Booleans, Functions, and Objects

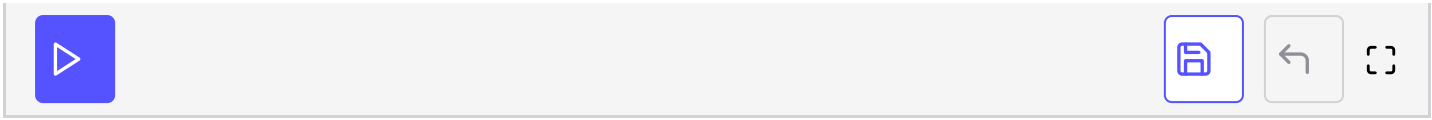
In this lesson, you will learn how to declare a variable that can hold true or false.

We'll cover the following

Boolean primitive type

A boolean value is the most basic primitive in JavaScript and it remains the same with TypeScript. Boolean values restrict the assignment to two values: `true` and `false`. These terms are case sensitive – only the **lowercase** format is accepted.

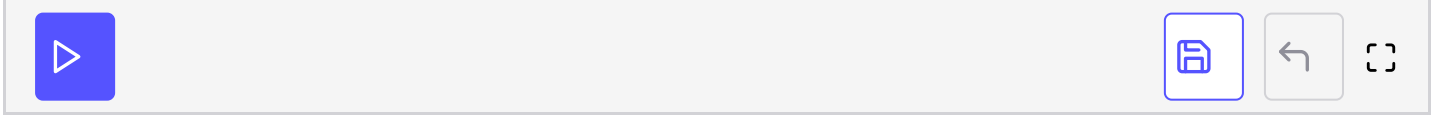
```
let b: boolean = true;
console.log(b);
```



You cannot assign the value `0` or `1`, or the `true` or `false` values using any upper case letters.

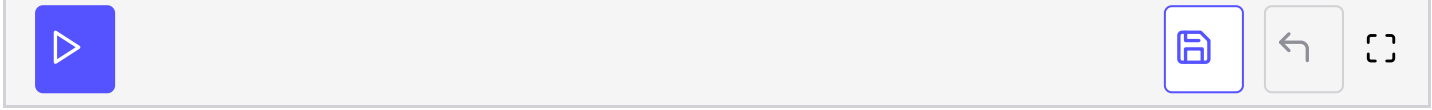
> Note: The following case shows two cases that TypeScript will not ✕ let you compile.

```
let bWithUumber: boolean = 0;
let bWithUpperCase: boolean = TRUE;
```



You can reverse a boolean value by assigning the same value with an exclamation point before it.

```
let myValueCanChange: boolean = true;
console.log(myValueCanChange);
myValueCanChange = !myValueCanChange;
console.log(myValueCanChange);
```



Boolean values are often the result of an operation like `===` or `<`, `>`, `!==`, `<=`, `>=`.

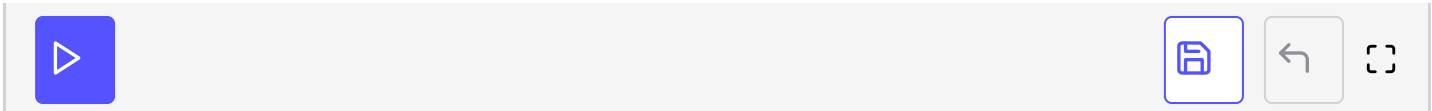
```
let number1 = 5;
let number2 = 5;
let number3 = 100;
let biggerThan: boolean = number1 > number3;
console.log(biggerThan); // False

let smallerThan: boolean = number1 < number3;
console.log(smallerThan); // True

let biggerOrEqualThan: boolean = number1 <= number2;
console.log(biggerOrEqualThan); // True

let beEqual: boolean = number3 === number3;
console.log(beEqual); // True

let notBeEqual: boolean = number1 !== number3;
console.log(notBeEqual); // True
```

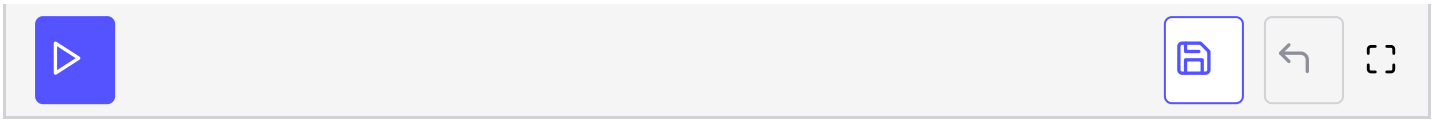


TypeScript allows other value types to act like a boolean. For example, `undefined` and `null`, `-0`, `0` and `NaN` will return `false`. This is because JavaScript includes many values to be “falsy”.

```
let x1 = undefined;
let x2 = null;
let x3 = -0;
let x4 = 0;
let x5 = NaN;
let x6 = {};
let x7: any[] = [];

function printTrueOrFalse(b: any | any[]): void{
  if(b){
    console.log(`The value ${b} is true`);
  } else{
     console.log(`The value ${b} is false`);
  }
}
printTrueOrFalse(x1);
printTrueOrFalse(x2);
printTrueOrFalse(x3);
printTrueOrFalse(x4);
```

```
printTrueOrFalse(x5);
printTrueOrFalse(x6);
printTrueOrFalse(x7);
```



The previous example prints “is false” for each function call, except for the last two.

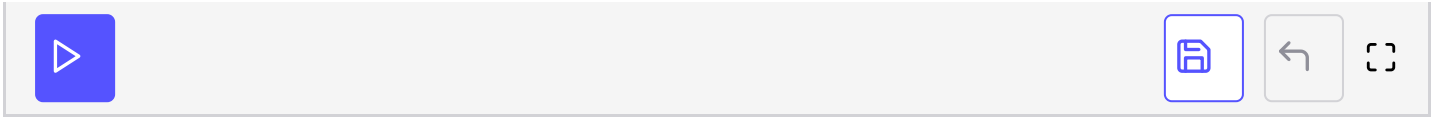
A detail to note is that the parameter of the function is of type `any`. If you change the type to `boolean`, TypeScript will not compile. The last two elements are true because an empty object is still an object and an array, even when empty, is still an array.

The `Boolean` function

TypeScript, like JavaScript, lets you invoke a `Boolean` function. This function proves handy to convert different types into a boolean type.

Typescript follows JavaScript rules, hence few values can return surprising results. For example, `Boolean("false")` will return `true` and `Boolean("not false")` will return `true;`.

```
console.log(Boolean(false)); // False
console.log(Boolean("false")); // True
console.log(Boolean("not false")); // True
```



Let’s keep in mind that `Boolean` is taking an `unknown` type that we will see in a few lessons.

Boolean objects

The boolean object behaves like the boolean function. One difference is that the result is not a `boolean` but an object that wraps the boolean value. The following outputs are objects like `[Boolean: false]` instead of the primitive variable value of `false`.

```
console.log(new Boolean(false)); // False
console.log(new Boolean("false")); // True
console.log(new Boolean("not false")); // True
```