Understanding Type Annotations in TypeScript

Summary: in this tutorial, you will learn about type annotations in TypeScript.

What is Type Annotation in TypeScript

TypeScript uses type annotations to specify explicit types for identifiers such as variables, functions, objects, etc.

TypeScript uses the syntax `: type` after an identifier as the type annotation, which `type` can be any valid type.

Once an identifier is annotated with a type, it can be used as that type only. If the identifier is used as a different type, the TypeScript compiler will issue an error.

Type annotations in variables and constants

The following syntax shows how to specify type annotations for variables and constants:

```
let variableName: type;
let variableName: type = value;
const constantName: type = value;
```

In this syntax, the type annotation comes after the variable or constant name and is preceded by a colon (`:`).

The following example uses `number` annotation for a variable:

```
let counter: number;
```

After this, you can only assign a number to the `counter` variable:

```
counter = 1;
```

If you assign a string to the `counter` variable, you'll get an error:

```
let counter: number;
counter = 'Hello'; // compile error
```

Error:

```
Type '"Hello"' is not assignable to type 'number'.
```

You can both use a type annotation for a variable and initialize it in a single statement like this:

```
let counter: number = 1;
```

In this example, we use the number annotation for the `counter` variable and initialize it to one.

The following shows other examples of primitive type annotations:

```
let name: string = 'John';
let age: number = 25;
let active: boolean = true;
```

In this example, the `name` variable gets the `string` type, the `age` variable gets the `number` type, and the `active` variable gets the `boolean` type.

Type annotation examples

Arrays

To annotate an array type you use a specific type followed by a square bracket `: type[]` :

```
let arrayName: type[];
```

For example, the following declares an array of strings:

```
let names: string[] = ['John', 'Jane', 'Peter', 'David', 'Mary'];
```

Objects

To specify a type for an object, you use the object type annotation. For example:

```
let person: {
  name: string;
  age: number;
};

person = {
  name: 'John',
  age: 25,
}; // valid
```

In this example, the `person` object only accepts an object that has two properties: `name` with the `string` type and `age` with the `number` type.

Function arguments & return types

The following shows a function annotation with parameter type annotation and return type annotation:

```
let greeting : (name: string) => string;
```

In this example, you can assign any function that accepts a string and returns a string to the `greeting` variable:

```
greeting = function (name: string) {
    return `Hi ${name}`;
};
```

The following causes an error because the function that is assigned to the `greeting` variable doesn't match its function type.

```
greeting = function () {
    console.log('Hello');
};
```

Error:

```
Type '() => void' is not assignable to type '(name: string) => string'. Type 'void' is not
```

Summary

- Use type annotations with the syntax `: [type]` to explicitly specify a type for a variable, function, function return value, etc.