# Overview Of Typescript_Types of Ts



## 메모

- 명시적 타입 추론은 최소화

# Types of Ts 1

### ▼ 객체 생성

```
const nicoplayer : {
    name : string,
    age ?: number
} = {
    name : "nico"
}


if(nicoplayer.age && nicoplayer.age > 10){

}

//아래와 같이 작성하면 오류가 나는 이유
// if(player.age && player.age > 10){

// }
/*
아래와 같이 작성하면 오류가 나는 이유
    if( player.age > 10){

    }
```

```
player의 age는 undifined가 될 수 있기 때문에 조건 (player.age &&)을
추가해주어야 한다.
 */
```

## ▼ Alias 타입

```
type Age = number;
type Player = {
    name : string,
    age ?: Age
}

const pnico : Player = {
    name : "nico"
}

const plynn : Player = {
    name : "lynn",
    age : 12
}
```

## ▼ function return

```
/*
function playerMaker(name: string) {
    return {
        // name: name
        name
    }
}

const nico = playerMaker("nico")
nico.age = 12;

//이렇게 하면 오류 발생!
 */


//Player 클래스로 반환함을 알려주면 명시되어 있지 않던
//age속성 또한 Player 클래스에 들어있기 때문에 사용할 수 있다.
```

```
function playerMaker(name: string) : Player{
   return {
      // name: name
      name
   }
}

const nico = playerMaker("nico")
nico.age = 12;
```

▼ 축약

```
type Age = number;
type Name = string;
type Player = {
   name : string,
   age ?: Age
}

const playerMaker = (name: string) : Player => ({name})
const nico = playerMaker("nico")
nico.age = 12
```

# Types of Ts 2

### ▼ read only 속성 (immutability_불변성)

```
type Age = number;
type Name = string;
type Player = {
   readonly name : Name,
   age ?: Age
}

const playerMaker = (name: string) : Player => ({name})
const nico = playerMaker("nico")
nico.age = 12
nico.name = "las"   //오류발생


const numbers: readonly number[] = [1,2,3,4]
number.push(1)  // 수정할 수 없다.
```

## ▼ Tuple 타입

```
//정해진 갯수의 요소를 가져야하는 array를 지정할 수 있다.
const player:[string, number, boolean] = ["nico", 1, true]
player[0] = 1    //player[0]은 문자열 "nico"로 숫자가 들어갈 수 없다.

//readonly 사용
const player01: readonly [string, number, boolean] = ["nico", 1, true]
player01[0] = "hi"    //readonly속성으로 수정 할 수 없다.
```

## ▼ any 타입

```
/*
any타입은 ts로부터 빠져나올 때 사용
아무 타입이나 사용 가능
any는 사용을 권장하지는 않음
*/

let a : undefined = undefined   //undefined
let b : null = null             //null
let c : any = []                //any

type Player = {
   age ?: number   //undefined | number
}

// any타입의 연산
const aa : any[] = [1,2,3,4]
const bb : any = true

aa + bb    //이 연산은 가능하다
```

# Types of Ts 3

## ▼ unknown

```
//어떤 타입인지 모르는 변수일 경우 사용
//예) API로부터 응답을 받는데 그 응답의 타입을 모를 경우

let a : unknown;
//ts의 보호를 받아 어떤 작업을 하고자 할 경우
```

```
//이 변수의 타입을 먼저 확인해야하는 방식으로 사용해야한다.

//unknown변수는 if문을 통한 변수 a의 타입을 확인 후 사용한다.
if(typeof a === 'number'){  //숫자 타입 확인 조건문
   let b = a + 1       //숫자 연산 적용 가능
}

a.toUpperCase() //조건이 없는 경우, 사용할 연산 적용이 어렵다.

if(typeof a === 'string'){  //문자 타입 확인 조건문
   let b = a.toUpperCase();    //문자 관련 연산 적용 가능
}
```

## ▼ void

```
// 함수 hello는 아무것도 return하지 않는다.
function hello(){
   console.log('x');
}

const a = hello();
a.toUpperCase(); //허용되지 않는 작업
```

## ▼ never

```
//함수가 절대 return을 하지 않을 때 발생

//상황1
//함수에서 exception(예외)이 발생할 때
function hello() : never {
   return "X"  //오류발생
}

//return 하지 않고 오류를 발생시키는 함수
function hello() : never {
   throw new Error("xxx")  //정상작동
}



//상황2
//타입이 두가지 일 수도 있는 상황에 발생
function hello(name : string | number) {
  name + 1 //이 연산은 name이 string일 수도 있기 때문에 불가한 연산이다.
}

function hello(name : string | number) {
   if(typeof name === 'string'){
```

```
        name    //여기서는 string type
    } else if (typeof name === "number"){
        name    //여기서는 number type
    } else {
        name    //여기서는 never type
    }

}
```