

TypeScript

기본

[Chapter 1]

01. 왜 TypeScript를 학습해야 할까요?

- ① 안정성 보장
- ② 개발 생산성 향상
- ③ 협업에 유리
- ④ Java에 점진적 적용 가능

02. 반환값에 타입을 붙이면 그것이 TypeScript

const 변수: 반환값('변수');
↳ '아닌가요?
let name: string = 'kim';

기본적으로 변수이름 뒤에 콜론과 함께 타입 표기

```
<> typescript.html > number
1 let car:string='bmw';          문자열타입
2 let age:number=30;            숫자타입
3 let isAdult:boolean=true;    부리언타입
4 let array1:number[]=[1,2,3]; //대괄호 사용용
5 let array2: Array<number> = [1,2,3]; //제네릭 사용용
6 let age:number=30;
7 age=31; //재할당 가능
8
9 const age:number=30;
10 age=31; //재할당 불가능, 오류 발생생
11
```

03. 함수에서의 TypeScript → (name:string)

parameter (매개 변수) 타입은, 매개변수 바로 뒤에 표기하고, 반환값의 타입은, 파라미터 뒤에 콜론과 함께 예상되는 반환값의 타입을 명시해줍니다.

```
typescript
function greet(name: string): string{
  return 'Hello, ' + name;
}
```

→ 함수선언식

① 화살표 함수

'간결하고 간단한 함수 표현식'을 적용하는 함수 선언 방식

- 화살표 함수 기본 문법

Const 함수명 (매개변수1:타입1, 매개변수2: 타입2) : 반환값 타입 \Rightarrow {}

;

ex) const add (a: number, b: number): number \Rightarrow {}
return a+b;

①-(1) 매개변수 x : const 함수명 ()

→ 매개변수 자리 비워둠

괄호 생략 가능

①-(2) 매개변수가 1개 : const 함수명(매개변수:타입) : 반환값 타입 \Rightarrow {}

①-(3) 함수본문이 1줄 \rightarrow {}, return 생략 가능

<> typescript.html

```
1  const add(A:number,B:number):number=>{  
2      return A+B;  
3  };  
4  
5  const add=add(A,B);  
6  console.log(add);
```

hoisting

② 함수선언식의 특징

디버깅이 용이함

함수가 선언되기 전에도 호출 할 수 있다(hoisting)

영향을 받음

코드 가독성이 좋아 이해하기 쉽다.

반드시 함수 이름을 가지고 있음.(재사용 가능 O)

function으로 시작

③ 화살표 함수의 특징

일반함수에 비해 간결

중복된 매개변수 선언 X

객체를 생성 X(생성자 함수로 호출 X)

익명함수로 많이 실행

04. 리터럴 타입

→ String, number, boolean

TypeScript에서 특정한 값 그 자체만을 허용하는 **타입**을 정의할 수 있는 기능

특정 값 하나만 허용

코드 안정성↑, 오류 방지

① 문자열 리터럴 타입의 기본 개념

문자열 리터럴 타입 → 해당 문자열만 가질 수 있음

예시 1: 문자열 리터럴 타입

```
const name: number = " Matthew";  
const name: "Matthew" = "Matthew";
```

위 코드에서 변수 `name`은 문자열 리터럴 타입

"Matthew" 를 가지고 있습니다. 따라서 이 변수는 "Matthew"라는 값만 가질 수 있으며, 다른 문자열을 대입하면 TypeScript에서 오류를 발생시킵니다.

잘못된 예시

```
const name: "Matthew" = "Yaho";
```

이 코드에서는 변수 `name`이 "Matthew"라는 특정한 값만 가질 수 있도록 정의했기 때문에 "Yaho"를 대입하려고 하면 타입 불일치 오류가 발생합니다.

② 객체 리터럴 타입 (O·L·T)

→ 정해진 값만 가질 수 있음

O·L·T 정의 → 미리 정의된 프로퍼티만 가져야 함

typescript.html

```
1 const person(name:string;age:number)  
2 ={  
3   name: "John",  
4   age: 27 //age에 "yaho"가 들어가면 오류를 발생시킴  
5 };
```

정의된 프로퍼티 외에 추가적인 프로퍼티 허용X

typescript.html

```
1 const person(name:string;age:number)  
2 ={  
3   name: "John",  
4   age: 27 //age에 "yaho"가 들어가면 오류를 발생시킴  
5   hobby:"soccer" // 불필요한 프로퍼티 방지  
6 };
```

③ 인덱스 시그니처(Index Signature)

객체에 추가적인 프로퍼티를 허용하기 위해 사용

[key: string]: any

추가!

TypeScript ▾ 복사 캡션

```
const person: { name: string; age: number; [key: string]: any } = {
    name: "Matthew",
    age: 27,
    job: "Software Developer"
};
```

job 말고도 추가로 들어갈 수 있나...?

위 코드에서는 인덱스 시그니처 [key: string]: any를 사용하여 추가적인 프로퍼티(job 등)를 허용하고 그 값은 any 타입으로 정의했습니다. 이 방식으로 제의 특정 프로퍼티 외에도 임의의 키-값 쌍을 허용합니다.

④ 선택적 프로퍼티

있어도 되고 없어도 되는 프로퍼티

프로퍼티 이름 뒤에 ? 를 붙여 사용

예시: 선택적 프로퍼티

```
const person: { name: string; age?: number } = {
    name: "Matthew"
};
```

⑤ 자바스크립트 객체는 const로 선언되도 수정 가능

05. 배열, 튜플 타입

① 배열

배열타입 정의

```
const array: string[] = ['야호', '하하'];
const array: Array<string> = ['야호', '하하'];
```

② 튜플(Tuple)

각 요소 자리에 고정되어 있는 배열

const tuple: [String, boolean, number] = ["매튜", true, 26];

06. 유니언 타입(Union Types)

둘 이상의 타입을 허용 → 변수가 여러 타입 중 하나를 가질 수 있게 함.

① 유니언 타입의 기본 개념

파이프(|) 기호 사용

두 개 이상의 타입을 결합

유니언 타입의 기본 예시

```
let value: string | number;  
  
value = "Hello"; // 정상  
value = 123; // 정상  
value = true; // 오류: 'boolean' 타입은 허용  
되지 않습니다.
```

위 예시에서 `value` 는 `string` 또는 `number` 타입을 가질 수 있으며, 문자열이나 숫자 값을 대입할 수 있습니다. 그러나 `boolean` 타입의 값을 대입하려고 하면 오류가 발생합니다.

② 유니언 타입의 활용

[함수의 인자) 지정
반환 타입

함수에서 유니언 타입 사용

```
function printValue(value: string | number) {  
    console.log(value);  
}  
  
printValue("Hello"); // 출력: Hello  
printValue(123); // 출력: 123
```

위 함수 `printValue` 는 인자로 문자열과 숫자 모두를 받을 수 있도록 유니언 타입을 지정했습니다. 따라서 호출 시에 문자열이나 숫자를 모두 전달할 수 있습니다.

③ 유니언타입과 조건부 로직(타입 좁히기)

[`typeof`) 변수의 실제 타입 검사, 처리
`instanceof`

④ 유니언 타입과 배열

⑤ 유니언 타입과 랙터럴 타입

유니언 타입이 적용된 배열

```
let mixedArray: (string | number)[] = ["Hello",  
123, "World", 456];
```

위 예시는 `string` 과 `number` 타입을 모두 허용하는 배열을 선언한 것입니다. 배열에 문자열과 숫자가 섞여 있을 수 있으며, 각각의 값이 해당 타입에 맞는지 체크할 수 있습니다.

07. 타입스크립트에만 존재하는 타입

- ① any
- ② unknown
- ③ void
- ④ never

08. Type Aliases (타입 별칭)

특정 타입이나 인터페이스를 참조 할 수 있는 타입 변수

```
typescript                                ⌂ 복사
type MyType = string | number;

위와 같이 type 키워드를 사용하여 MyType 이라는 별칭을 만들면,
string | number 타입을 사용할 때마다 직접 쓰지 않고 MyType 을 대신 사용할 수 있음.
```

① Union type 을 타입 별칭으로!

```
typescript

// ✅ Type Alias 사용
type UserRole = "admin" | "editor" | "viewer" | "guest";
let userRole: UserRole;
```

```
Type Aliases 를 객체에 사용할 경우 매우 유용하다.

type UMC = {
  nickname: string;
  part: string;
}

let member: UMC = { nickname: 'Matthew', part: 'WEB' }

& 인산자 를 활용하여, object 타입을 합칠 수 있습니다.

type TNickname = { nickname: string }
type TName = { name: string }

type TMember = TNickname & TName;

let me: TMember = {
  name: '김용민',
  nickname: '김용민'
}
```

09. Interface 구체 타이핑

① 인터페이스 병합

```

interface UMC {
  name: string;
  nickname: string;
}

interface UMC {
  skill: string;
}

자동병합
let member: UMC = { name: '김용민', nickname: '매튜', skill: ' '

```

같은 이름을 가진 여러 인터페이스 → 자동 병합

(1) 자동병합

(2) 학장성

(3) 타입 안전성 유지

* 오류

interface UMC {

name: string;

?

interface UMC {

] type이

달라서 오류 발생

name: number;

?

② 네임 스페이스

문제발생

인터페이스 자동 병합 → 의도치 않게 다른 사람것도 같이

010. Generic(제네릭)

함수, 타입, 클래스 등에서 사용할 구체적인 타입을 미리 정하지 않고 타입 변수를 사용해 유연하게 처리할 수 있는 방식

제네릭의 사용법

- 함수나 클래스의 매개변수처럼, 타입의 자리를 비워 두고, 사용하는 순간 타입을 지정합니다.

```

function identity<T>(arg: T): T {
  return arg;
}

// 시중 시장에 나온 시장 (number -> 수시 대입)
let result = identity<number>(42); // T = number

```

1. <T> (제네릭 타입 선언)

- 이 부분에서 **T**는 제네릭 타입 변수를 선언하는 것입니다.
- <T>**는 이 함수가 제네릭 타입을 사용할 것을 명시합니다. 즉, **T**는 아직 구체적인 타입이 정해지지 않았고, 함수가 호출될 때 외부에서 지정된 타입이 들어옵니다.

2. **arg: T** (매개변수 타입)

- 함수의 매개변수 **arg**는 **T** 타입을 가집니다.
- 즉, 첫 번째 **<T>**에서 선언된 제네릭 타입을 이 매개변수의 타입으로 사용하겠다는 의미입니다. 함수가 호출될 때 **T** 자리에 어떤 타입이 지정되면 **arg**는 그 타입을 따릅니다.

3. **: T** (반환 타입)

- 이 부분은 함수의 반환 타입을 나타냅니다.
- 함수의 반환 값 또한 **T** 타입이어야 함을 의미합니다. 따라서 이 함수는 매개변수로 받은 **T** 타입의 값을 그대로 반환하게 됩니다.

011. Enum (열거형) ↗ 이것도 type?인가?

이름이 있는 상수 집합을 정의하는 방법

① 숫자형

첫번째 값은 0부터 시작하며, 이후 값들은 자동으로 1씩 증가

첫번째 값 수동 설정 가능, 양방향 매팅

```
enum Direction {  
    Up,      // 0  
    Down,    // 1  
    Left,    // 2  
    Right   // 3  
}  
  
let dir: Direction = Direction.Up;  
console.log(dir); // 0
```

- 여기서 `Direction.Up`의 값은 0이고, `Direction.Down`은 1로 자동으로 증가합니다.
- 만약 특정 값으로 시작하고 싶다면, 첫 번째 값에 수동으로 설정할 수 있습니다.

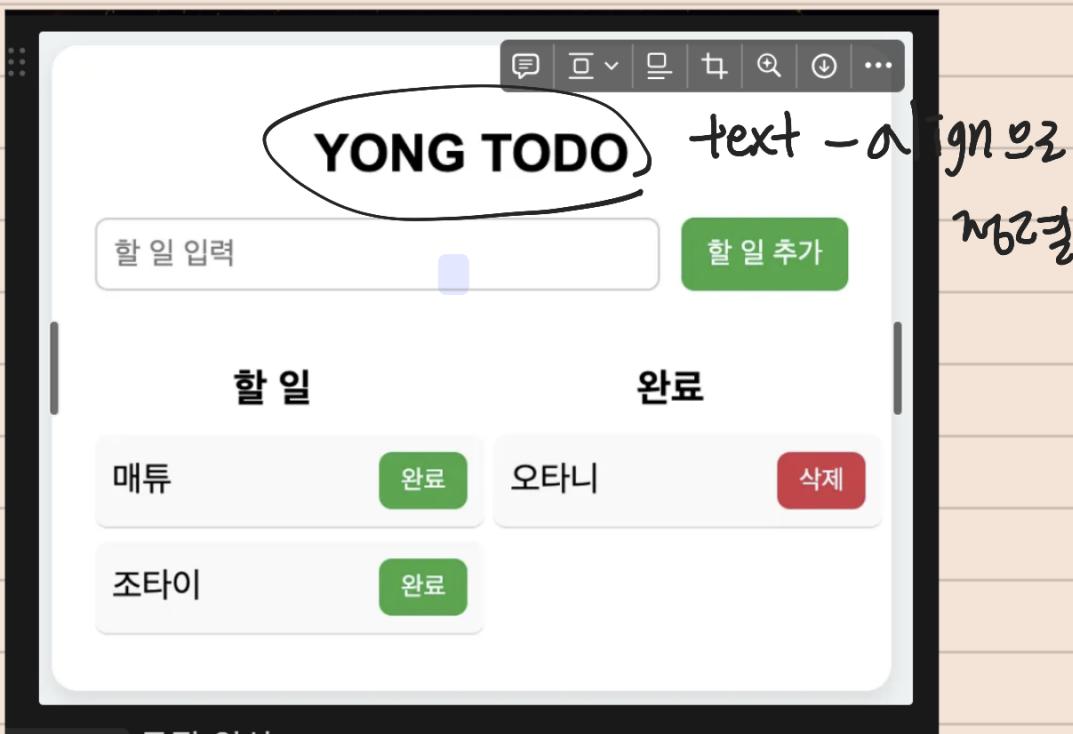
```
enum Status {  
    Success = 1, // 1  
    Failure,     // 2  
    Pending      // 3  
}
```

② 문자형

값들을 수동으로 문자열로 지정

```
enum Color {  
    Red = "RED",  
    Green = "GREEN",  
    Blue = "BLUE"  
}  
  
let myColor: Color = Color.Green;  
console.log(myColor); // "GREEN"
```

02. Utility Type



입력 → form HTML

▪ ex5-7.html

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>로그인 품</title>
6   </head>
7   <body>
8     <form name="loginform" method="post">
9       사용자 아이디: <input type="text" name="username"><br>
10      비밀번호: <input type="password" name="password"><br>
11      <input type="submit" value="로그인">
12    </form>
13  </body>
14 </html>
```

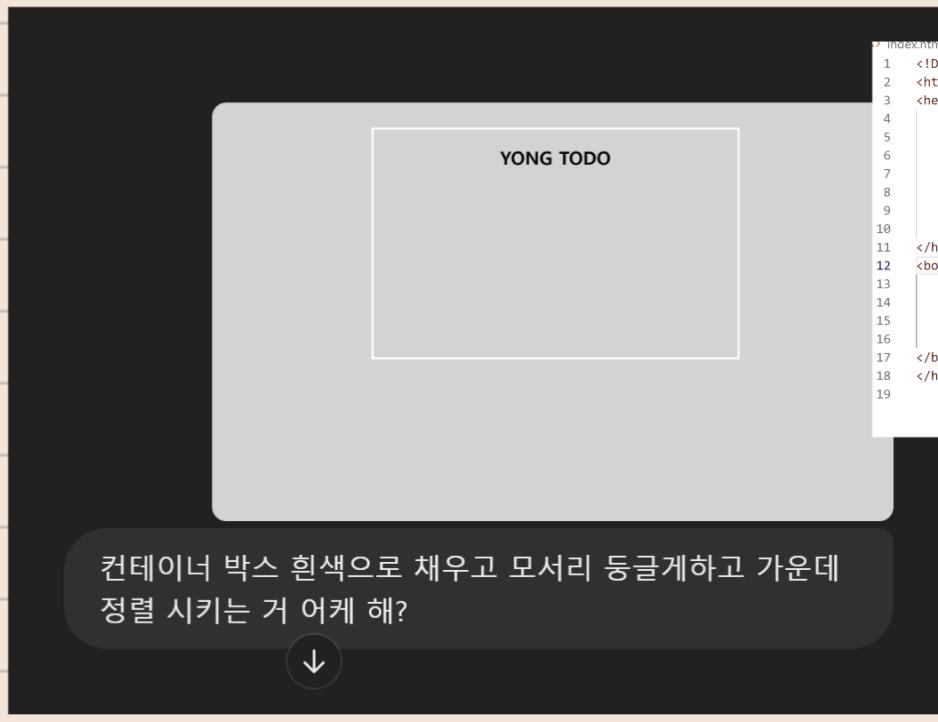
할일추가 버튼

완료, 삭제 버튼

YONG
TODO (?)

```
color: black;
text-align: center; // 가운데 정렬
font-weight: 700; // 글자 두께
font-size: 30pt; // 글자 크기
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>YONG TODO</title>
    <style>
        h2 {
            color: black;
            text-align: center;
            font-weight: 700;
            font-size: 30pt;
        }
    </style>
</head>
<body>
    <h2>YONG TODO</h2>
    <div class="outer-div">
        <div class="inner-div">
        </div>
    </div>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0"/>
    <!--css 파일 임포트-->
    <link rel="stylesheet" href="./style.css"/>
    <!--js 파일 임포트-->
    <script type="module" src="./dist/script.js" defer></script>
    <title>YONG TODO</title>
</head>
<body style="background-color: #lightgrey;">
    <div class="container-box">
        <h2>YONG TODO</h2>
    </div>
</body>
</html>
```

```
# style.css > .container-box
1  h2{
2      color: black;
3      text-align: center;
4      font-weight: 700;
5      font-size: 30pt;
6  }
7
8
9  .container-box {
10     width: 800px;
11     height: 500px;
12     border: solid 5px #ffff;
13     margin-top: 100px;
14     margin-left: auto;
15     margin-right: auto;
16     text-align: center;
17 }
```

```
#outer{ width:300px; height:280; position:relative;}
```