



## JS Advanced & Typescript

### FullStack VDO

<b>Wat is Typescript?</b>	<b>1</b>
JS vs TS	1
Onze webbrowser werkt alleen met HTML, CSS en JavaScript	2
Het toevoegen van types aan attributen en methodes	5
<b>TypeScript basics</b>	<b>7</b>
Explicit types	7
Implicit types	7
Any type	7
Interface	8
Optional properties	9
Return type	9
<b>Conclusie</b>	<b>10</b>

# Wat is Typescript?

Nota voor de volgers van Bill Gates (aka Windows users)

Als er in de onderstaande tekst naar de terminal wordt verwezen en Windows als besturingssysteem wordt gebruikt, betekent dit dat de opdracht moet worden uitgevoerd in **Git Bash**. Als CTRL+V niet werkt om de gekopieerde inhoud in de terminal te plakken, probeer dan SHIFT+INSERT.

TypeScript is een “strongly typed” programmeertaal. In tegenstelling tot JavaScript, wat een zwak getypeerde programmeertaal is.

Betekent dit dat het nodig is om een geheel nieuwe taal vanaf het begin te leren? Helemaal niet! Alle JavaScript-code is ook geldige TypeScript-code. TypeScript is een superset van JavaScript, het biedt extra syntaxis bovenop JavaScript om type veiligheid te bereiken/garanderen.

## JS vs TS

```
// JavaScript - index.js
let fullName;
fullName = "Massimo";

function sayHello(name, age) {
  return `Hello ${name}, you are ${age} years old!`;
}

const result = sayHello(fullName, 45);
console.log(result);
```

```
// Typescript - index.ts
let fullName: string;
fullName = "Massimo";

function sayHello(name: string, age: number): string {
  return `Hello ${name}`;
}

const result = sayHello(fullName, 45);
console.log(result);
```

Er zijn twee verschillen tussen TypeScript en JavaScript.  
Types zijn toegewezen aan variabelen/attributen:

**: string** achter de variabele fullName,  
de parameternaam en de functie/methode sayHello.  
**: number** achter de parameter leeftijd.

De bestandsnaamextensie is .ts in plaats van .js.

## Onze webbrowser werkt alleen met HTML, CSS en JavaScript

Juist! TypeScript wordt gecompileerd naar JavaScript. In de meeste moderne frameworks zal dit automatisch gebeuren. Om dit beter te begrijpen, bekijken we een voorbeeld.

**Maak een nieuwe map om in te werken en cd erin.**

```
mkdir typescript-example && cd $_
```

Wat extra terminal toelichting:

**mkdir typescript-example** maakt een nieuwe map aan: typescript-example.

**&&** voert de opeenvolgende commando's simultaan uit.

**cd** is uiteraard de "change directory".

**\$\_** refereert naar de pas aangemaakte map

**Maak van de directory een npm-project.**

```
npm init
```

Druk op Enter bij alle vragen die volgen op dit commando.

Er is een bestand package.json aanwezig, dit betekent dat dit een npm-project is dat de installatie van afhankelijkheden oftewel dependencies mogelijk maakt.

dependencies zijn software pakketjes gemaakt door derden, beschikbaar op het npm (Node Package Manager) netwerk.

Om TypeScript te kunnen gebruiken, is het noodzakelijk om het typescript-pakket te installeren.

```
npm install --save-dev typescript
```

**Opmerking:** als een dependency alleen nodig is voor ontwikkeling, is het mogelijk om deze als een devDependency te installeren. Dit kan gedaan worden door --save-dev flag te gebruiken. Als deze ook nodig is in de productie, installeer deze dan zonder de --save-dev flag. Hiermee wordt de dependency toegevoegd aan de property dependency in het bestand **package.json** bestand.

na de installatie beschikken we over het **tsc** commando (Typescript compiler)

We voegen hierna een extra script toe aan de package.json

```
// package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "compile": "tsc index.ts", /////////////// new ///////////////
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

## Voeg een index.ts file toe

Aangezien alle JavaScript geldige TypeScript is, kan JavaScript in het bestand index.ts worden geplaatst.

```
let fullName;
```

```
fullName = "Massimo";

function sayHello(name, age) {
  return `Hello ${name}, you are ${age} years old!`;
}

const result = sayHello(fullName);
console.log(result);
```

Vervolgens kan het TypeScript-bestand worden gecompileerd met behulp van de TypeScript-compiler om een JavaScript-bestand te krijgen.

```
npm run compile
```

De uitvoer in de terminal na het uitvoeren van de bovenstaande opdracht:

```
npm run compile

> test@1.0.0 compile
> tsc index.ts

index.ts:8:16 - error TS2554: Expected 2 arguments, but got 1.

8 const result = sayHello(fullName);
    ~~~~~

index.ts:4:25
   4 function sayHello(name, age) {
     ~~~
   An argument for 'age' was not provided.

Found 1 error in index.ts:8
```

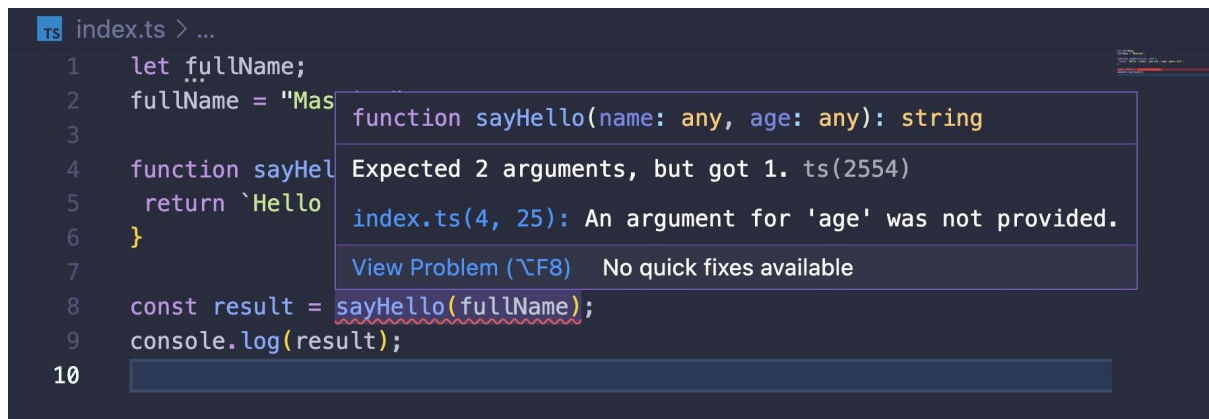
De TypeScript-compiler ziet dat er twee parameters nodig zijn voor de functie sayHello, maar er is er maar één beschikbaar. Ook al geeft dit een fout weer, de compiler compileert nog steeds de code en het resulterende JavaScript-bestand index.js wordt aangemaakt.

```
var fullName;
fullName = "Massimo";
```

```
function sayHello(name, age) {  
    return "Hello ".concat(name, ", you are ").concat(age, " years old!");  
}  
var result = sayHello(fullName);  
console.log(result);
```

Zoals je kan zien zijn de ES6 template literals vervangen door oeroude ES5 en lager concat chains. De “moderne” typescript werd aldus succesvol getranspiled. van .ts naar .js;

Ook in onze visual studio code zien we nu dat er effectief een probleem is met de code alvorens we de compiler loslaten.



```
1 let fullName;  
2 fullName = "Mas  
3  
4 function sayHello  
5     return `Hello  
6 }  
7  
8 const result = sayHello(fullName);  
9 console.log(result);  
10
```

function sayHello(name: any, age: any): string  
Expected 2 arguments, but got 1. ts(2554)  
index.ts(4, 25): An argument for 'age' was not provided.  
View Problem (⌘F8) No quick fixes available

## Het toevoegen van types aan attributen en methodes

We veranderen volgende items in onze index.ts:

**: string en : number** typings werden toegevoegd aan de fullName, de parameters name, age en een type werd ook toegevoegd aan de methode sayHello. de arguments werden geswapped wanneer we de methode aanroepen. De 2de parameter heeft een typo: fulName idp van fullName.

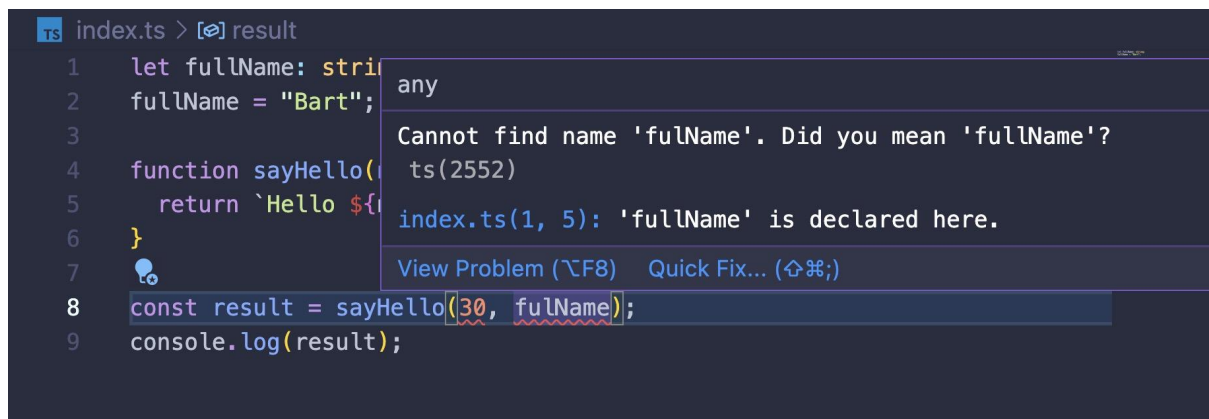
```
let fullName: string;  
fullName = "Massimo";  
  
function sayHello(name: string, age: number): string {
```

```
return `Hello ${name}, you are ${age} years old!`;
}

const result = sayHello(45, fullName);
console.log(result);
```

- Argument of type 'number' is not assignable to parameter of type 'string'
- Cannot find name 'fullName'. Did you mean 'fullName'?

Door gebruik te maken van Typescript detecteren we typo's en vergissingen in de volgorde van parameters sneller en with less effort! Visual studio code suggereert ook een quickfix (ook zonder co-pilot :)



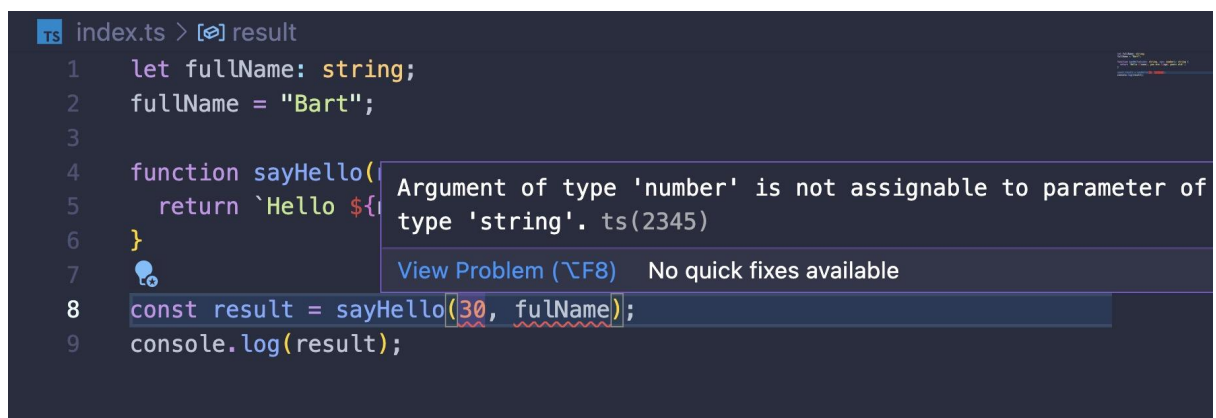
```
TS index.ts > [0] result
1 let fullName: string;
2 fullName = "Bart";
3
4 function sayHello(
5   return `Hello ${
6 }
7
8 const result = sayHello(30, fullName);
9 console.log(result);
```

any

Cannot find name 'fullName'. Did you mean 'fullName'? ts(2552)

index.ts(1, 5): 'fullName' is declared here.

View Problem (⌘F8) Quick Fix... (⇧⌘;)



```
TS index.ts > [0] result
1 let fullName: string;
2 fullName = "Bart";
3
4 function sayHello(
5   return `Hello ${
6 }
7
8 const result = sayHello(30, fullName);
9 console.log(result);
```

Argument of type 'number' is not assignable to parameter of type 'string'. ts(2345)

View Problem (⌘F8) No quick fixes available

# TypeScript basics

## Explicit types

Wanneer wij als ontwikkelaars een variabele maken, is het handig om te weten van welk type de variabele is. Dit kan expliciet worden gedeclareerd met TypeScript. Dit kan gedaan worden met de `:` type syntaxis.

```
let fullName: string;  
fullName = "John Duck.";
```

## Implicit types

Het is echter niet altijd nodig om expliciet aan te geven wat voor type een variabele is. TypeScript kan dit ook zelf bepalen. We noemen dit impliciete typen.

```
const fullName = "John Duck.";
```

Omdat aan de variabele `fullName` een waarde is toegewezen, kan TypeScript zelf bepalen dat het type van deze variabele het type waarde is dat eraan is toegewezen. In dit geval is het type **string**.

Dit concept wordt ook wel **type inference** genoemd.

## Any type

Wanneer een variabele wordt gedeclareerd zonder toegewezen waarde, is de variabele van het type `any`. Dit is een type dat elk type kan bevatten. Ook wel een impliciete `any` genoemd.

```
let fullName; // implicit any  
let fullName: any; // explicit any  
// Note: TypeScript will not complain about the above two lines
```



```
// because it is a valid JavaScript code
// not recommended; better to use explicit any or another more specific type
```

## Interface

```
const person = {
  firstName: "John"
  lastName: "Duck",
  age: 45,
};
```

In het bovenstaande voorbeeld worden impliciet types toegewezen aan de constante person. Dit is een object met de eigenschappen firstName, lastName en age. Deze eigenschappen hebben elk een impliciete typering. Dit is een **string** voor voornaam en achternaam en een **number** voor leeftijd.

Als we expliciet typen gebruiken ziet dit er als volgt uit:

```
const person: {
  firstName: string,
  lastName: string,
  age: number,
} = {
  firstName: "John"
  lastName: "Duck",
  age: 45,
};
```

Dit is niet leesbaar en slordig 😞 en op dit op te lossen gebruiken we een interface dewelke als multi object type zal fungeren.

```
interface Person {
  firstName: string;
  lastName: string;
  age: number;
}

const person: Person = {
  firstName: "John",
  lastName: "Duck",
};
```

```
    age: 45,  
};
```

## Optional properties

```
interface Person {  
    firstName: string;  
    lastName: string;  
    age?: number;  
}  
  
const person: Person = {  
    firstName: "John",  
    lastName: "Duck",  
    age: 45,  
};  
  
const person2: Person = {  
    firstName: "Janine",  
    lastName: "Duck",  
};
```

Met het **age?** maken we age optioneel... en zal aldus geen foutmelding geven omdat er in person2 geen age wordt meegegeven;

## Return type

Niet alleen variabelen hebben een bepaald type, ook functies hebben een type. Dit type wordt bepaald door de waarde die wordt geretourneerd.

```
function getFullName(firstName: string, lastName: string) {  
    return `${firstName} ${lastName}`;  
}  
  
const returnFullName = (firstName: string, lastName: string) => {  
    return `${firstName} ${lastName}`;  
};
```

In de bovenstaande functies wordt een string geretourneerd. Daarom is het retourtype impliciet een tekenreeks. Eén functie wordt gedefinieerd met het functie sleutelwoord en één met een anonieme fat arrow functie.

Dit kan ook expliciet worden vermeld.

```
function getFullName(firstName: string, lastName: string): string {  
    return `${firstName} ${lastName}`;  
}  
  
const returnFullName = (firstName: string, lastName: string): string => {  
    return `${firstName} ${lastName}`;  
};
```

Dit gebeurt in beide gevallen door het type toe te voegen na het laatste ronde haakje. **returned de functie niets** (bv een console.log) dan wordt het type **:void** gebruikt.

## Conclusie

**Alle JavaScript is geldig TypeScript.** TypeScript voegt extra functionaliteit toe om het leven van ontwikkelaars gemakkelijker te maken, fouten zoals het doorgeven van verkeerde typen worden gedetecteerd tijdens het compileren. Aangezien browsers alleen JavaScript ondersteunen, geen TypeScript, is het **noodzakelijk om TypeScript naar JavaScript te compileren.**

Veel javascript frameworks waaronder **Angular maken by default gebruikt van de Typescript.** In dit geval gebeurt de **compilatie automatisch** en hoeven we aldus geen extra actie te ondernemen.

