TypeScript

This tutorial is based on my JavaScript tutorial.
Please be aware that I **won't** get over JS basics again.

Introduction

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Lexicon

NAME	DESCRIPTION	USAGE

Type assertions

Type assertions are a way to tell the compiler the type of a value when TypeScript cannot infer it automatically.

1. AS keyword

Allows you to tell the compiler the type of a value.

```
const myValue: unknown = "Hello, world!";
const myString: string = myValue as string;
```

2. Angle-bracket syntax

Not recommended for use in JSX or TSX files because it conflicts with JSX syntax. – *ref: good practices*

```
const myValue: unknown = "Hello, world!";
const myString: string = <string>myValue;
```

3. Non-null assertion

We tell the **myValue** constant that it can be asserted as a string or null type.

```
const myValue: string | null = null;
const myString: string = myValue!;
```

4. Type guards

4.1 Is String

```ts
function isString(value: unknown): value is string {
    return typeof value === "string";
}
```

4.2 Is Array<T>

```ts
function isArrayOfType<T>(value: unknown): value is Array<T> {
    return Array.isArray(value) && value.every((item) => typeof item === typeof T);
}
```

4.3 Is Record<K, T>

```ts
function isRecordOfType<K extends string, T>(value: unknown): value is Record<K, T> {
    return typeof value === "object" && value !== null && Object.values(value).every((item) => typeof item === typeof T);
}
```

4.4 Is keyof T

```ts
function isValidKey<T>(key: unknown, obj: T): key is keyof T {
    return key in obj;
}
```

4.5 Is T extends U ? X : Y

```ts
function isType<T, U, X, Y>(value: T, type: U): value is T extends U ? X : Y {
    return value instanceof type;
}
```

Bitwise operators

OPERATOR	DESCRIPTION
&	It performs a Boolean AND operation on each bit of its integer arguments.
|	It performs a Boolean OR operation on each bit of its integer arguments.
^	It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.
~	It is a unary operator and operates by reversing all the bits in the operand.
<<	It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros.
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.
>>>	This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

```
var a: number = 2;   // Bit representation 10
var b: number = 3;   // Bit representation 11
var result;

result = (a & b);
console.log("(a & b) => ", result)

result = (a | b);
console.log("(a | b) => ", result)

result = (a ^ b);
console.log("(a ^ b) => ", result);

result = (~b);
console.log("(~b) => ", result);

result = (a << b);
console.log("(a << b) => ", result);

result = (a >> b);
console.log("(a >> b) => ", result);

// (a & b) =>  2
// (a | b) =>  3
// (a ^ b) =>  1
// (~b) =>  -4
// (a << b) =>  16
// (a >> b) =>  0
```

Parameters

Optional parameters

Here, we can pass a name or nothing when calling the function.

```
function Name(name?: string) {
    if (name) {
        return name;
    }
    return "no name";
}
```

Rest parameters

Rest parameters don't restrict the number of values that you can pass to a function. However, they must all be of the same type.

In OOP programming, it is often used if we don't know if we're passing an array, a single value or a null value of the type.

```
function Name(...name:string[]) {
    if (name) {
        return name;
    }
    return "no name";
}
```

Default parameters

```
function Name(name:string = "Jhon") {
    if (name) {
        return name;
    }
    return "no name";
}
```

Tuples

A tuple is a finite ordered list that can serve more than one data type.

```
var mytuple = [10, "Hello"];
```

Union

Gives programs the ability to combine one or two types.

```
var val:string|number
val = 12
console.log("numeric value of val "+val)
val = "This is a string"
console.log("string value of val "+val)
```

```
var val:string|string[]
```

Namespaces

TypeScript allows defining namespaces ☺.

A namespace is a way to logically group related code.

```
namespace SomeNameSpaceName {
    export interface ISomeInterfaceName { }
    export class SomeClassName { }
}
```

** You can also make nested namespaces but I don't recommend it.*

Modules

Internal modules

Old way (with JS)

```
module TutorialPoint {
    export function add(x, y) {
        console.log(x + y);
    }
}
```

New way (with TS)

```
namespace TutorialPoint {
    export function add(x, y) {
        console.log(x + y);
    }
}
```

Ambients

When you are consuming a bunch of third party js libraries like nodejs you can't rewrite it in TypeScript.
Ensuring typesafety and intellisense while using these libraries will be challenging for a TypeScript programmer. Ambient declarations help to seamlessly integrate other js libraries into TypeScript.

Extension

The extension for an ambient file is **".d.ts"**.

Syntax

```
declare module Module_Name {

}
```

Example

Assume you been given a third party JavaScript library which contains code similar to this:

```
var hi;
(function (hi) {
    var Hello = (function () {
        function Hello() {
        }
        Hello.prototype.Hello = function (message) {
            return message;
        }
    });
});
```

As a typescript programmer, you don't want to rewrite this library to typescript. Still you need to use the Hello() method with type safety.
We can achieve this in our ambient file.

```
declare module hi {
    export class Message {
        Hello(messagae: string): string;
    }
}
```

Good practices

Angle brackets syntax

Do **not** use them in JSX or TSX files because it conflicts with JSX syntax.

Made by :

Matthias Brat

Socials:

github.com/matthiasbrat

stackoverflow.com/users/17921879/matthias-b

matthiasbrat.dev