TypeScript any Type

Summary: in this tutorial, you will learn about the TypeScript `any` type and how to use it properly in your code.

Introduction to TypeScript any type

Sometimes, you may need to store a value in a variable. But you don't know its type when writing the program. And the unknown value may come from a third-party API or user input.

In this case, you want to **opt out of the type checking** and allow the value to pass through the compile-time check.

For example:

```
let result: any;

result = 1;
console.log(result);

result = 'Hello';
console.log(result);

result = [1, 2, 3];
const total = result.reduce((a: number, b: number) => a + b, 0);
console.log(total);
```

Output:

In this example:

- First, declare the variable `result` with the type `any`.
- Second, assign number 1 to the `result` and display its value to the console.
- Third, assign the string literal `'Hello'` to the `result` and show its value to the console.
- Finally, assign an array of numbers to the `result` variable, calculate the `total` using the `reduce()` method, and log the `total` to the console.

Let's take another typical example:

```
// json may come from a third-party API
const json = `{"latitude": 10.11, "longitude":12.12}`;

// parse JSON to find location
const currentLocation = JSON.parse(json);
console.log(currentLocation);
```

Output:

```
{ latitude: 10.11, longitude: 12.12 }
```

In this example, TypeScript infers the value of the `currentLocation` variable as `any`. We assign an object returned by the `JSON.parse()` function the `currentLocation` variable.

However, when we access the non-existing property (`x`) of the `currentLocation` variable, TypeScript does not carry any checks.

This is working fine and shows an `undefined` value in the console:

```
console.log(currentLocation.x); // undefined
```

Output:

```
undefined
```

The TypeScript compiler doesn't complain or issue any errors.

The `any` type provides you with a way to work with the existing JavaScript codebase. It allows you to gradually opt in and opt out of type-checking during compilation. Therefore, you can use the `any` type for migrating a JavaScript project over to TypeScript.

TypeScript any: implicit typing

If you declare a variable without specifying a type, TypeScript assumes that you use the `any` type. This feature is called type inference. TypeScript guesses the type of the variable. For example:

```
let result;
```

In this example, TypeScript infers the type for you. This practice is called implicit typing.

> Note that to disable implicit typing to the `any` type, you change the `noImplicitAny` option in the `tsconfig.json` file to true. You'll learn more about the `tsconfig.json` in the later tutorial.

TypeScript any vs. object

If you declare a variable with the `object` type, you can also assign any value to it. However, you cannot call a method on it even if the method exists. For example:

```
let result: any;
result = 10.123;
console.log(result.toFixed());
result.willExist(); //
```

In this example, the TypeScript compiler doesn't issue any warning even the `willExist()` method doesn't exist at compile time because the `willExist()` method might be available at runtime.

If you run the code, you'll see the following error message on the console window:

```
TypeError: result.willExist is not a function
```

However, if you change the type of the `result` variable to `object`, the TypeScript compiler will issue two errors:

```
let result: object;
result = 10.123;
result.toFixed();
```

Errors:

```
app.ts:2:1 - error TS2322: Type 'number' is not assignable to type 'object'.

2 result = 10.123;
  ~~~~~~

app.ts:3:8 - error TS2339: Property 'toFixed' does not exist on type 'object'.

3 result.toFixed();
         ~~~~~~~

Found 2 errors in the same file, starting at: app.ts:2
```

Summary

- The TypeScript `any` type allows you to store a value of any type. It instructs the compiler to skip type-checking.
- Use the `any` type to store a value that you don't know its type at the compile-time or when you migrate a JavaScript project over to a TypeScript project.