Assertion Functions

This lesson talks about a new feature of TypeScript 3.7: the assertion function.

WE'LL COVER THE FOLLOWING

- Asserting untyped code
- Custom assertions

Asserting untyped code

In JavaScript, you must assert the variables' type since it is not a typed language. While the assertion function is not something you would require using purely TypeScript, it is handy to know in case you want to have a transpilation error on assertion.

```
function showLandArea(address, x, y) { // No typê
    assert(typeof address === "string");
    assert(typeof x === "number");
    assert(typeof y === "number");

    console.log(`The house ${address.substr(10)} as an area of ${x * y} meters`)
}
showLandArea("1234 Street ABCDE", 10, 5);
// showLandArea("1234 Street ABCDE", "10", "5"); // Assertion will catch the 10 and 5 as stri
```

With **TypeScript 3.7** and above, the assert condition (the first parameter of the `assert` function) must be `true` or will throw an `AssertionError`. Otherwise, it keeps executing.

What is new is that TypeScript is smart enough to transform the untyped parameter into the asserted type after the assertion. This means that in the previous example, after the assertion, the value of `address` is `string` and the other two variables are `numbers`. The advantage is that, while editing the code,

you get Intellisense suggestions for the right type of each member, but you also avoid having in the method the parameter at `any`.

The previous code at **line 8** executes well because of the type is respected even if not defined. Uncommenting **line 9** results from the assertion to raise and transpilation fail.

Custom assertions

TypeScript 3.7 also gives you the option to create your own custom assertion function that has the same ability to cast automatically if no exception is thrown from the assertion function. The assertion function requires you to have a return type, declared by using `asserts P is string` where P is the parameter name. The following example asserts the address to be a string with at least 3 characters.

Line 3 and **Line 6** in the assert function of **line 1** are the ones that verify the assertion conditions. If neither of these two is true, the function behaves it was not invoked.

```
function assertAddress(address: any): asserts address is string {
    if (typeof address !== "string") {
        throw new Error("Not a string!");
    }
    if(address.length < 3){
      throw new Error("Address must be above 3 characters");
    }
}

function createAddress(newAddress: any): void{
   assertAddress(newAddress);
   // newAddress is a string from that point if a string with 4+ characters
}
```

In most modern TypeScript, relying on type directly is straightforward and easy to refactor, understand the meaning of a variable and more align with TypeScript's features. Nonetheless, assertion is a tool that is good to have in your toolbox.