Avoiding `any` at Any Time Possible

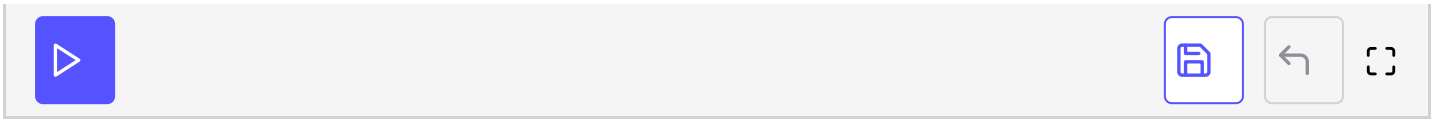
In this lesson, you will learn about a variable type that you should only use in a particular situation.

We'll cover the following

Dangerous world

You must avoid (as much as possible) the type `any`, principally because it can hold any value and therefore doesn't enforce any protections. If you are integrating an existing JavaScript project with TypeScript, every variable will be, by default, set to `any` until they are defined.

```
let x: any = "string";
x = true;
x = { title: "Object with a string member" };
x = [1, 2, 3];
x = 1;
```



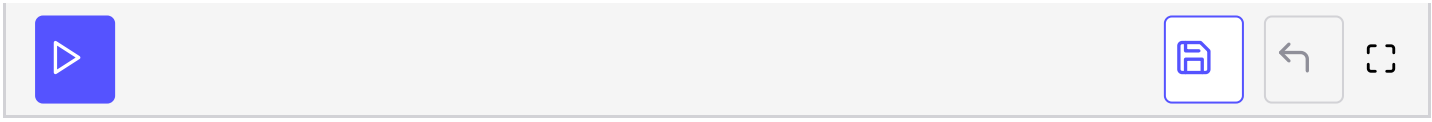
This is also the case with a value coming from an Ajax response in JSON format. Every `any` variable will let you assign any value but could also invoke any function.

The following code has a response of type `any` on **line 8**. Unfortunately, the response can be a string or a JSON object of any form. It is possible to mitigate this issue with *casting*, which you will see later to not propagate the `any` further in the code.

> Move your cursor above `req.response` in the following code (line 8) to see the type of `any`.

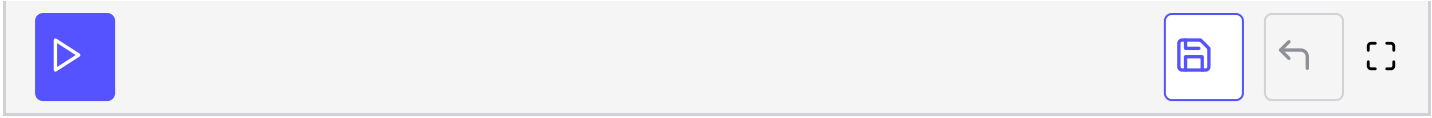
```
function get(url: string) {
    return new Promise(function(resolve, reject) {
        var req = new XMLHttpRequest();
        req.open("GET", url);

        req.onload = function() {
            if (req.status == 200) {
                resolve(req.response);
            } else {
                reject(Error(req.statusText));
            }
        };
    });
}
```



The danger is that the function may not be available. For example, say you set a variable with a number value that calls for an array of the function `.length`. This will transpile, but raise a runtime exception because a number doesn't have a length function in the browser, and return `undefined` when running under Node js.

```
let myAnyString: any = 123;
console.log(myAnyString.length);
```



Readability

A piece of code that uses `any` is harder to maintain because it is harder to understand. The way code is typed is a live documentation of what is expected. For example:

```
function configure(object: any, option: any) {
    // ...
}
```

should tell you less than:

```
interface Server {
    ipv4: string;
    ipv6: string;
```

```
    port: number;
    https: boolean;

}
interface ServerOptions {
    maxUser: number;
    maxConcurrentRequests: number;
}

function configure(object: Server, option: ServerOptions): Server {
    //...
    return object;
}
```

While both pieces of code in execution will perform in the same way, the second one is clearer about what inputs are needed and what the output will be. Similarly, the readability inside functions is improved when a local variable is well defined.