What is a Number in TypeScript?

This lesson delves into how to define integers, floats and doubles in TypeScript.

WE'LL COVER THE FOLLOWING

Another very common primitive is probably the number. Since TypeScript is a superset of JavaScript, numbers work the same in both languages. The openness of JavaScript allows for a broad set of numbers. Integers, signed floats or unsigned floats are permitted. By default, a number will be base 10.

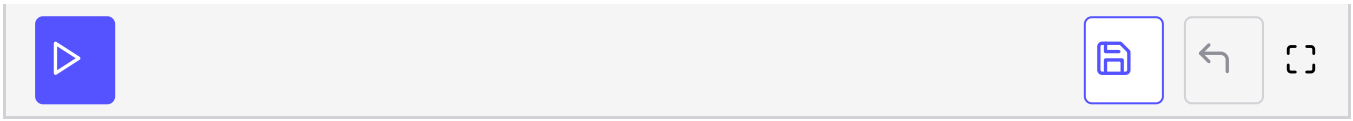
When a type is explicitly assigned to a variable, the type will be removed once the JavaScript is generated. The reason is that typing does not exist in JavaScript. It explains why TypeScript only has `number`. The following code will produce three variables without an explicit type in JavaScript but if `typeof` is used will return the dynamic type: `number`.

```
const x: number = 10;
let z: number = 15;
var p: number = 123;
console.log("Here are 3 variables of type number", x, z, p);
```

Number base

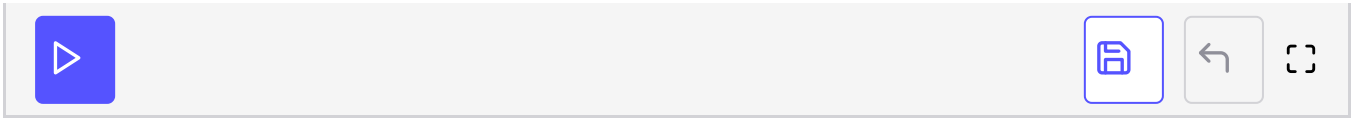
You can also assign base 16 (hexadecimal) or base 8 (octal) or base 2 (binary) with the prefix `0x`, `0o` and `0b`, though it's rarely used:

```
let dec: number = 10;
let hex: number = 0x10;
let octo: number = 0o10;
let bin: number = 0b10;
console.log("Here are 4 numbers", dec, hex, octo, bin);
```



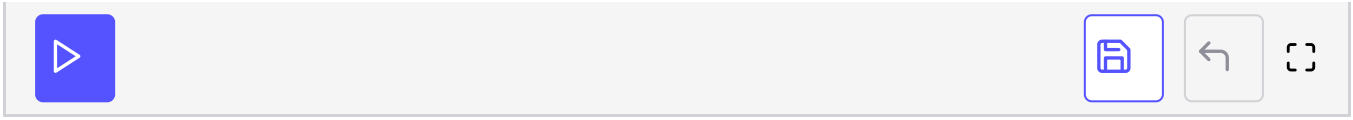
Like most variables in TypeScript, there is no need to explicitly mark the variable type at the time of declaration. TypeScript can infer the type. The following code is the same as the code above. If you move your cursor on top of each variable, you will see `number`.

```
let dec2 = 10;
let hex2 = 0x10;
let octo2 = 0o10;
let bin2 = 0b10;
console.log("Here is 4 numbers", dec2,hex2,octo2,bin2);
```



However, the following code does **not** define the four variable as `number`.

```
const dec2 = 10;
const hex2 = 0x10;
const octo2 = 0o10;
const bin2 = 0b10;
console.log("Here is 4 numbers", dec2,hex2,octo2,bin2);
```



The code compiles, but if you move your cursor above each variable you will see the type `10`, `16`, `8` and `2`. The type is actually the value meaning that **only** those values are acceptable. The difference between this snippet and the one before is `let` and `const`. With `let` the variable may be reassigned at any time during the life of the variable, hence the narrowest type that TypeScript can infer is `number`.

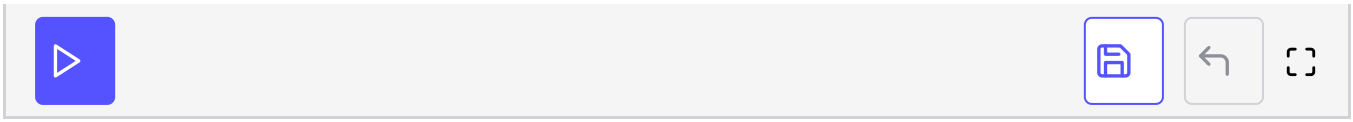
However, in the last example with `const`, TypeScript knows that the value will not change, hence it can narrow the type down to the only value possible.

The `number`: integer, decimal, and signed?

The `number` type is the same as in JavaScript: it defines the type for integer, float, double, etc. So integer, float and positive or negative numbers all will be referred to as the single type `number`. A type declared as `number` (implicitly or explicitly) can be checked at runtime as well with `typeof` which will return `number`.

Numbers are also not signed. This means they can be positive or negative.

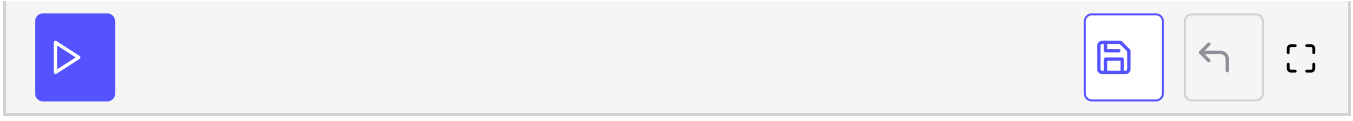
```
let int: number = 1;
let float: number = 1.1;
let negative: number = -100;
console.log(typeof(int));
console.log(typeof(float));
console.log(typeof(negative));
```



Not a Number (NaN)

In TypeScript, as in JavaScript, the type `number` can be assigned with `NaN` meaning that it is not a number.

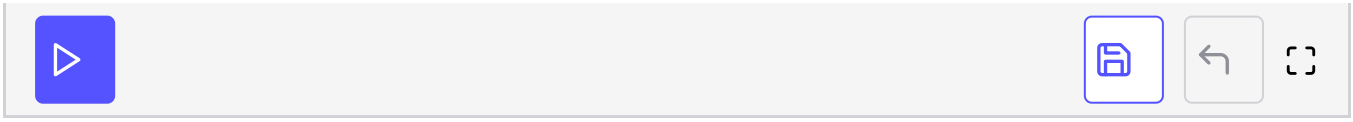
```
let myNumberIsNotANumber: number = NaN;
console.log(myNumberIsNotANumber);
console.log(typeof(myNumberIsNotANumber));
```



Separator

A numeric separator is a feature that simplifies how we write numbers. A long number can be hard to read and adding a separator can reduce confusion. When writing a number, you can use the underscore symbol to mark every thousand, for example. There is no rule on where to place a group separator other than it must be between two numbers.

```
const numericSeparator1 = 560000067;
const numericSeparator2 = 560_000_067;
const numericSeparator3 = 5_6_0_000_0_6_7;
const numericSeparator4 = Number(5_000);
const numericSeparator5 = Number("5_000"); // Nan
const numericSeparator6 = parseInt("5_000");
const numericSeparator7 =  0xFAB_F00D;
const numericSeparator8 =  0b1111_11111000_11110000_00001100;
console.log(numericSeparator1)
console.log(numericSeparator2)
console.log(numericSeparator3)
console.log(numericSeparator4)
console.log(numericSeparator5)
console.log(numericSeparator6)
console.log(numericSeparator7)
console.log(numericSeparator8)
```



When using `NaN` JavaScript cannot transform a string with a separator as a proper number. TypeScript does not warn or give an error at transpilation time because TypeScript only checks that the type is `string` which is legit but does not evaluate every operation.

Numeric separators work with decimal, octal, binary and hexadecimal. It is available from TypeScript 2.7 and is now available in the last version of ECMAScript. It is possible to use this feature in older versions of ECMAScript because TypeScript transforms the separator out during transpilation and you can target older versions of ECMAScript and still use the feature.