How to Narrow a Type with the in Operator

This lesson will look at how to use in to narrow a type.

WE'LL COVER THE FOLLOWING

The `in` operator explained

The `in` operator can narrow a type from a union. The left part of the operand is a string or a string literal. The right part is a union type. The result is a `Boolean` that returns `true` if the union contains the string, and `false` if not.

Usage of the `in` operator explained

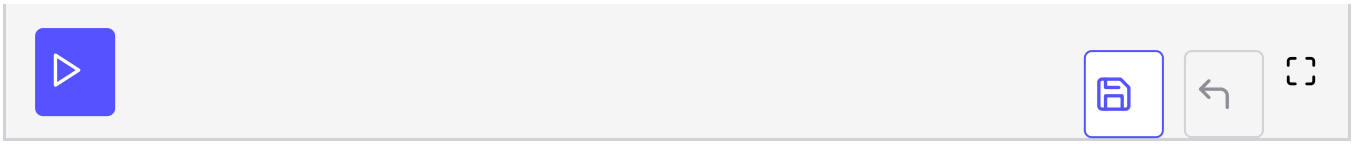
In the example below, the parameter `x` can be of two types: `IN_A` or `IN_B`. The member `m1` is unique to `IN_A`, hence can be used to discriminate which interface is passed down. By using the `in` operator the code will go in the `if` if the type is `IN_A`.

```
interface IN_A {
    m1: number;
    m2: boolean;
}
interface IN_B {
    m3: string;
}

function foo(x: IN_A | IN_B) {
    if ("m1" in x) { // m1 is only in IN_A
        console.log("Type narrowed to IN_A", x.m1, x.m2);
    } else { // IN_B
        console.log("Type narrowed to IN_B", x.m3);
    }
    console.log("A is still IN_A or IN_B");
}

foo({ m1: 1, m2: true }); // Implicit IN_A
```

```
foo({ m3: "" }); // Implicit IN_B
```



Narrowing the type gives the advantage that within the scope of the conditional statement, TypeScript provides Intellisense to cover all members of the type as well as ensure that during transpilation, all members used are the ones from the narrowed type.