

Overloading von Funktionssignaturen

```
export class Test{

    public add( a: Number, b:Numbe r) :void;
    public add( a: String, b:Stri ng) :void;
    public add( a: boolean, b:bool -
ean ):void;
    public add( a, b):void{
        con sol e.l og( a+b);
    }
    public static run():any{
        let t:Test = new Test();
        t.a dd( 'a' , 'b');
        t.a dd( 3,5);
        t.a dd( tru e, t rue);
    }
}
Test.r un();
Erzeugt die Ausgaben:
ab
8
2
```

<https://www.bennadel.com/blog/3339-using-method-and-function-overloading-in-typescript.htm>

Typescript Bindungsarten

[xxx]	@Input	Input Parameter
(xxx)	@Output	Callback Parameter

Conditional Types

```
declare type StringOrNull<T> = T extends string
? string
: T extends null
? null
: never;
export function upperC ase <T extends string |
null>( text: T): String OrN ull <T>;
export function upperC ase (text: string | null):
string | null {
    if (typeof text === 'string') {
        return text.t oUp per Case();
    }
    return null;
}
// String
upperC ase ('' ).t oLo cal eLo wer Case();
// Null
upperC ase (null);
// Union
let maybe: string | null = null as any;
upperC ase (ma ybe);
```

<https://speakerdeck.com/gregonnet/conditional-types>



By **Huluvu424242**
(FunThomas424242)

Published 2nd July, 2019.

Last updated 7th July, 2019.

Page 1 of 1.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

cheatography.com/funthomas424242/

github.com/Huluvu424242