TS Überblick

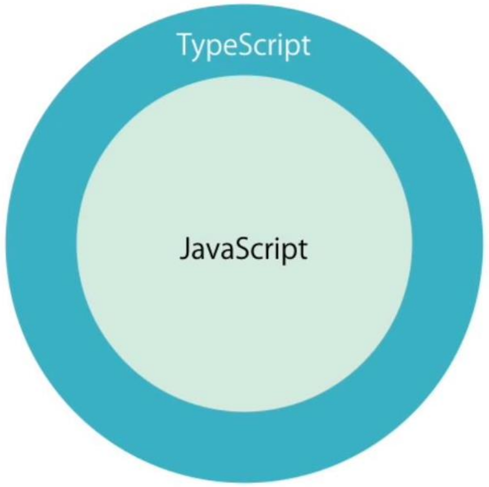
Übermenge von JavaScript

Streng typisiert

- Compiletime-Errors

Objektorientiert

Gute Toolunterstützung

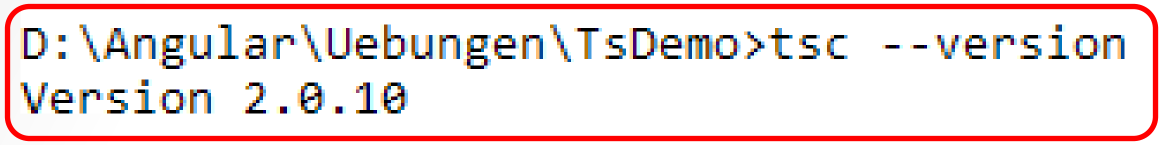


HTL Leonding
next level

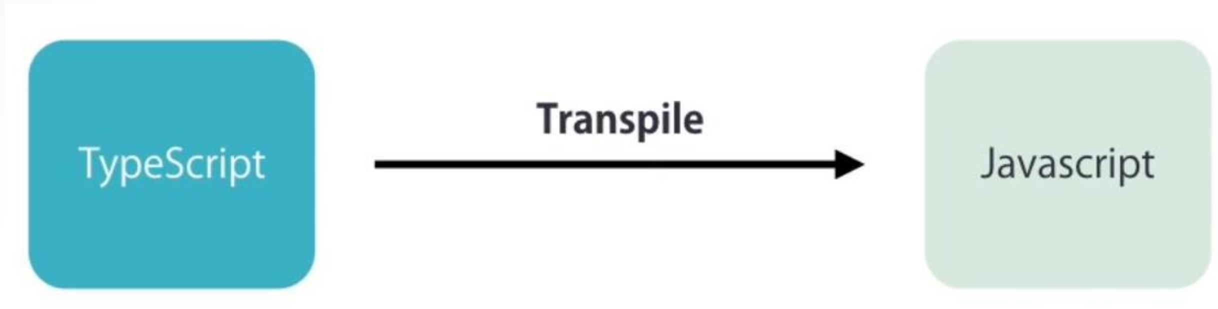
Edit – Build - Run

Codieren in TS

Compilieren (transpile) in JS ES5 oder JS ES6



```
D:\Angular\Uebungen\TsDemo>tsc --version
Version 2.0.10
```



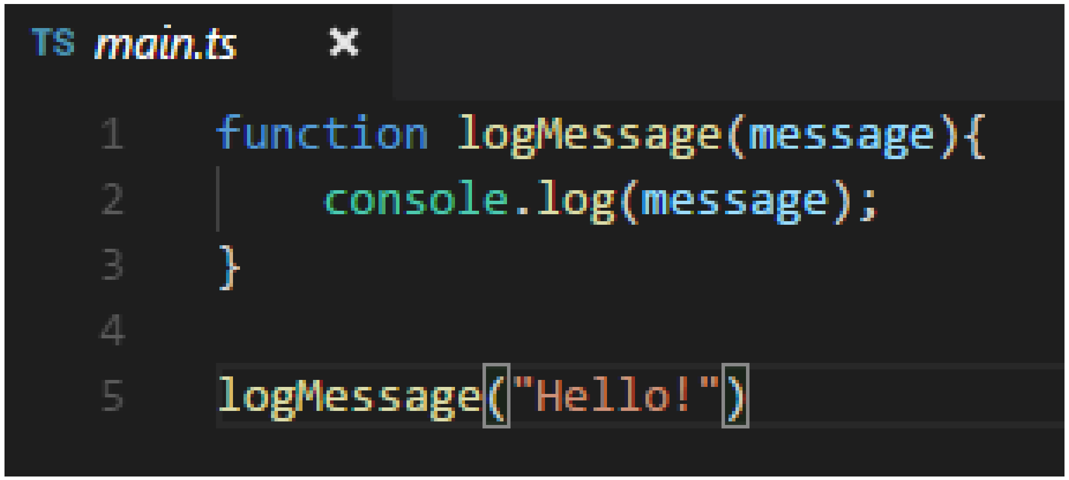
Ausführen im Browser

Debuggen auf Basis von TS

Simples TS-Program

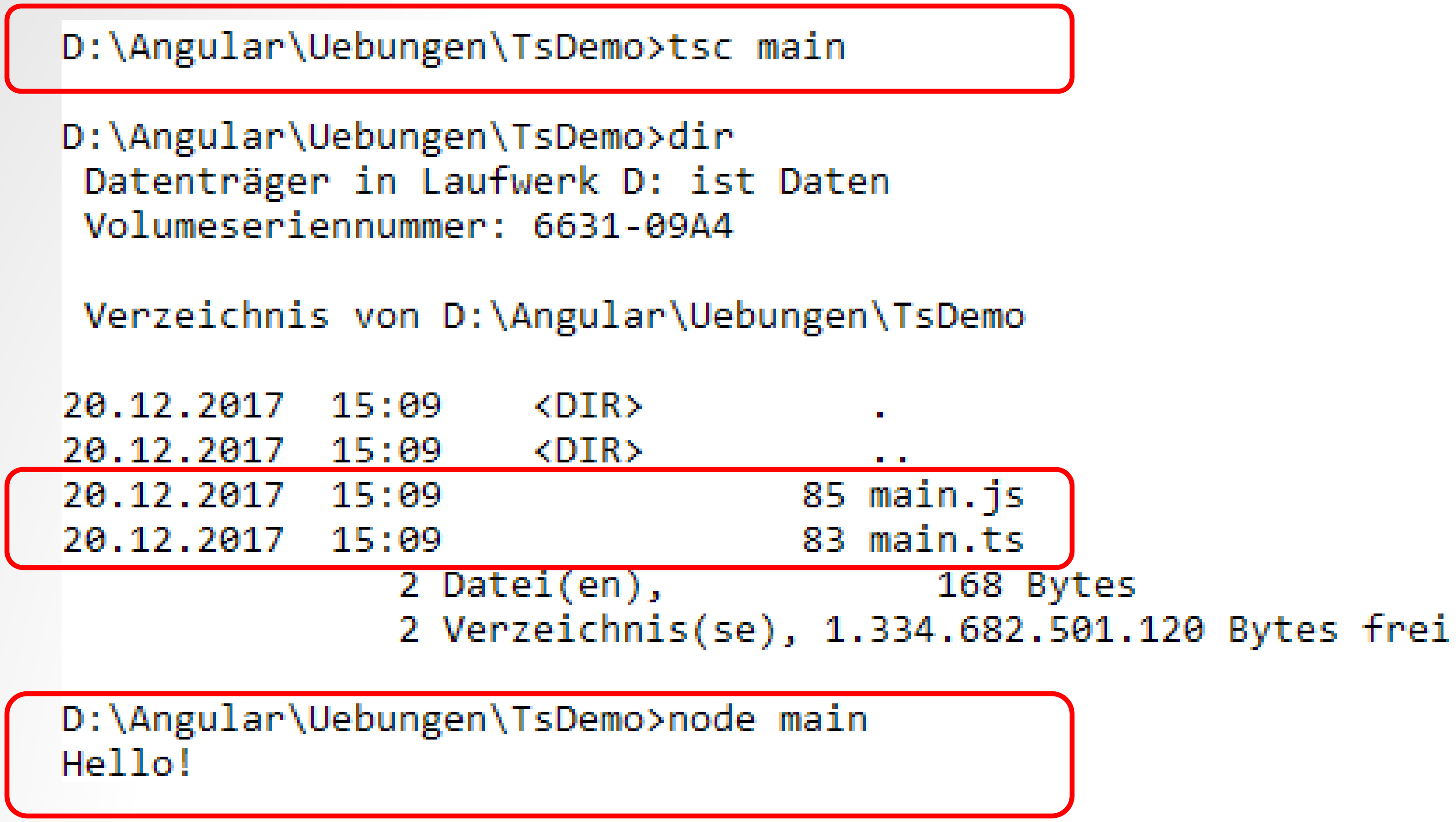
Definition einer einfachen Funktion

Aufruf im „Hauptprogramm“



```
function logMessage(message){
    console.log(message);
}

logMessage("Hello!")
```

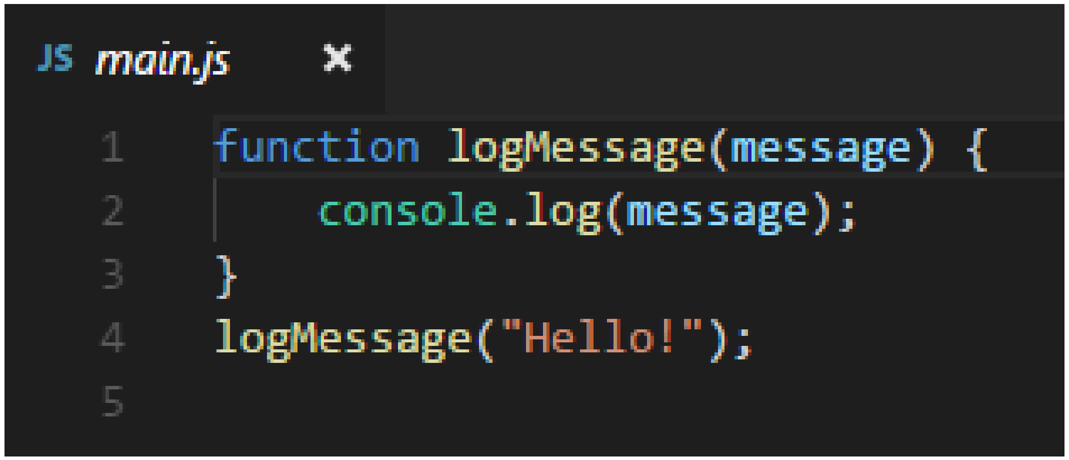
Build and run



```
D:\Angular\Uebungen\TsDemo>tsc main

D:\Angular\Uebungen\TsDemo>dir
 Datenträger in Laufwerk D: ist Daten
 Volumeseriennummer: 6631-09A4

 Verzeichnis von D:\Angular\Uebungen\TsDemo

20.12.2017  15:09    <DIR>          .
20.12.2017  15:09    <DIR>          ..
20.12.2017  15:09                85 main.js
20.12.2017  15:09                83 main.ts
               2 Datei(en),            168 Bytes
               2 Verzeichnis(se), 1.334.682.501.120 Bytes frei

D:\Angular\Uebungen\TsDemo>node main
Hello!
```

Transpiliertes js-Programm

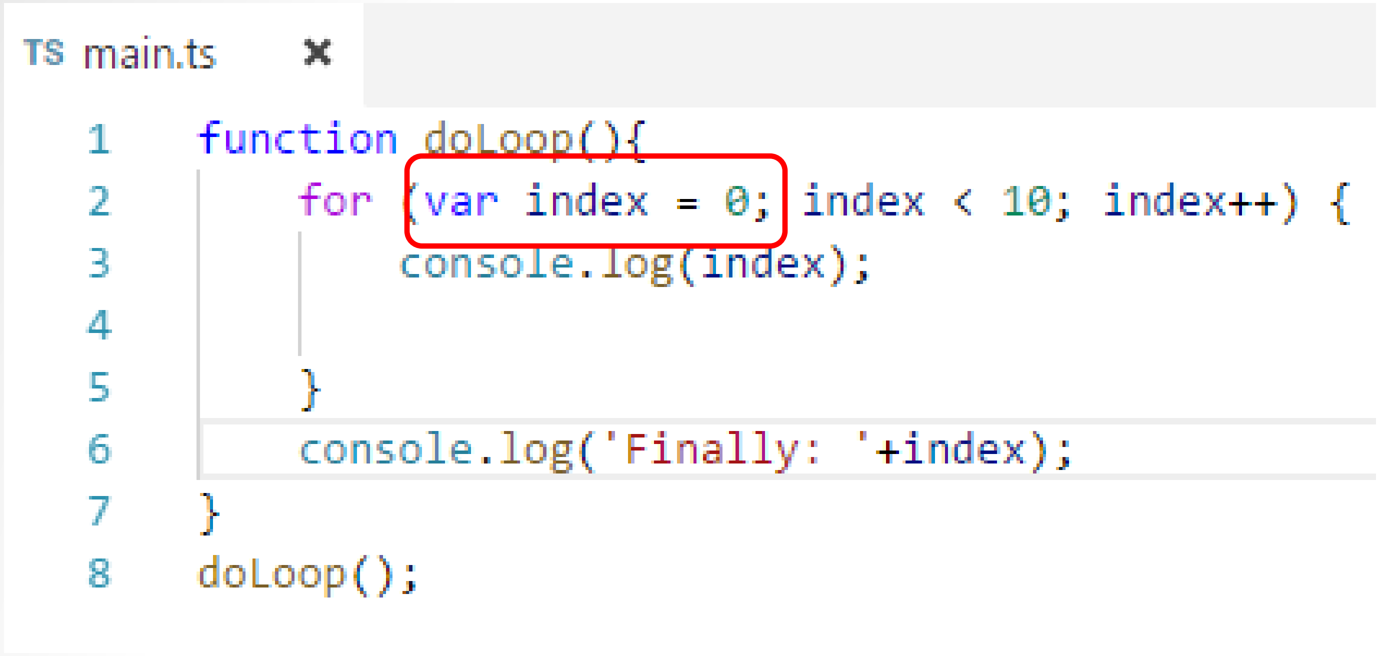
Ident ➔ gültiger JS-Code ist auch gültiger TS-Code



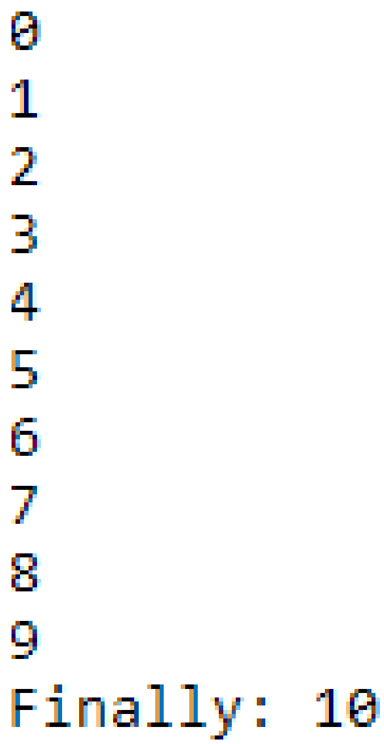
```
function logMessage(message) {
    console.log(message);
}
logMessage("Hello!");
```

Gültigkeitsbereich var

var (JS ES5) ➔ Innerhalb der Funktion



```
function doLoop(){
    for (var index = 0; index < 10; index++) {
        console.log(index);

    }
    console.log('Finally: '+index);
}
doLoop();
```



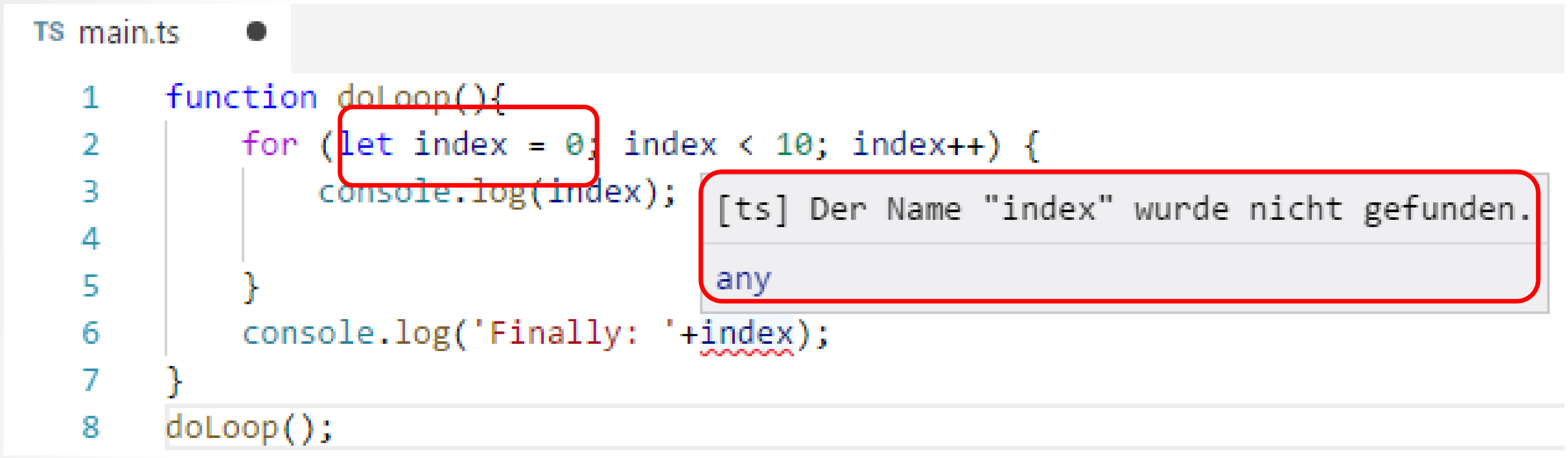
```
0
1
2
3
4
5
6
7
8
9
Finally: 10
```

Gültigkeitsbereich let

let (JS ES6) ➔ Blockorientiert

Erkennt Compilerfehler

Compiliert aber in gleiche JS-Datei wie vorher



main.ts

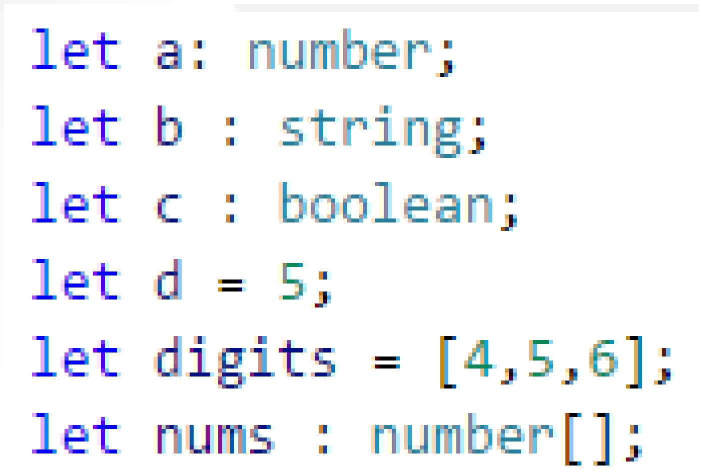
```
function doLoop(){
    for (let index = 0; index < 10; index++) {
        console.log(index);

    }
    console.log('Finally: '+index);
}
doLoop();
```

```
[ts] Der Name "index" wurde nicht gefunden.
any
```

HTL Leonding
next level

Variablendeklarationen

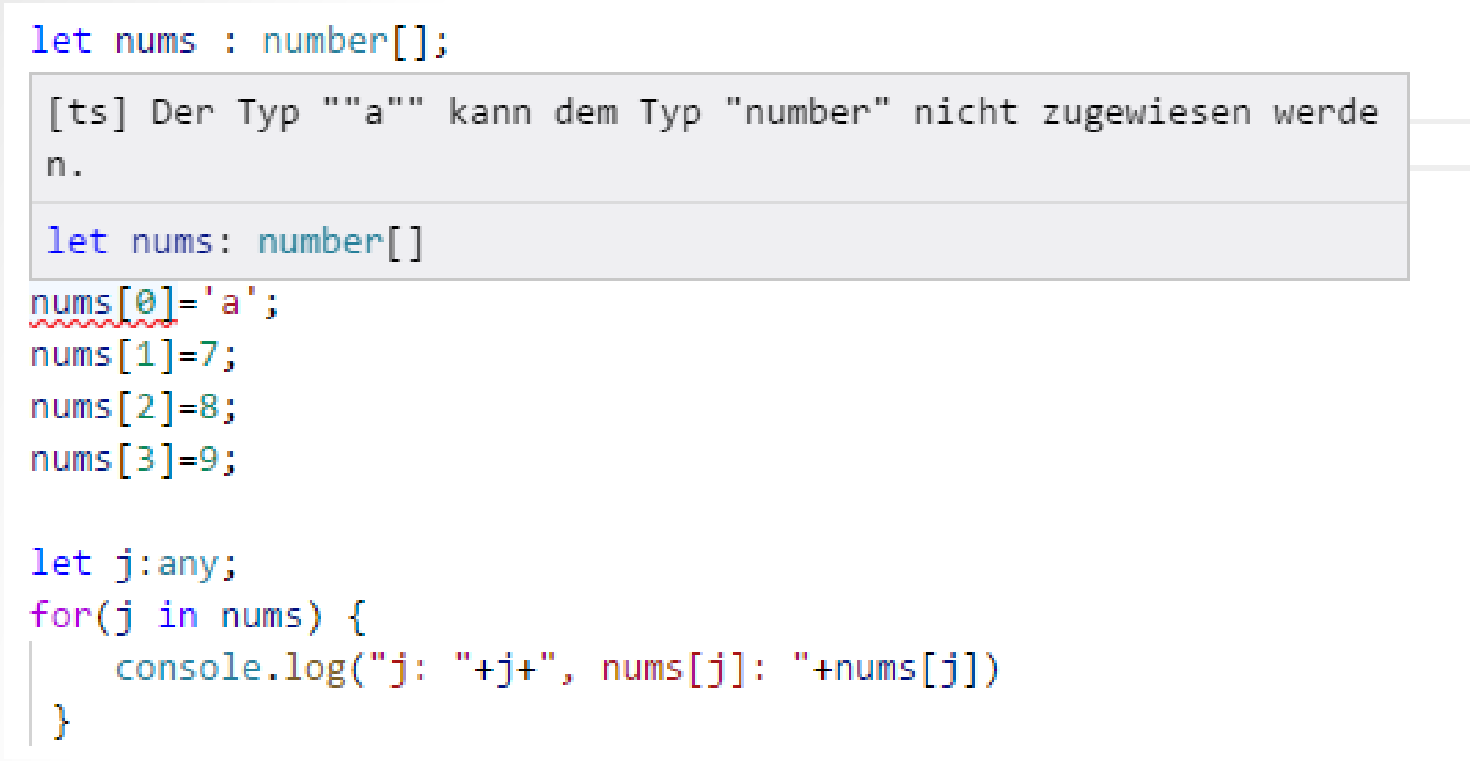
Explizite Typangabe oder Typinferenz



```
let a: number;
let b : string;
let c : boolean;
let d = 5;
let digits = [4,5,6];
let nums : number[];
```

HTL Leonding
next level

Typsicherheit

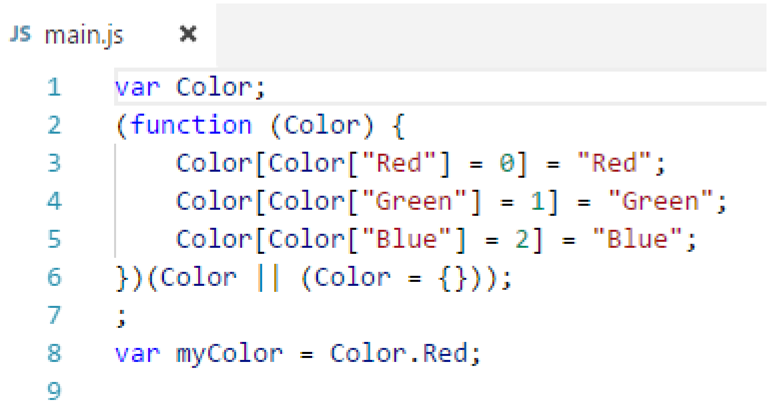


```
let nums : number[];
[ts] Der Typ ""a"" kann dem Typ "number" nicht zugewiesen werde
n.
let nums: number[]
nums[0]='a';
nums[1]=7;
nums[2]=8;
nums[3]=9;

let j:any;
for(j in nums) {
    console.log("j: "+j+", nums[j]: "+nums[j])
 }
```

Aufzählungen enum

```
enum Color { Red=0, Green=1, Blue=2};

let myColor = Color.Red;
```

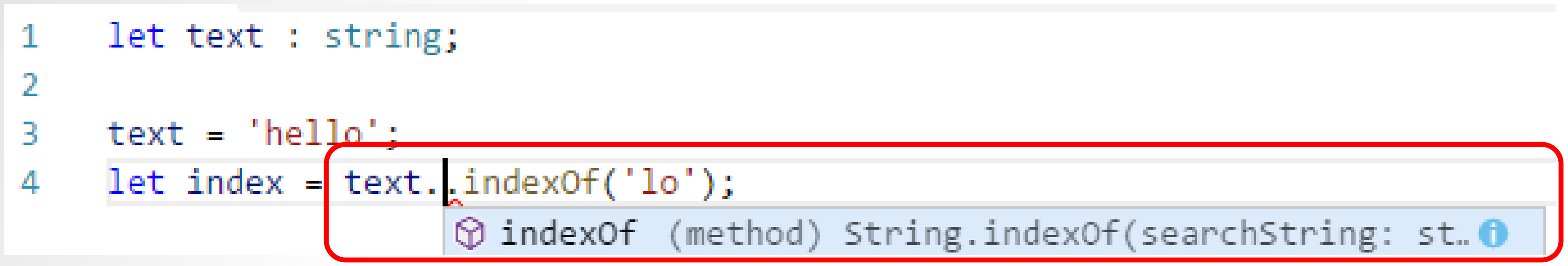
- Erzeugter JS-Code



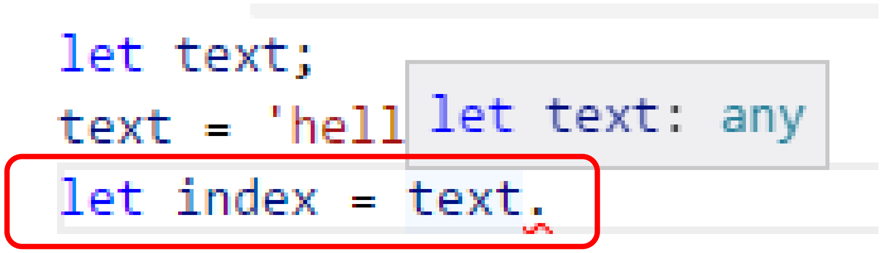
main.js

```
var Color;
(function (Color) {
    Color[Color["Red"] = 0] = "Red";
    Color[Color["Green"] = 1] = "Green";
    Color[Color["Blue"] = 2] = "Blue";
})(Color || (Color = {}));
;
var myColor = Color.Red;

```

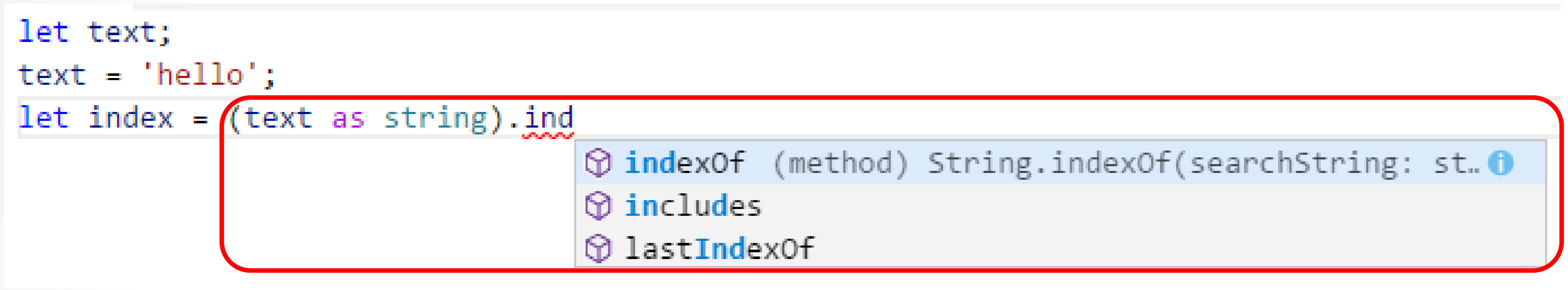
Typ herleitbar → Intellisense



```
let text : string;

text = 'hello';
let index = text..indexOf('lo');
               indexOf  (method) String.indexOf(searchString: st…
```



```
let text;
text = 'hell  let text: any
let index = text.
```



```
let text;
text = 'hello';
let index = (text as string).ind
                         indexOf  (method) String.indexOf(searchString: st…
                         includes
                         lastIndexOf
```

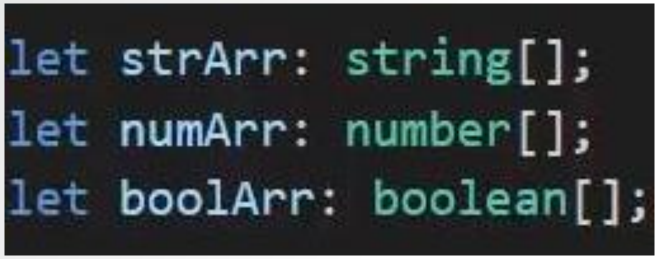
HTL Leonding
next level

Übung Types

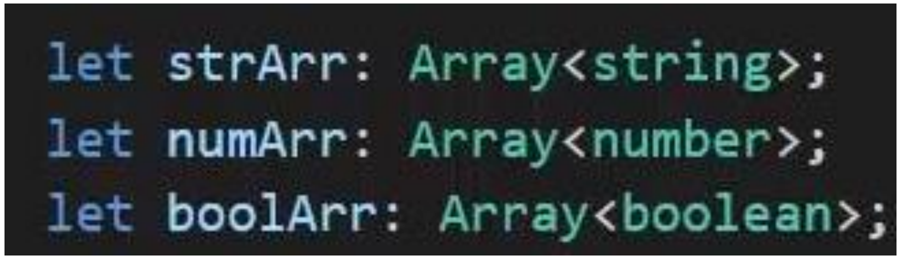
Typsicherheit ausprobieren

string, number, boolean, any

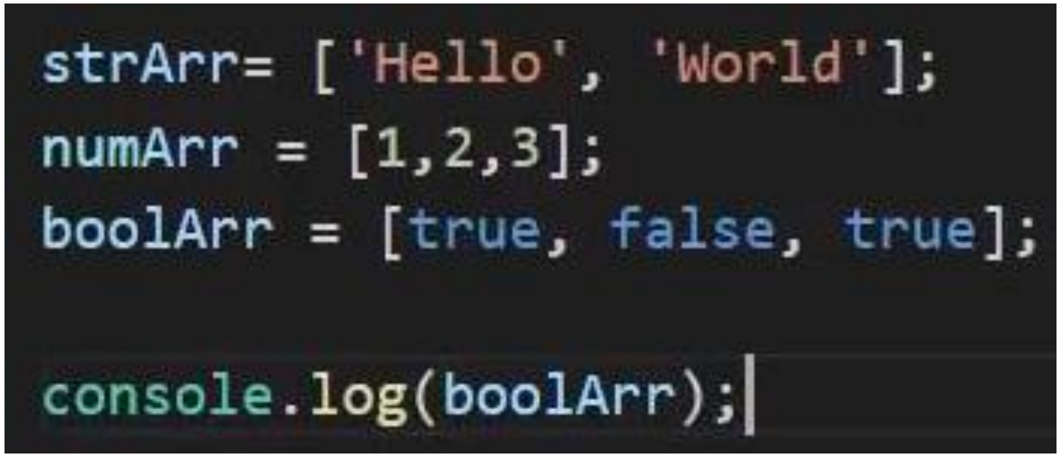
Arrays



```
let strArr: string[];
let numArr: number[];
let boolArr: boolean[];
```



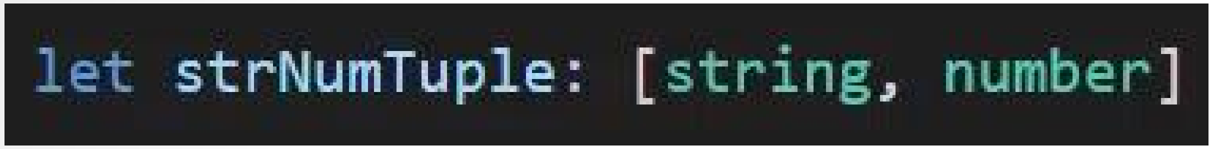
```
let strArr: Array<string>;
let numArr: Array<number>;
let boolArr: Array<boolean>;
```



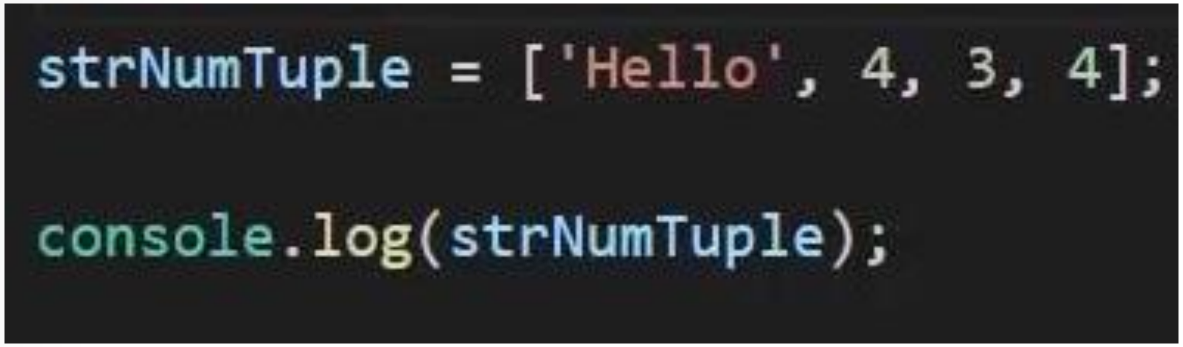
```
strArr= ['Hello', 'World'];
numArr = [1,2,3];
boolArr = [true, false, true];

console.log(boolArr);
```

HTL Leonding
next level

Tuple



```
let strNumTuple: [string, number]
```

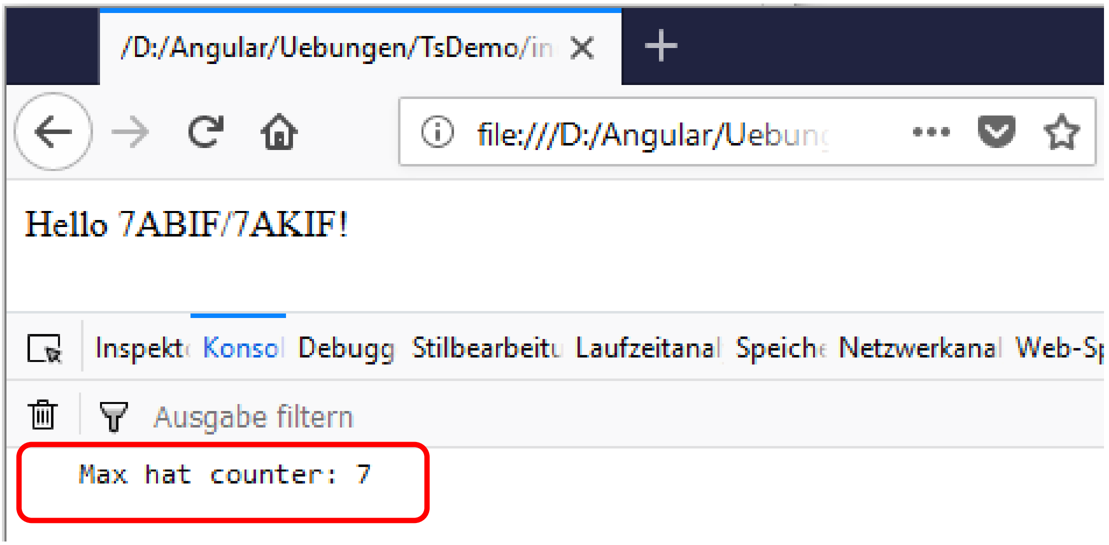


```
strNumTuple = ['Hello', 4, 3, 4];

console.log(strNumTuple);
```

HTL Leonding
next level

Beispiel PupilCounter

```
let pupilsCounter: [string, number];

pupilsCounter = ['Max', 7];
let numberOfMax = pupilsCounter[1];

console.log(pupilsCounter[0]+ ' hat counter: '+numberOfMax);
```


/D:/Angular/Uebungen/TsDemo/in
file:///D:/Angular/Uebun
Inspekt Konsol Debugg Stilbearbeit Laufzeitanal Speich Netzwerkanal Web-Sp
Ausgabe filtern

Hello 7ABIF/7AKIF!

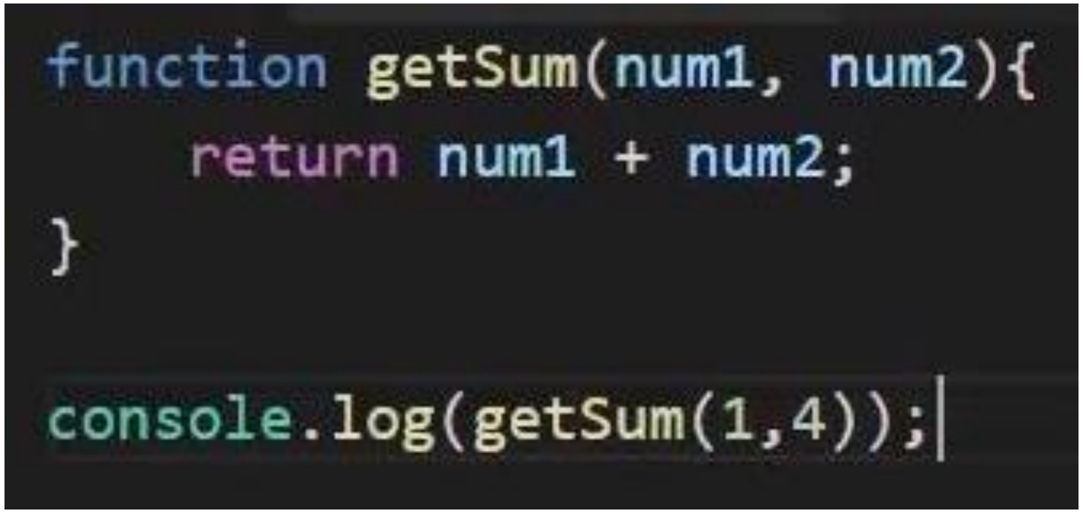
```
Max hat counter: 7
```

HTL Leonding next level

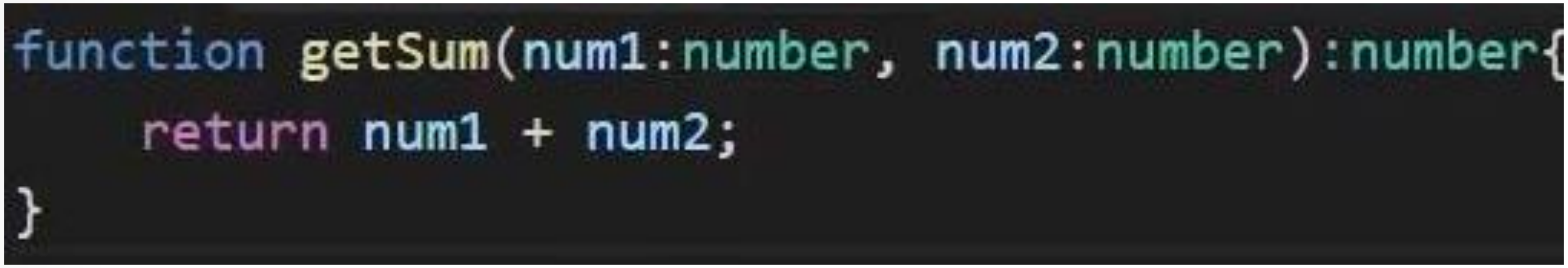
Functions

Ähnlich wie in C# und Java

Typdefinition optional



```
function getSum(num1, num2){
    return num1 + num2;
}

console.log(getSum(1,4));
```

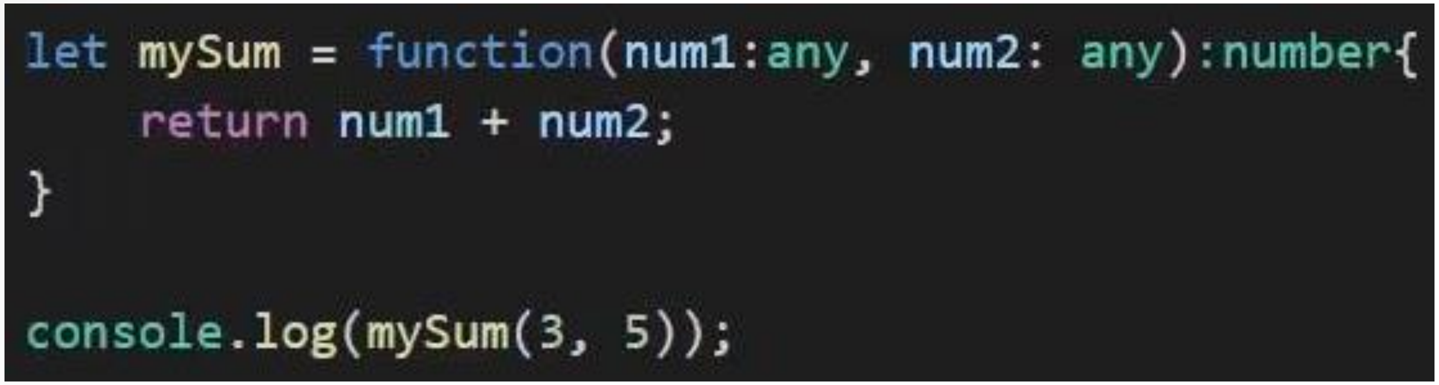


```
function getSum(num1:number, num2:number):number{
    return num1 + num2;
}
```

Funktionen sind normale Elemente

In Variablen speicherbar

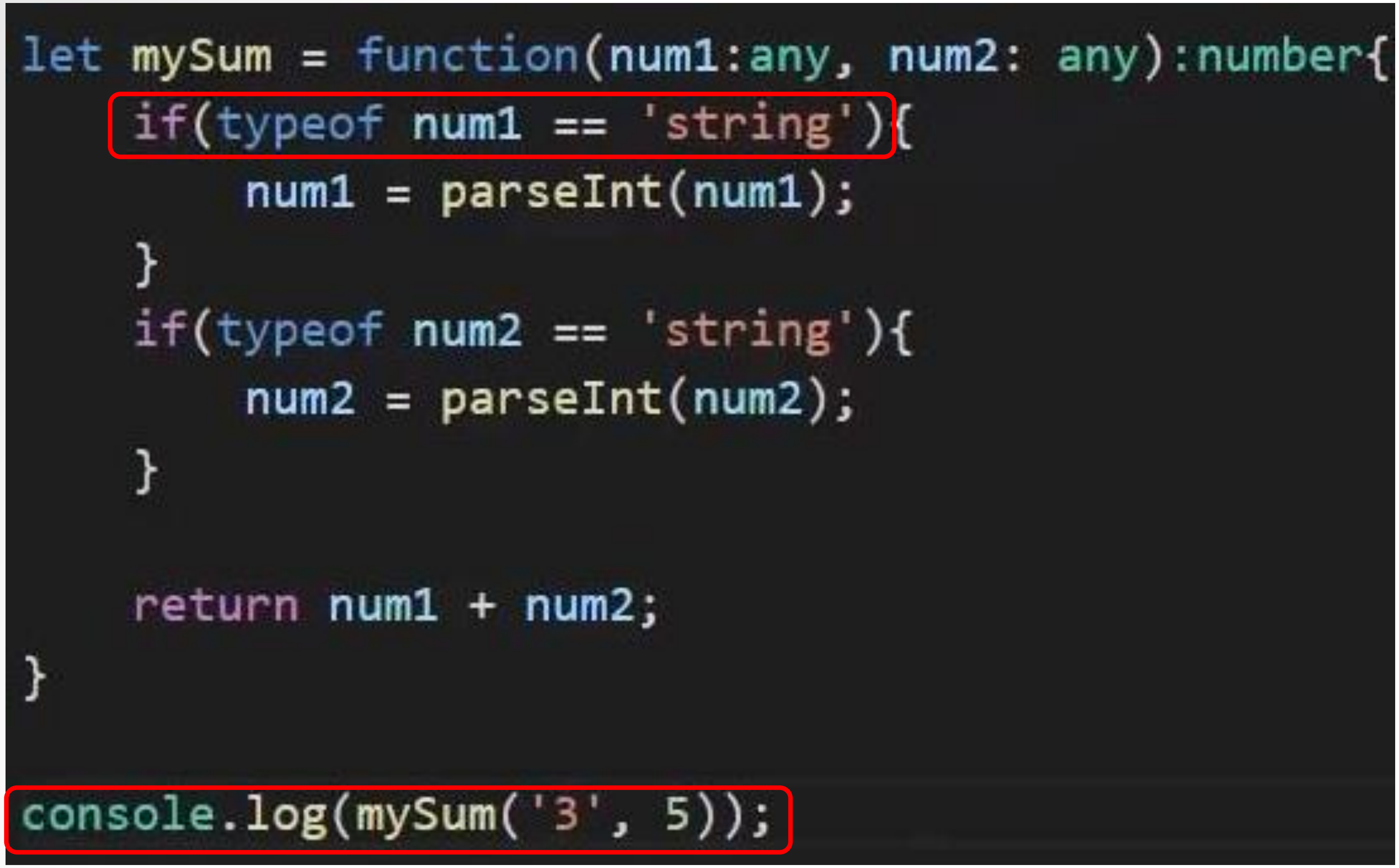
Als Parameter übergebbar



```
let mySum = function(num1:any, num2: any):number{
    return num1 + num2;
}

console.log(mySum(3, 5));
```

HTL Leonding
next level

Beispiel: Summe mit automatischem parse

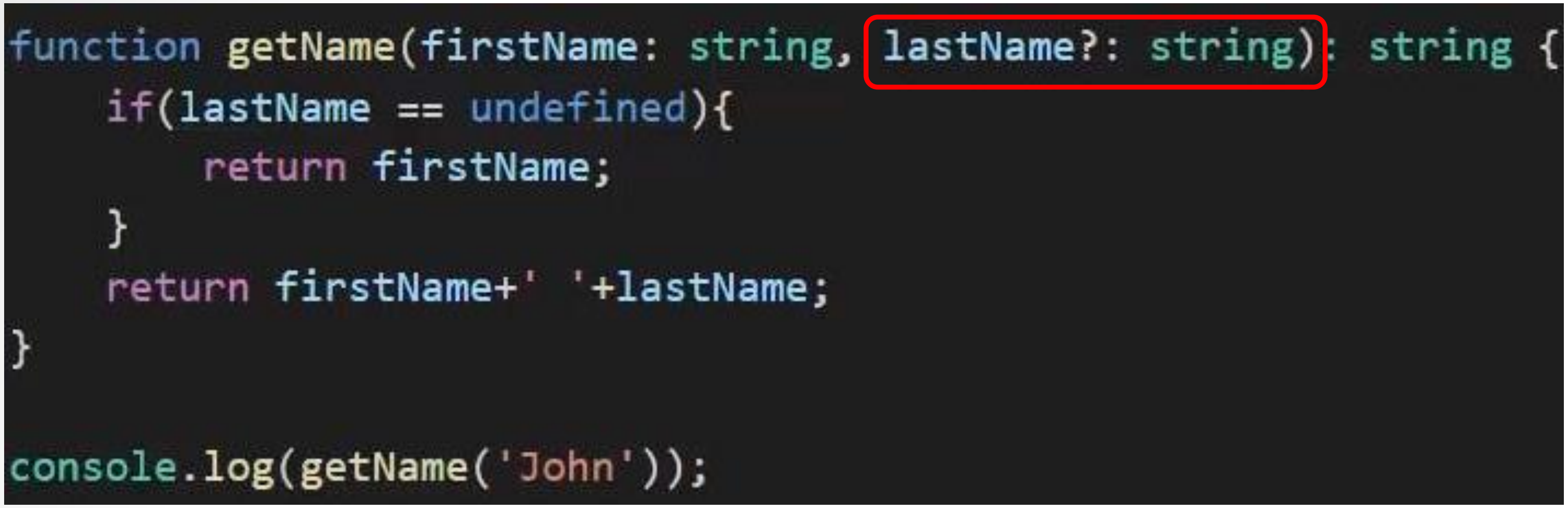


```
let mySum = function(num1:any, num2: any):number{
    if(typeof num1 == 'string'){
        num1 = parseInt(num1);
    }
    if(typeof num2 == 'string'){
        num2 = parseInt(num2);
    }

    return num1 + num2;
}

console.log(mySum('3', 5));
```

HTL Leonding
next level

Optionale Parameter

Überprüfen mit undefined



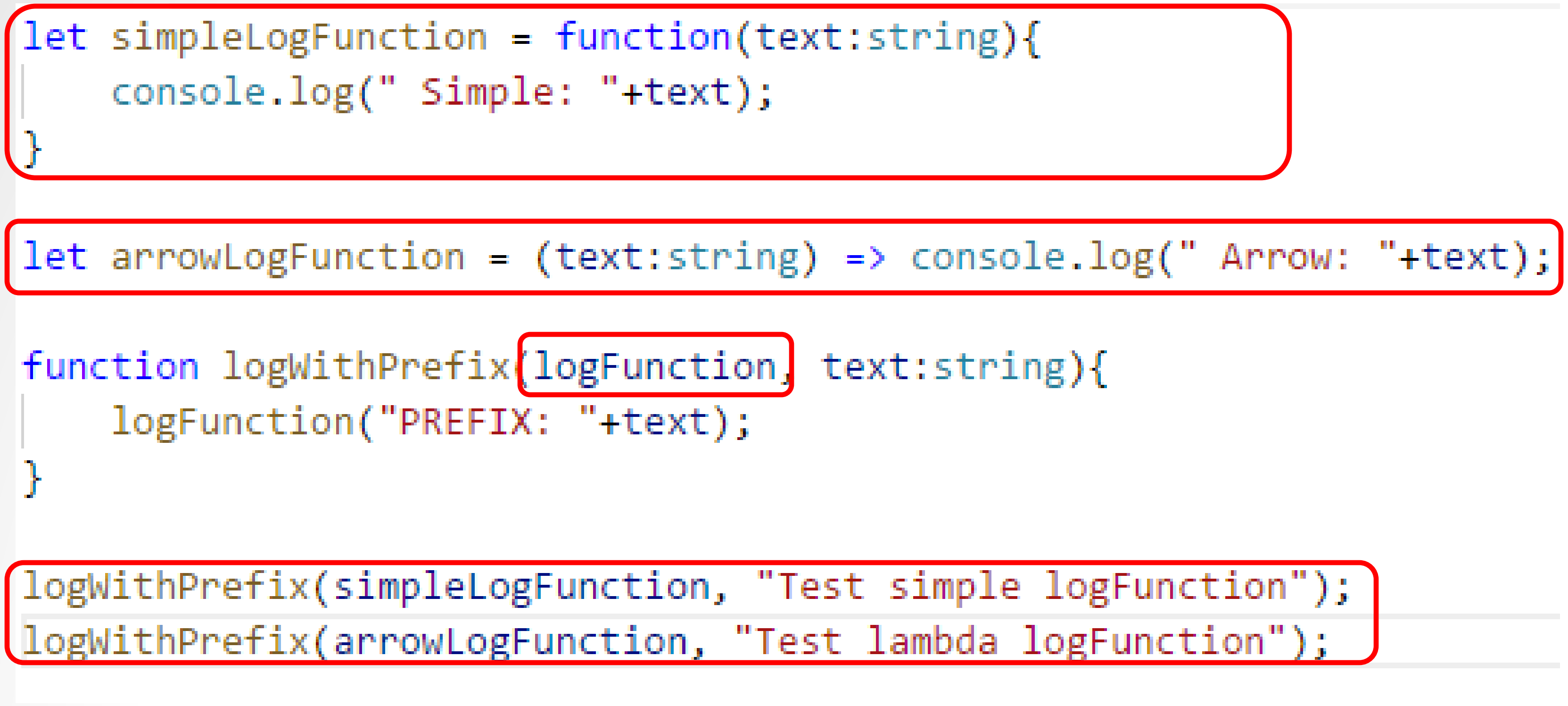
```
function getName(firstName: string, lastName?: string): string {
    if(lastName == undefined){
        return firstName;
    }
    return firstName+' '+lastName;
}

console.log(getName('John'));
```

HTL Leonding
next level

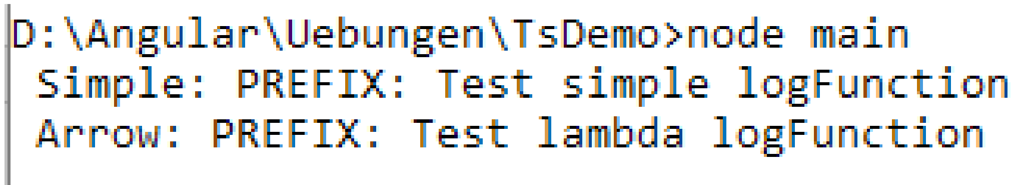
Komfort wie in C#

Funktionen als Parameter übergeben

LambdaExpressions heißen ArrowFunctions



```
let simpleLogFunction = function(text:string){
    console.log(" Simple: "+text);
}

let arrowLogFunction = (text:string) => console.log(" Arrow: "+text);

function logWithPrefix(logFunction, text:string){
    logFunction("PREFIX: "+text);
}

logWithPrefix(simpleLogFunction, "Test simple logFunction");
logWithPrefix(arrowLogFunction, "Test lambda logFunction");
```



```
D:\Angular\Uebungen\TsDemo>node main
 Simple: PREFIX: Test simple logFunction
 Arrow: PREFIX: Test lambda logFunction
```

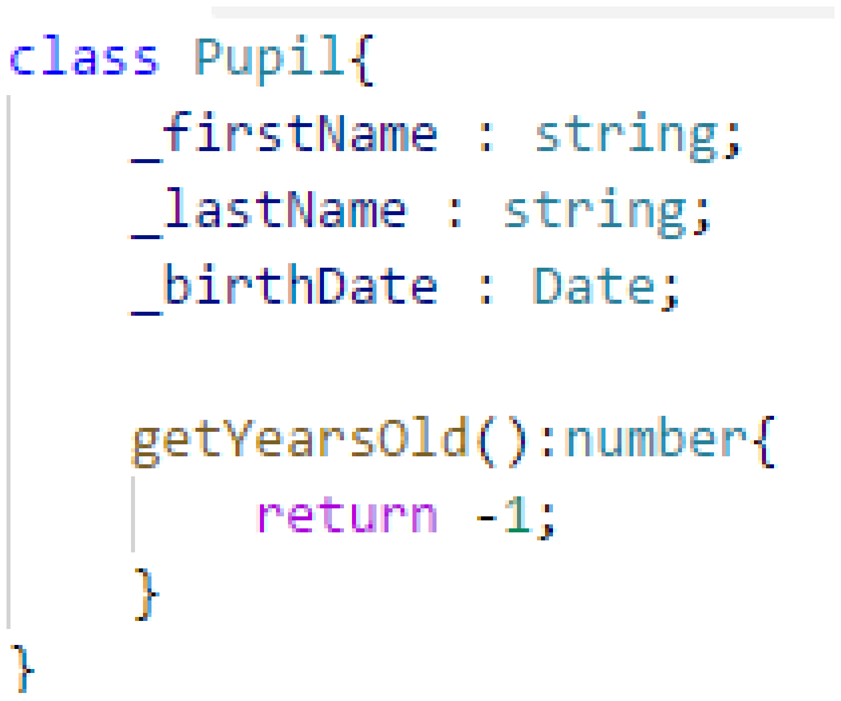
HTL Leonding
next level

Klassen

Fields

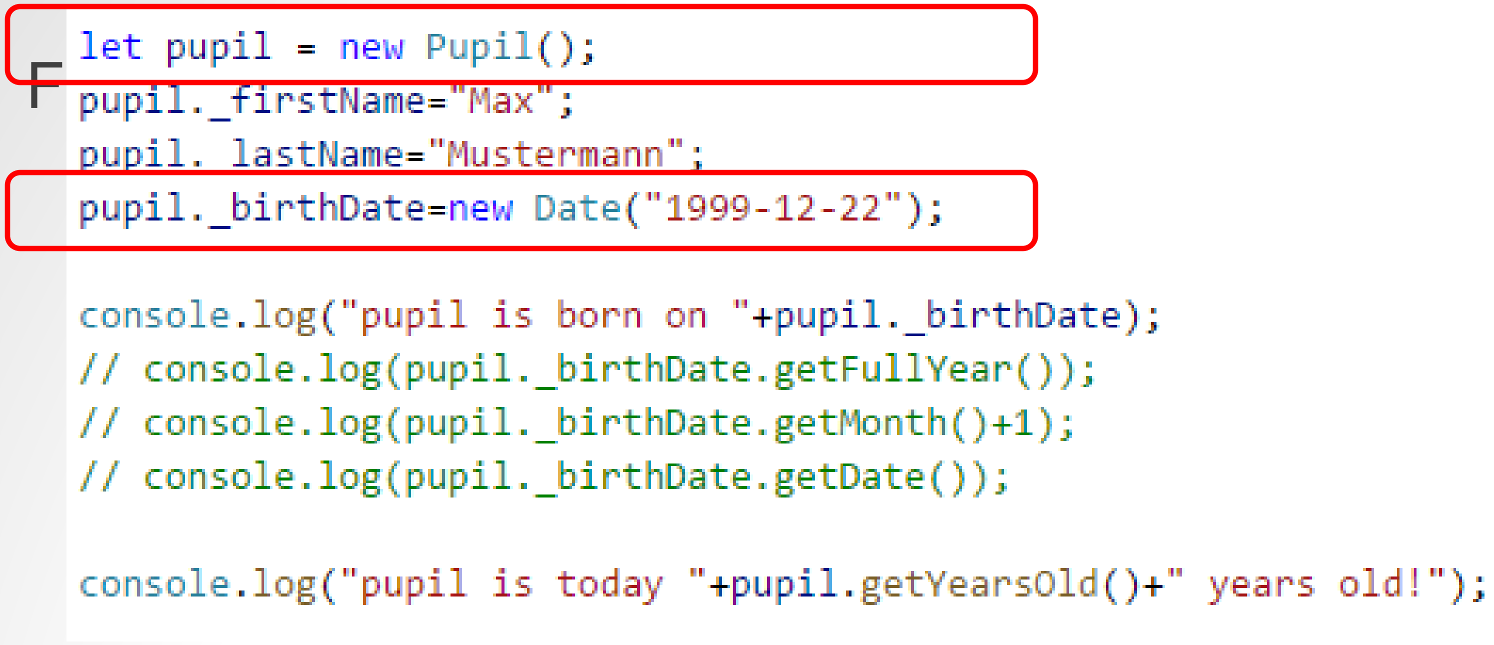
Functions in Klassen sind Meth

Standard public



```
class Pupil{
    _firstName : string;
    _lastName : string;
    _birthDate : Date;

    getYearsOld():number{
        return -1;
    }
}
```

Klassen - Verwendung

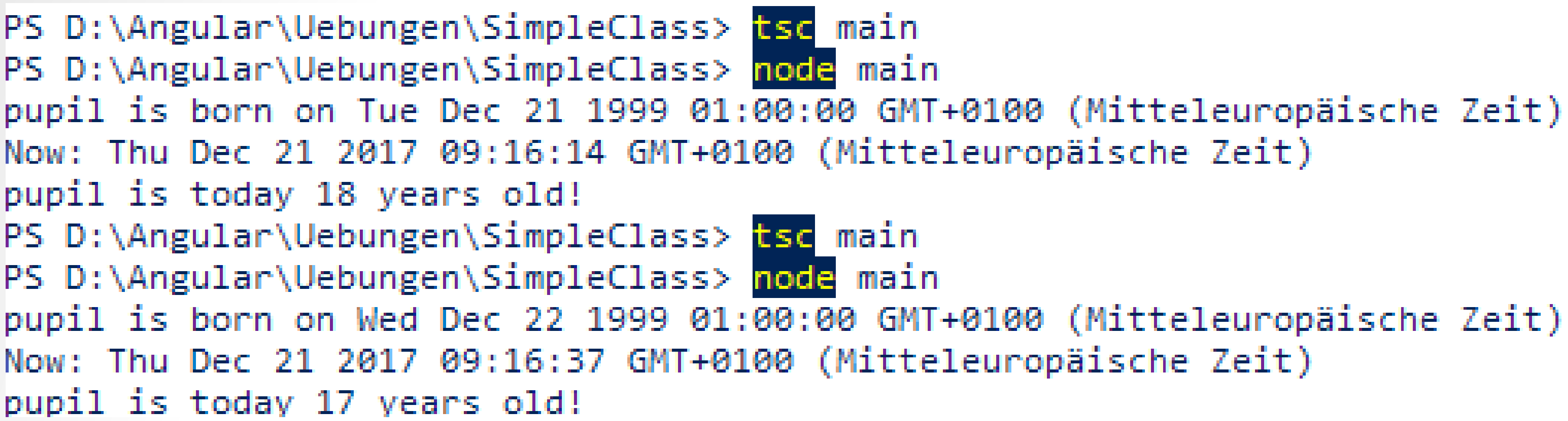


```
let pupil = new Pupil();
pupil._firstName="Max";
pupil._lastName="Mustermann";
pupil._birthDate=new Date("1999-12-22");

console.log("pupil is born on "+pupil._birthDate);
// console.log(pupil._birthDate.getFullYear());
// console.log(pupil._birthDate.getMonth()+1);
// console.log(pupil._birthDate.getDate());

console.log("pupil is today "+pupil.getYearsOld()+" years old!");
```

HTL Leonding
next level

Wie viele Jahre ist der Schüler alt?

Gewünschte Ausgabe des Programms

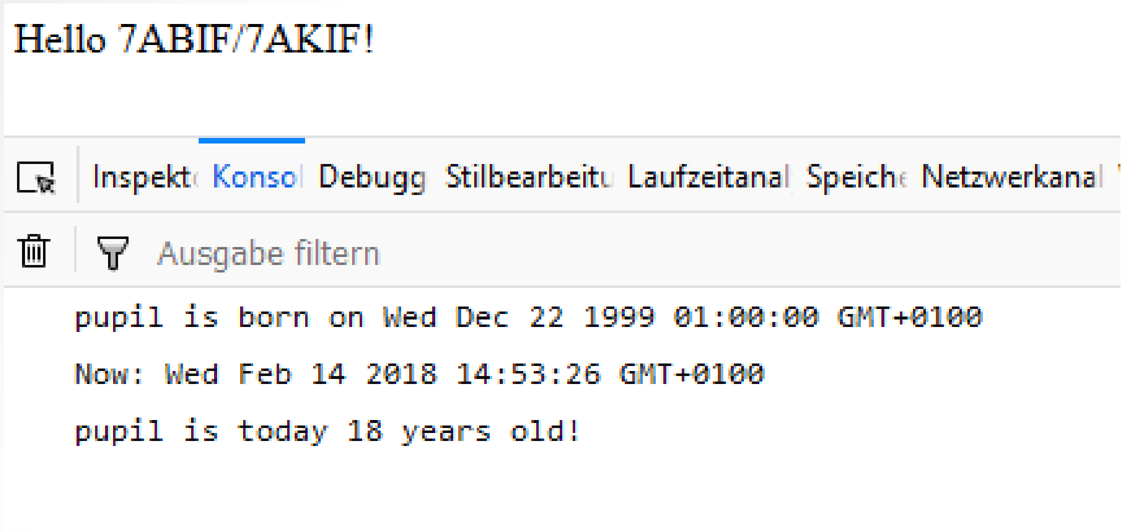


```
PS D:\Angular\Uebungen\SimpleClass> tsc main
PS D:\Angular\Uebungen\SimpleClass> node main
pupil is born on Tue Dec 21 1999 01:00:00 GMT+0100 (Mitteleuropäische Zeit)
Now: Thu Dec 21 2017 09:16:14 GMT+0100 (Mitteleuropäische Zeit)
pupil is today 18 years old!
PS D:\Angular\Uebungen\SimpleClass> tsc main
PS D:\Angular\Uebungen\SimpleClass> node main
pupil is born on Wed Dec 22 1999 01:00:00 GMT+0100 (Mitteleuropäische Zeit)
Now: Thu Dec 21 2017 09:16:37 GMT+0100 (Mitteleuropäische Zeit)
pupil is today 17 years old!
```

Übung

HTL Leonding
next level

Alternativ im Browser

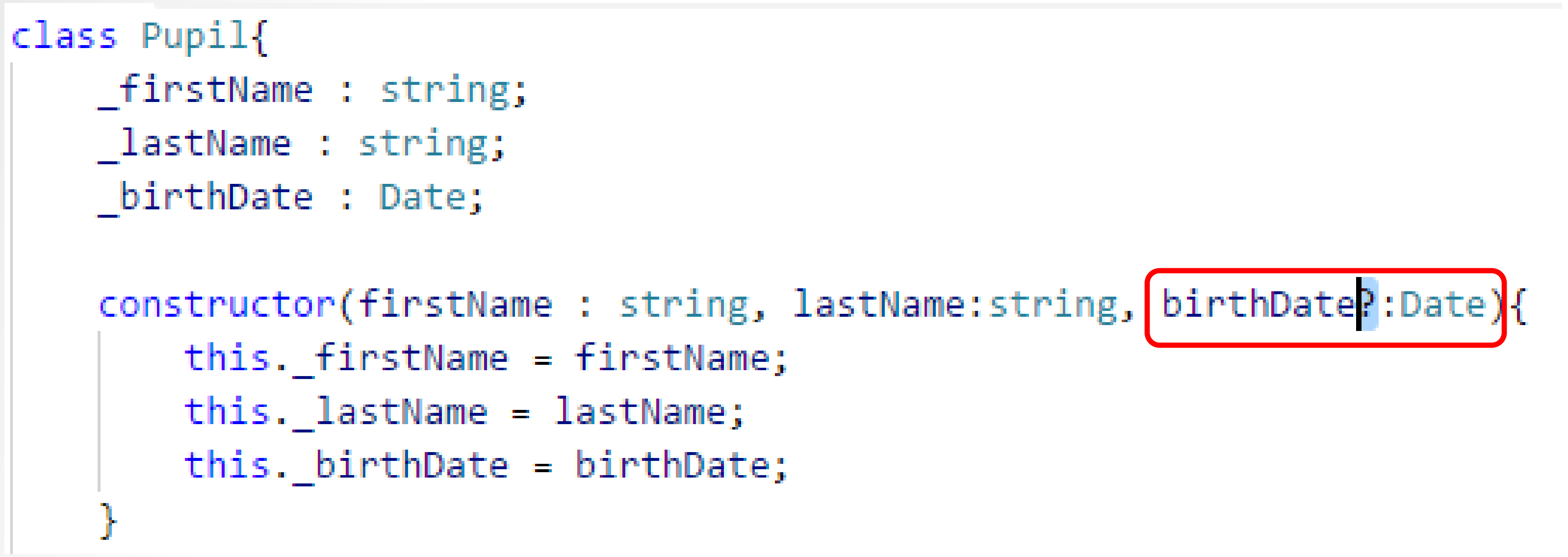

Hello 7ABIF/7AKIF!

```
pupil is born on Wed Dec 22 1999 01:00:00 GMT+0100
Now: Wed Feb 14 2018 14:53:26 GMT+0100
pupil is today 18 years old!
```

Constructor, optionale Parameter

Constructor wir in C#/Java

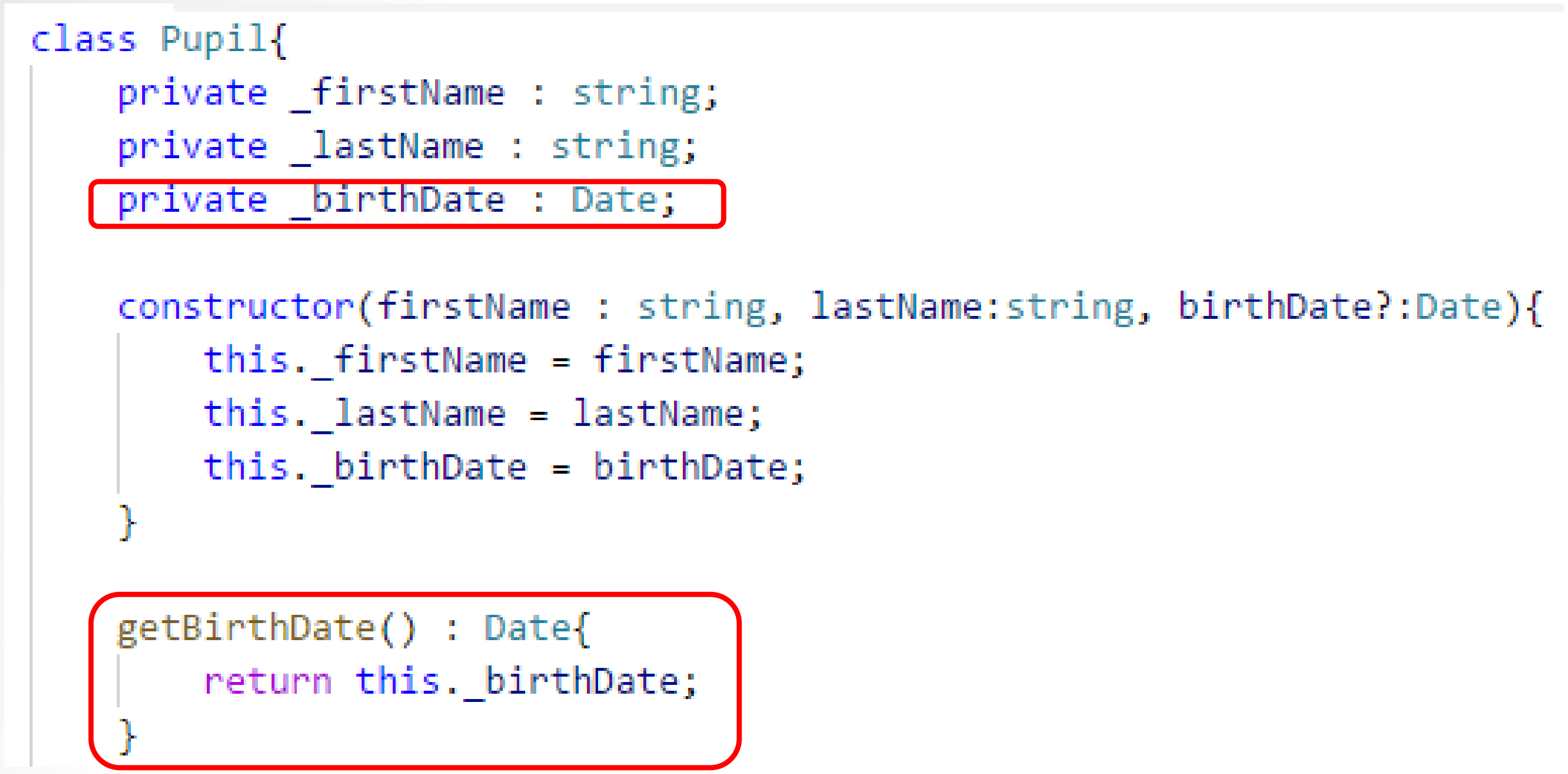
Optionale Parameter am Schluss der Parameterliste



```
class Pupil{
    _firstName : string;
    _lastName : string;
    _birthDate : Date;

    constructor(firstName : string, lastName:string, birthDate?:Date){
        this._firstName = firstName;
        this._lastName = lastName;
        this._birthDate = birthDate;
    }
```

Kapselung der fields

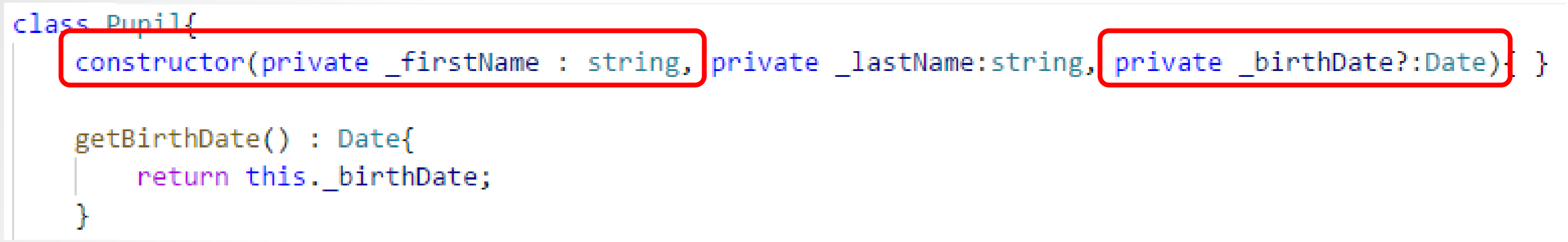
Get/Set-Methode wie in Java



```
class Pupil{
    private _firstName : string;
    private _lastName : string;
    private _birthDate : Date;

    constructor(firstName : string, lastName:string, birthDate?:Date){
        this._firstName = firstName;
        this._lastName = lastName;
        this._birthDate = birthDate;
    }

    getBirthDate() : Date{
        return this._birthDate;
    }
```

Definition der fields im Constructor

Sehr kompakter Code



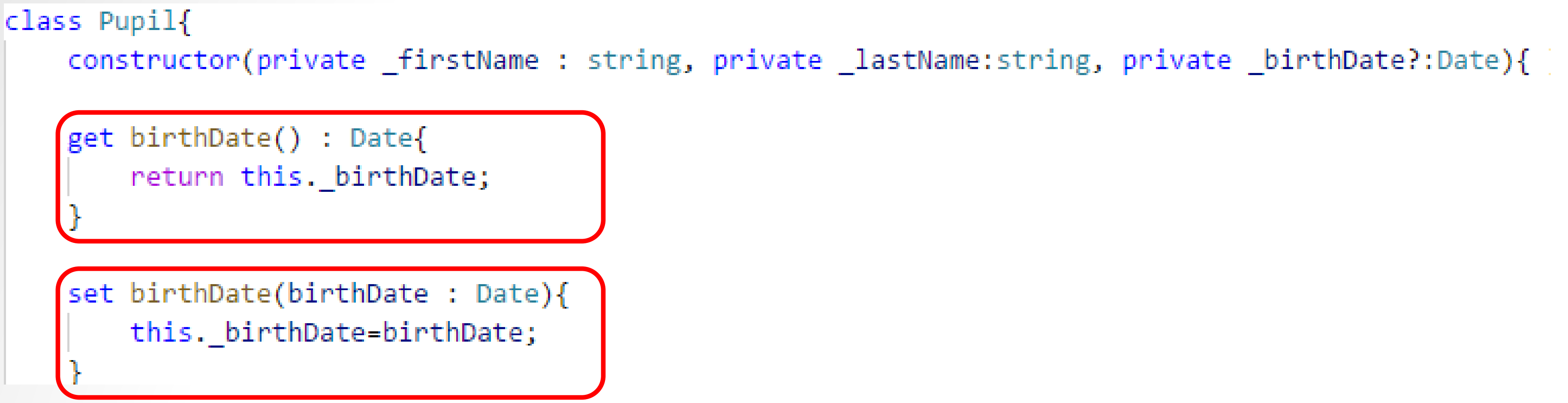
```
class Pupil{
    constructor(private _firstName : string, private _lastName:string, private _birthDate?:Date){ }

    getBirthDate() : Date{
        return this._birthDate;
    }
```

HTL Leonding
next level

Properties in TS

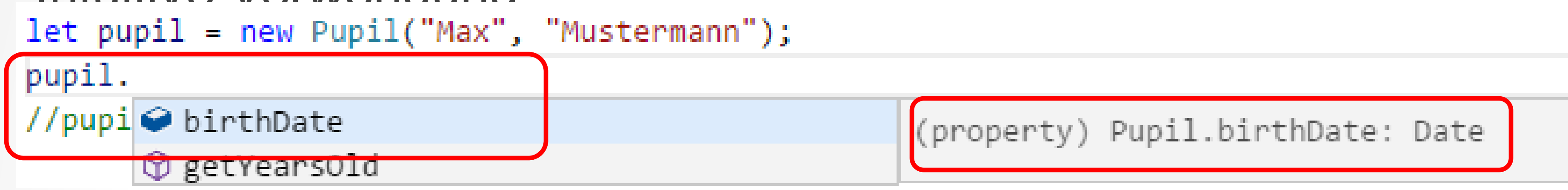
Kapselung in Getter/Setter wie in C#

Validierung, Converter, …



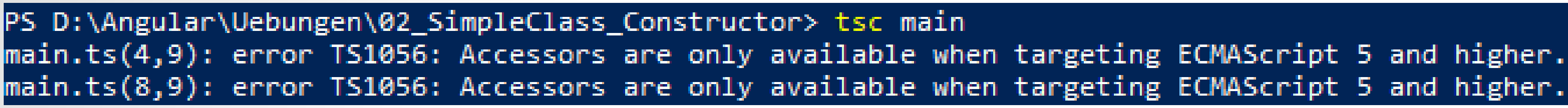
```
class Pupil{
    constructor(private _firstName : string, private _lastName:string, private _birthDate?:Date){

    get birthDate() : Date{
        return this._birthDate;
    }

    set birthDate(birthDate : Date){
        this._birthDate=birthDate;
    }
```

Intuitive Verwendung



```
let pupil = new Pupil("Max", "Mustermann");
pupil.
//pupi
```

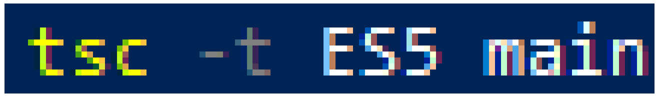
Compiler liefert Fehlermeldung



```
PS D:\Angular\Uebungen\02_SimpleClass_Constructor> tsc main
main.ts(4,9): error TS1056: Accessors are only available when targeting ECMAScript 5 and higher.
main.ts(8,9): error TS1056: Accessors are only available when targeting ECMAScript 5 and higher.
```

tsc compiliert per default gegen ES3

Compiler konfigurieren (Target ES5)

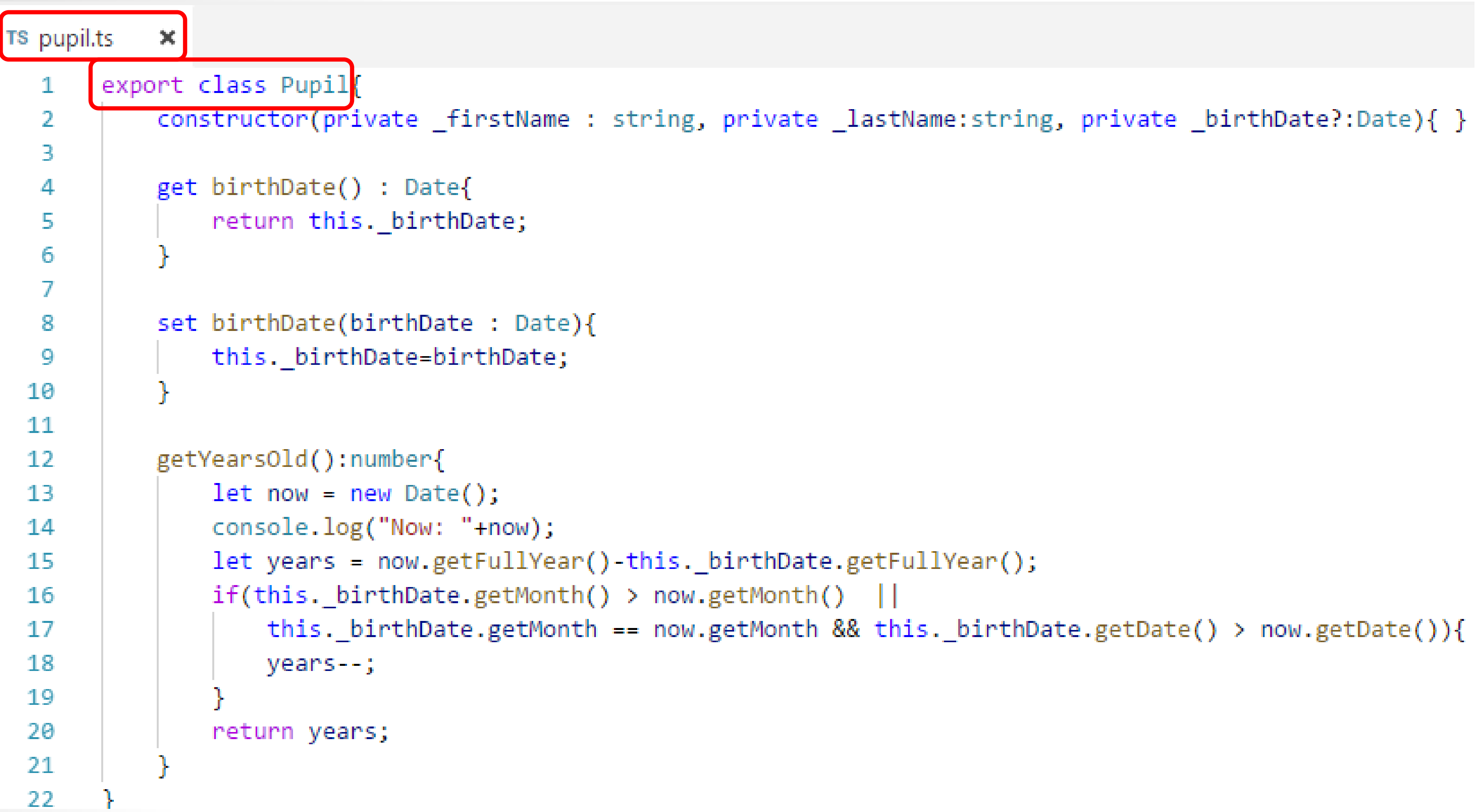


```
tsc -t ES5 main
```

HTL Leonding
next level

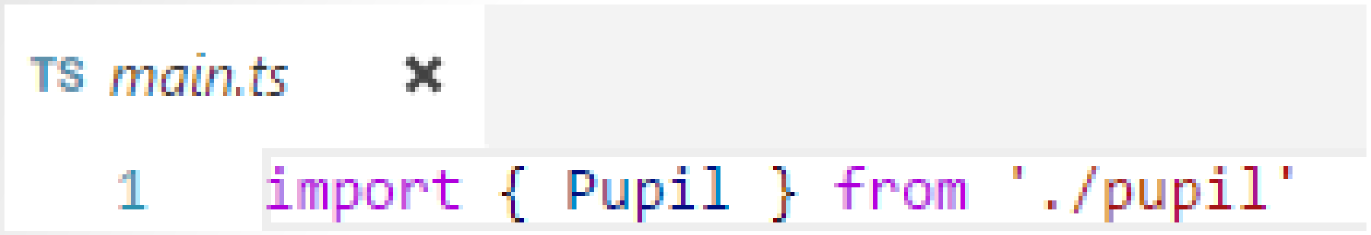
TS-Modul (nicht Angular Module)

Klasse in eigene Datei und „export“


TS pupil.ts ✖

```
export class Pupil{
    constructor(private _firstName : string, private _lastName:string, private _birthDate?:Date){ }

    get birthDate() : Date{
        return this._birthDate;
    }

    set birthDate(birthDate : Date){
        this._birthDate=birthDate;
    }

    getYearsOld():number{
        let now = new Date();
        console.log("Now: "+now);
        let years = now.getFullYear()-this._birthDate.getFullYear();
        if(this._birthDate.getMonth() > now.getMonth()  ||
            this._birthDate.getMonth == now.getMonth && this._birthDate.getDate() > now.getDate()){
            years--;
        }
        return years;
    }
}
```

Verwendung über import


TS main.ts

```
import { Pupil } from './pupil'
```

Vererbung ist natürlich auch möglich

```
class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}

class Horse extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 45) {
        console.log("Galloping...");
        super.move(distanceInMeters);
    }
}

let sam = new Snake("Sammy the Python");
let tom: Animal = new Horse("Tommy the Palomino");

sam.move();
tom.move(34);
```