 없는 데이터 만들기

codingOn

사용자 정의 type

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

- Interface와 마찬가지로 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

실습 (interface 이용)

codingOn

- 화면에 나와잇는 heroGame_A 와 heroGame_B 를 만족할 수 있는 interface Game 만들어보기

```
let heroGame_A:Game = {
    title: 'DC 언체인드',
    price: 50000,
    desc: 'DC 히어로 & 빌런 각각의 개성은 물론, 액션의 재미까지!',
    category: '액션',
    platform: '모바일'
  };

  let heroGame_B:Game = {
    title: 'MARVEL 퓨처파이트',
    price: 65000 ,
    category: '롤플레잉',
    platform: '모바일'
  };
```

함수에서의 type 선언

함수 선언과 typescript

- 선언시에 타입설정, 호출할 땐 기존처럼
  1. 매개변수 타입설정
  2. 함수의 return타입에 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, 호출시 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");
    }
}
```

실습

- 두 개의 수를 매개 변수로 받고 합을 console.log 로 출력하는 함수 sum1 만들기

테스트는 이렇게!

```
sum1(5, 11);
```
// 16이 콘솔창에 출력되는지

실습

(전개 연산자 이용)

- 매개 변수로 여러 개의 수를 받고 전달된 값을 모두 더하는 함수 sum2
- 모두 합산한 값을 return 합니다.

테스트 코드 입력해!

```
console.log(sum2(1, 2, 3, 4, 10));
```

// 20

codingOn

함수와 generic <T>

Generic

- 만약, 타입을 특정할 수 없는 함수가 있다면?
  - 이를테면 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
  - 타입을 함수의 파라미터처럼 사용할 수 있음

실습

- 제네릭 이용해서 함수 arrElement 선언하기
- 배열과 인덱스 번호를 매개변수로 받고,
- Arr[index]에 대한 요소를 리턴하는 함수 만들기,
- 함수의 리턴 타입까지 작성하기

- (선택) 첫번째 인자로 들어간 배열의 길이보다 큰 index 수(두번째 인자)가 전달된다면 return false 시키기

```
console.log(arrElement<string>(["a"], 1)); // false
```

```
function arrElement(배열, 숫자){
    return 배열[숫자]
}
```