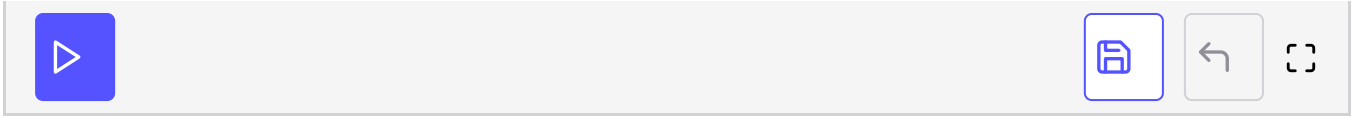
Assertion Functions

This lesson gives more detail on assertion functions.

The assertion function is new to **TypeScript version 3.7**. It aims to not only ensure that a value is of a specific type, but to throw at (run-time) an exception if it does not respect the expected type. The assertion function use cases are more oriented for a codebase that is coming from Javascript were asserting types was a common practice. With a TypeScript only code base, it is not as needed since type is defined at the declaration.

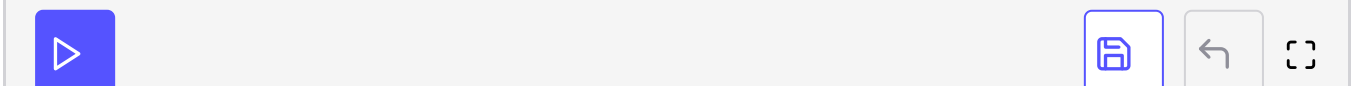
```
function getLength(str: any): number {
    if (typeof str !== "string") {
        throw Error("String Required")
    }
    return str.length;
}
```



The problem is the repetition of errors. With the assertion functions, you can wrap the exception into a function. At **line 8**, the function calls the *assertion function*. The syntax at **line 1** has a function with `asserts val is string` which indicates TypeScript how to handle the type passed in the first parameter.

```
function isString(val: any): asserts val is string {
    if (typeof val !== "string") {
        throw Error("String Required");
    }
}

function getLength(str: any): number {
    isString(str);
    return str.length;
}
```



The code will throw an Error when the condition fails. The assertion function tells TypeScript that the variable is actually a `string` and underneath the call, to the assertion function the type of the variable will be set. In this case, to a `string`.

The assertion function's new capabilities reside in the return type. The previous example uses `asserts val is string`. In a more generic way, you see that as: `assert <VAR> is <TYPE>`. The assertion functions can have as many parameters as you want and any kind of condition inside. If the function does not throw an exception, then the return type is the one from the `<TYPE>` in the return type of the function.

```
function isLengthRangeString(val: any, min: number, max: number): asserts val is string {
    if (typeof val !== "string") {
        throw Error("String Required");
    }
    const len = val.length; // For sure `val` is a string here
    if (len < min || len > max) {
        throw Error(`String is not in the range of ${min} and ${max}`);
    }

}
function getLength(str: any, min: number): number {
    isLengthRangeString(str, min, 15);
    return str.length;
}

console.log(getLength("Good", 1));
// console.log(getLength("NoGood", 10));
// console.log(getLength("Not Good because way too long", 1));
```

The example above shows three examples. The first one is fine and respects the boundary of the assertion function. The next two are not. Even if the types are all strings, they do not respect additional validations of the assertion function.