**Another Great Use-case for "typeof"**

The typeof operator can also be very useful for quickly getting the type of a (potentially complex) function.

Consider the following scenario:

You have a function sum(a: number, b: number) that sums two numbers. You also have subtract(a: number, b: number) that subtracts two numbers.

In addition, you have performMathAction(cb) which receives a cb parameter. cb should be either a pointer to the sum or the subtract function. So that you could either call performMathAction(sum) or performMathAction(subtract).

Hence the idea is that performMathAction calls sum or subtract internally (and passes values to those functions).

This scenario is, of course, a bit made up but the described use-case of having a function that needs to interact with other functions is absolutely not - it's instead something you'll often do in your JavaScript / TypeScript code.

In order to properly define performMathAction, you must declare the type of the cb parameter.

You could do it like this:

function performMathAction(cb: (a: number, b: number) => number) {

// some code...

}

But since you already defined sum and subtract somewhere else in your code, you can also simply derive the function type with help of typeof.

Like this:

function sum(a: number, b: number) {

return a + b;

}

function subtract(a: number, b: number) {

return a - b;

}

type SumFn = typeof sum;

type SubtractFn = typeof subtract;

function performMathAction(cb: SumFn | SubtractFn) {

// some code...

}