TypeScript

Angular

HOGESCHOOL PXL

Introductie TypeScript

- Ontwikkeld door Microsoft
- Javascript-achtige syntax
- Voorzien van volledig type support
- Voorzien van volledig OOP support
- Compiled naar JavaScript

=> Wordt gebruikt doorheen Angular vanaf Angular 2

TypeScript Syntax

- Uitbreiding op bestaande JavaScript
- Datatypes en object ondersteuning toegevoegd
- Eenvoudig om te leren
- Kan legacy JavaScript code bevatten
- Kan gebruikt worden buiten Angular

Definitie van variabelen

- Standaard Javascript ondersteunt maar één variabele type:

```
let x = ‘blabla’; // String data
let y = false     // boolean data
let z = 33    // numeric data
let a = ‘whatever’;   // any data
```

- TypeScript ondersteunt verschillende variabele types:

```
let x: string = ‘blabla’; // String data
let y: boolean = false    // boolean data
let z: number = 33    // numeric data
let a: any = ‘whatever’;  // any data
```

Definitie van arrays

- Standaard in JavaScript:

```
let kleuren = [‘rood’,’blauw’,’zwart’,’groen’];
let getallen = {1,2,3,4,20};
let personen = [{naam: ‘Dries’},{naam:’Bob’}];
```

- In TypeScript

```
let kleuren: string[] = [‘rood’,’blauw’,’zwart’,’groen’];
let getallen: number[] = {1,2,3,4,20};
let personen: Object[] = [{naam: ‘Dries’},{naam:’Bob’}];
```

Definitie van arrays

- Datatypes bij arrays zorgen ervoor dat enkel objecten van dat datatype toegevoegd kunnen worden:
  - Enkel getallen kunnen toegevoegd worden:

```
let getallen: number[] = {10,20,30}
```

  - Verschillende types van objecten kunnen toegevoegd worden:

```
let namen: Object[];
```

  - Enkel Objecten van het type Person kunnen toegevoegd worden:

```
let people: Person[];
```

Classes en Objecten

- Definitie van een klasse met properties in TypeScript:

```
class Cat{
    naam:string;
    type:string;
}
```

- Aanmaken van objecten van het type Cat:

```
let cats: Cat[];
let cat1: Cat = {name: “Luna”, type:”European Shorthair”};
let cat2: Cat = {name: “Kiara”, type:”Maine Coon”};

cats[0] = cat1;
cats[1] = cat2;
```

Classes en Objecten

- Gebruik van constructors:

```
class Cat{
    naam:string;
    type:string;

    constructor(naam: string, type:string){
        this.naam = naam;
        this.type = type;
    }
}

let cat1 = new Cat("Luna", "European Shorthair");
let cat2 = new Cat("Kiara", "Maine Coon");

let cats: Cat[] = [cat1, cat2];
```

Constructors alternatief

- Argumenten in de constructor worden automatisch toegevoegd als klasse properties:
  - Scope moet meegegeven worden (private / public)
  - Wordt vaak gebruikt bij dependency injection (zie later)

```
class Building{
    constructor(private address: string, private units:
number){}
}

let bld1 = new Building("1 main street", 4);
```

Interfaces

- Interfaces kunnen gebruikt worden als “template” voor een klasse.
- Eén van de meest gebruikte interfaces bij Angular is “OnInit”

```
interface Itrip{
    destination: string;
    days: number;
    display();
}
class BizTrip implements Itrip{
    constructor(private destination: string, private   days:
number){}

    display(){
       console.log(this.destination + “, “ + this.days);
    }
}
```

Functies & methodes

- Bij functies met een return value wordt het datatype meegegeven in de declaratie van de functie:

```
function getString(): string {
    let str: string = “My string”;
    return str
}
```

- Meegeven van argumenten binnen een functie zonder return value:

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

logMessage(getString());
```

Functies & methodes

- Het gebruik van een array als parameter van een functie:

```
let strings:string[] = [“abc”,”def”,”ghi”];

function displayStr(string_array: string[]){
    for(let idx in string_array){
        console.log(string_array[idx]);
    }
}

displayStr(strings);
```

Functies en methodes

- Bij methodes binnen een klasse gebruik je **geen** keyword “function”:

```
class BizTrip implements Itrip{
    constructor(private destination: string, private   days:
number){}

    display(){
       console.log(this.destination + “, “ + this.days);
    }
}
```

Werken met modules

- Voor het gebruik van verschillende .ts bestanden moeten classes geëxporteerd worden om ze te kunnen gebruiken.
- Bijvoorbeeld:
  - person.ts = Bestand met de klasse Person
  - app.ts = de logica van een applicatie waarin de klasse Person gebruikt moet worden.

Werken met modules

- Gebruik van het keyword **export** in person.ts
- **Import** toevoegen aan app.ts

```
///person.ts
export class Person {
    constructor(private fname, private lname){}
    display(){console.log(fname + " " + lname);
}

//app.ts
import {Person} from './person';

let p1 = new Person("John", "Doe");
p1.display();
```

Var, let & const

- const variabelen:
  - Variabelen waarvan de waarden slechts één maal gedefinieerd wordt
  - Best practise : uppercase declaratie

```
for(let i=0; i < 5;i++){
    ...
}

if(a === b){
    let x = 5;
}

const PI: number = 3.14159;
const URL: string = “http://pxl.be/products”;
```

Arrow functies

- “Arrow functies” wordt gebruikt als verkorte manier voor het definiëren van functies
- Linkse gedeelte van de pijl = de parameters van de functie
- Rechtse gedeelte van de pijl: implementatie van de functie
- Geen return keyword nodig

```
let adder1 = function(a,b){ return a + b; } //klassieke methode
let adder2 = (a,b)=>a+b; //arrow functie

console.log(adder1(1,2));
console.log(adder2(1,2));

let data1 = function(result) { return result.data; }
let data2 = result=>result.data;
```

Template strings

- Nieuwe syntax beschikbaar in ES6 & TypeScript
- Gebruik van back-ticks (`) om string literals te definiëren
- Expressies en variabelen inline vervangen
- Multiline ondersteuning

```
let name = `Dries`
let x = `Hallo ${name}`;

let y = `${name} heeft ${2 * 3} euro
     gewonnen!`;

function getName(){return “Joske”;}
let z = `Goedemorgen ${getName()}`;
```