TypeScript Function

TypeScript

TypeScript Function

1. 함수란?

함수는 JavaScript의 응용 프로그램을 구성하는 기본 요소이다. 그리고 함수는 추상화 계층, 클래스, 정보 숨기기, 모듈을 모방하는 방법이다. TypeScript에서는 클래스, 네임 스페이스 및 모듈이 있지만 함수는 여전히 작업 수행 방법을 설명하는 데 중요한 역할을 한다. 또한 TypeScript는 표준 JavaScript 함수에 몇 가지 새로운 기능을 추가하여 보다 쉽게 작업할 수 있다.

JavaScript와 마찬가지로, 명명 함수(Named function) 또는 익명 함수(Anonymous function)로 TypeScript 함수를 생성할 수 있다. JavaScript와 마찬가지로 TypeScript 함수는 함수 본문 외부의 변수를 참조할 수 있다.

```
let num: number = 2;

function onAvg(kor, eng) {
	return (kor + eng) / num;
}
```

TypeScript

TypeScript Function Type

2. 함수 타입

함수에 각 파라미터에 타입을 추가 하고, 리턴 타입을 추가하여 함수에 타입을 부여할 수 있다. TypeScript은 return 문을 보고 리턴 타입을 파악할 수 있으므로 많은 경우에 선택적으로 리턴 타입을 생략할 수도 있다.
함수 파라미터 타입이 있으면 함수 타입에 파라미터를 지정하는 이름에 상관없이 함수의 유효한 타입으로 간주된다.

반환 타입은 함수가 값을 반환하지 않는다면 반환 타입으로 void를 지정할 수 있다.

```
let num: number = 2;
function onAvg(kor: number, eng: number): number {
	return (kor + eng) / num;
}

function onAvg(kor: number, eng: number): void {
	console.log( (kor + eng) / num );
}
```

TypeScript

TypeScript Function Type

3. 함수 타입 추론

함수 내부에 익명 함수가 기술된 경우 TypeScript 컴파일러에 의해 타입을 추정할 수 있다.
이를 타입 추론의 한 형태인 컨텍스트 타입 지정이라 한다.

```
function onAdd(kor: number, eng: number): number{
	return kor + eng;
}

let onAdd = function(kor: number, eng: number): number {
	return kor + eng;
};

let onAdd: (kor: number, eng: number) => number = function(kor, eng) {
	return kor + eng;
};
```

TypeScript

TypeScript Function Type

4. Optional 파라미터

JavaScript에서 모든 파라미터는 선택 사항이며 사용자가 적합하다고 생각되는 파라미터 값을 생략 할 수 있다.
이 경우 그 파라미터는 undefined이다. TypeScript에서 이러한 Optional 파라미터는 파라미터 끝에 ? 를 추가하여 구현 할 수 있다. **이러한 모든 Optional 파라미터는 Required 파라미터 다음에 나와야 한다.**

```
function onAdd(kor: number, eng: number, math?: number ): number{
    math = (math === undefined) ? 0 : math;
    return kor + eng + math;
}
let onAdd: (kor: number, eng: number, math?: number ) => number = function(kor, eng, math){
    math = (math === undefined) ? 0 : math;
    return kor + eng + math;
}

onAdd(100, 90);
onAdd(100, 90, 80);
```

TypeScript

TypeScript Function Type

5. Default 파라미터

TypeScript에서 사용자가 파라미터를 입력하지 않거나 파라미터 값 대신 undefined를 전달한 경우 파라미터에 할당 될 값을 Default 설정 값으로 설정할 수 있다. 이를 Default-initialized 파라미터라고 한다.

Required 파라미터 다음에 오는 Default-initialized 파라미터는 Optional 파라미터처럼 취급되며 Optional 파라미터처럼 해당 함수를 호출할 때 생략할 수 있다. 이것이 Default-initialized 파라미터가 Optional 파라미터와 공유하는 특성입니다.

```
function onAdd(kor: number, eng: number, math: number = 0 ): number{
        return kor + eng + math;
}

onAdd(100, 90);
onAdd(100, 90, 80);
```

TypeScript

TypeScript Function Type

5. Default 파라미터

일반 Optional 파라미터와 달리 Default-initialized 파라미터는 Required 파라미터 다음에 올 필요가 없다. Default-initialized 파라미터가 Required 파라미터 앞에 오는 경우 사용자는 명시적으로 undefined를 전달하여 Default-initialized된 값을 가져 오도록 해야한다.

```
function onAdd(math: number = 0, kor: number, eng: number): number{
    return kor + eng + math;
}

onAdd(undefined, 100, 90);
onAdd(100, 90, 80);
```

TypeScript

TypeScript Arrow Function

6. Arrow 함수

화살표 함수 표현(**arrow function expression**)은 function 표현에 비해 구문이 짧고 자신의 this, arguments, super 또는 new.target을 바인딩 하지 않는다. 화살표 함수는 항상 익명이다. 이 함수 표현은 메소드 함수가 아닌 곳에 가장 적합하다. 그래서 생성자로서 사용할 수 없다.

```
console.log(materials.map(material => material.length));
```

화살표 함수는 일반 함수와 3가지 측면에서 다르다.

1. 화살표 함수는 항상 바인딩 된 this를 가진다.
2. 화살표 함수는 생성자로 사용할 수 없다. 이유는 Construct(new 키워드와 일반 함수가 함께 실행될 수 있게 해줌)라는 내부 메소드와 prototype 속성이 없기 때문이다. 그러므로 new (() => {}) 는 에러를 반환한다.
3. 화살표 함수는 ES6의 새로운 구조이기 때문에 새로운 방식의 인수 조작(기본 매개변수, 나머지 매개변수 등등)이 가능하고, 더이상 arguments 키워드를 지원하지 않는다.

```
let arrow = (name: string) => 'Hello ' + name;
```