TypeScript Basics

Let's dig deeper into TypeScript

Introduction

After seeing how straightforward the TypeScript Compiler works, we will continue to learn TypeScript basics.

There are many exercises and solutions in this chapter to follow.

A simple example

Let us inspect a very first example:

```ts
// 02_basics/listings/0_simple_example.ts

const message = "Hello there!";

message.toLowerCase();
message();
```

But some questions appear:

- Is "message" even callable?
- Does it have the property called toLowerCase on it?
- If it does, is toLowerCase even callable?
- If both are callable, **what do they return?**

A simple example

Most of the time, the developer has all this information `stored in his/her head`.

Probably there are some comments here and there that describe some behavior.

But still, it will be hard for another developer to jump in and continue where the code was left off.

A simple example

To start everything, we like to program and evolve **a greeter**.

To enable our tests, we will need to install TypeScript globally (**if not already done**):

```
1  npm install -g typescript
```

Note: -g installs globally

A simple example

If we look at our IDE, we can see that TypeScript has already picked up one of the main `pain-points` mentioned before:


```
const message = "Hello there!";


message.toLowerCase();
message();
```

TS2349: This expression is not callable.
    Type 'String' has no call signatures.

Suppress with @ts-ignore ⌥⇧↵   More actions... ⌥↵

const message: "Hello there!"

▬ 0_simple_example.ts

Motivation

Next, we will see a more sophisticated example of why TypeScript can help us in such cases.

Motivation

First, look into an example and check if everything works out correctly.

```ts
1  // 02_basics/listings/00_tooling.ts
2
3  import express from "express";
4  const app = express();
5
6  app.get("/", function (req, res) {
7  res.se
8  });
```

Motivation

It looks like our IDE does not understand what we like to do:

```
import express from "express";
const app = express();

app.get("/", function (req, res) {
    res.se
})

app.listen()
```

This is understandable since **we do not have express nor the according types installed**.

Motivation

Let us now install express and the typings with the following commands
(it will create a node-modules folder, package.json, and package-lock.json):

```
1  npm i --prefix . express
2  npm i --prefix . --save-dev @types/express
3  npm i --prefix . --save-dev ts-node
```

Note: --prefix . will install it directly in the same folder level. --save-dev ts-node is not absolutely needed but helps for some errors.

Motivation

Now it looks much better:

```
import express from "express";
const app = express();

app.get("/", function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) {
    res.sen
})                p send     <ResBody>(body?: ResBody) => Response<ResBody, Locals>
                   sendfile (path: string)                              void
app.1              sendFile (path: string, fn?: Errback)               void
               m  sendStatus (code: number)        Response<ResBody, Locals>
               f  sendDate                                          boolean
               m  setDefaultEncoding (encoding: Buff…  Response<ResBody, Locals>
               p  headersSent                                       boolean
                  Press ^. to choose the selected (or first) suggestion and insert a dot afterwards  Next Tip
```

Not only have the hints changed, but also our IDE knows about express.

A greeter

Now we will start to define a simple greeter. You will find the simplest greeter ever in *02_basics/listings/01_greetings.ts*:

```
1  // 02_basics/listings/01_greetings.ts:
2
3  console.log("Hello world!")
```

A greeter

Since it has a .ts suffix, we will try to convert it to JavaScript:

```
1  cd 02_basics/listings/
2  tsc 01_greetings.ts
```

A greeter

If it has worked, you should now see *02_basics/listings/01_greeting.js* created with the following content:

```
1  console.log("Hello world!");
```

Oh wait, the content has **not changed**, and this shows us one of the essential TypeScript facts.

A regular JavaScript-File can be stored as a .ts and compiled without an issue. Of course, this is not what we want, but it is important to remember.

An improved greeter

Let us now see an improved (JS) version of the greeter in
02_basics/listings/02_improved_greeter.js:

```
1  // 02_basics/listings/02_improved_greeter.js
2
3  function greet(person, date) {
4      console.log(`Hello ${person}, today is ${date}!`);
5  }
6
7  greet("David");
```

What will be the output of this console.log?

An improved greeter

It will be: Hello David, today is undefined!

And there we have **`a bug`** and JavaScript did not complain at all :-(
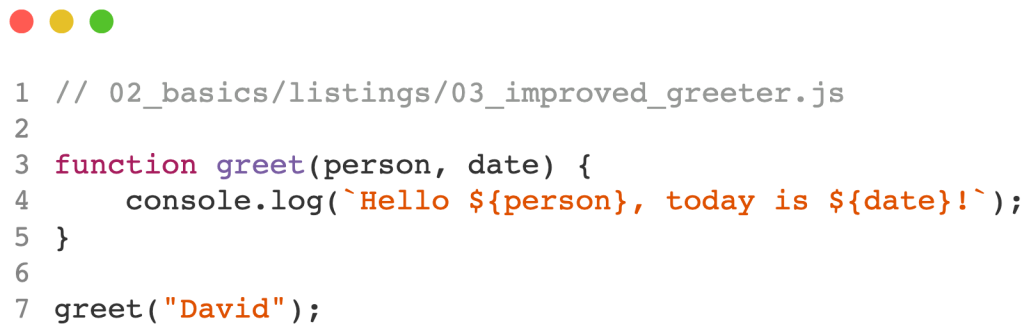
An improved greeter

Not complaining makes sense since nothing is wrong (at least not for JavaScript).

We only provided **one parameter** where two were expected, and JavaScript automatically added **undefined**.

Would it not be nice to see such errors during programming instead runtime?
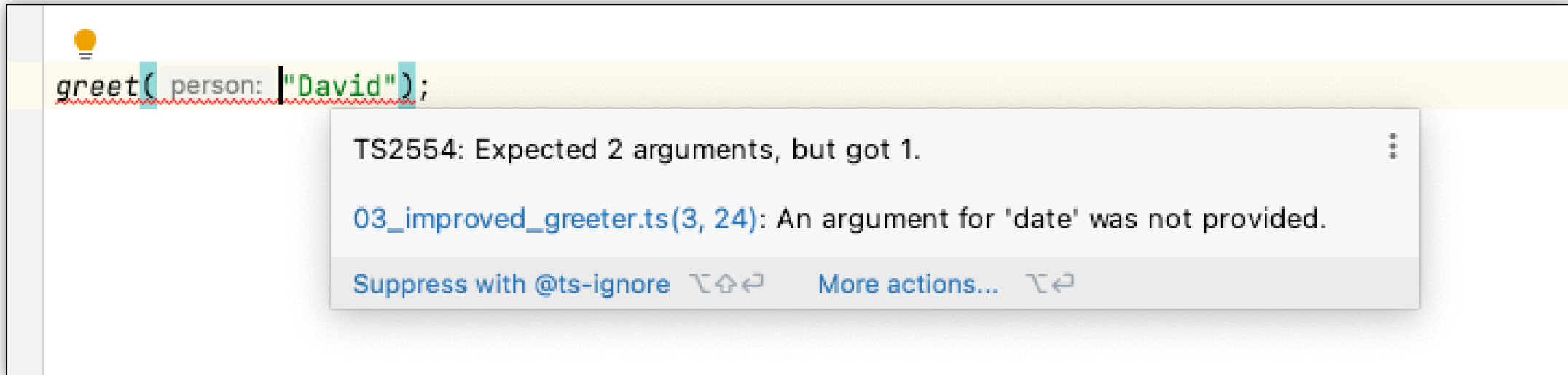
An improved greeter

Let us now see an improved (TS) version of the greeter:

```
1  // 02_basics/listings/03_improved_greeter.js
2
3  function greet(person, date) {
4      console.log(`Hello ${person}, today is ${date}!`);
5  }
6
7  greet("David");
```

The code is still the same, but what do you see in your IDE? Or if you run:
`tsc 03_improved_greeter.ts`?

An improved greeter

That is a much better experience, we know now during coding that something is missing.

An improved greeter

We can now fix it easily:

```ts
// 02_basics/listings/04_improved_greeter_fixed.ts

const greet = (person, date) => {
    console.log(`Hello ${person}, today is ${date}!`);
}

greet("David", "11th November");
```

An improved greeter

Of course, we can also use the newer arrow syntax; I will use both during the course interchangeably:

```ts
1  // 02_basics/listings/05_improved_greeter_fixed_es6.ts
2
3  const greet = (person, date) => {
4      console.log(`Hello ${person}, today is ${date}!`);
5  }
6
7  greet("David", "11th November");
```

It works now, but it is still not what we wanted; we are unsure that person is a string and date is a Date.

An improved greeter

We need to do something, now... typings to the rescue!

```ts
1  // 02_basics/listings/06_improved_greeter_fixed_es6_types.ts
2
3  const greet = (person: string, date: Date) => {
4      console.log(`Hello ${person}, today is ${date.toDateString()}!`);
5  }
6
7  greet("David", "11th November");
```

An improved greeter

And now as we run it with our TypeScript compiler we get the expected error:

```
1  TSError: Unable to compile TypeScript:
2
3  06_improved_greeter_fixed_es6_types.ts:7:16 - error TS2345: Argument of type 'string'
4  is not assignable to parameter of type 'Date'
```

An improved greeter

Let us fix it one last time:

```ts
1  // 02_basics/listings/07_improved_greeter_fixed_es6_types_v2.ts
2
3  const greet = (person: string, date: Date) => {
4      console.log(`Hello ${person}, today is ${date.toDateString()}!`);
5  }
6
7  // Careful, Date would return a string not a data object
8  greet("David", new Date());
```

Now it runs like it should and the TypeScript compiler is happy.

Types - Infer

Before we start talking more about types, let us see another simple example:

```
1 // 02_basics/listings/08_typescript_infer.ts
2
3 let mes = "Hello there!"
```

Types

When we hover over the content we can see that TypeScript automatically **`inferred`** the type for us.


```
// Typescript knows: let message: string
let mes = "Hello there!"

    let mes: string

    08_typescript_infer.ts
```
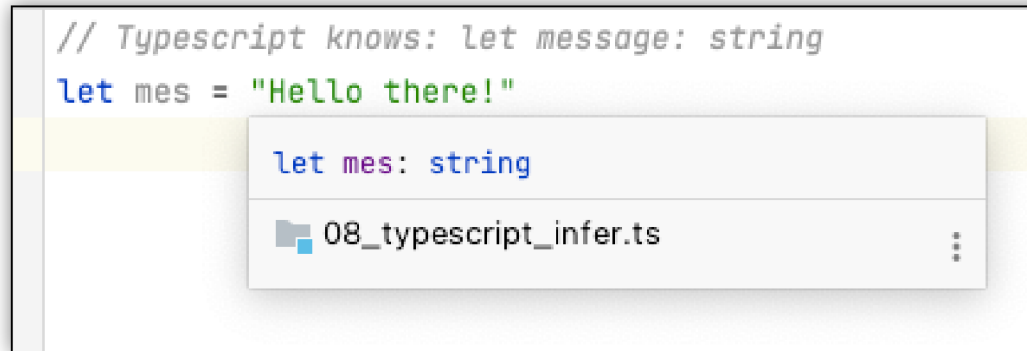
This is awesome, but most of the time you will write down the respective types for:

- increased readability
- no wrongly inferred types

Type Annotation

Please remember:

Type annotations **never change** the runtime behavior of your program.

Downleveling

Another thing we have quickly seen before is **downleveling**. Since not all language features are available in each ECMAScript version, the compiler output will look different depending on the target selected.

```
 1  // 02_basics/listings/09_downleveling
 2
 3  // tsc 09_downleveling.ts --target es3 (deprecated in TS 5.5)
 4  // tsc 09_downleveling.ts --target es5
 5  // tsc 09_downleveling.ts --target es2015
 6  // alternative: tsc --project tsconfig.json
 7
 8  const greet = (person: string, date: Date) => {
 9      console.log(`Hello ${person}, today is ${date.toDateString()}!`);
10  }
11
12  greet("David", new Date());
```

How does the output look like?

Downleveling

While ES2015 knows Template-Strings:

```
1   `Hello ${person}, today is ${date.toDateString()}!`;
```

This is a feature not introduced in ES3/ES5 - therefore the output is different:

```
1   "Hello " + person + ", today is " + date.toDateString() + "!";
```
ES3

ES5
```
1   "Hello ".concat(person, ", today is ").concat(date.toDateString(), "!"
```

Note: ES3 is "sadly" deprecated since TS 5.5, so this test is sadly gone

Strictness

Before we start diving into learning TypeScript, there are two important settings to be mentioned which can be set in tsconfig.json:

- noImplicitAny
- strictNullChecks

NoImplicitAny

Recall that in some places, TypeScript does not try to infer any types for us and instead falls back to the most lenient type: `any`.

This is not the worst thing that can happen - after all, falling back to any is just the plain JavaScript experience.

strictNullChecks

By default, values like null and undefined are assignable to any other type.

This can make writing some code more accessible, but forgetting to handle null is the cause of countless bugs in the world.

Good Job!

Well, that was a good first dive into TypeScript.

End

That was all for this chapter