Lesson 12 - Mathematical functions in C# .NET - The Math library

[C# .NET](https://www.ict.social/csharp) [Basic constructs](https://www.ict.social/csharp/basics) Mathematical functions in C# .NET - The Math library

[PHP professional, $7000 a month. Like the sound of that? We'll get you there for $120.](https://www.ict.social/api/Abc-Def/def/8" \t "_blank)

[Discover more...](https://www.ict.social/api/Abc-Def/def/8" \t "_blank)

[Last time](https://www.ict.social/csharp/basics/multidimensional-arrays-in-csharp-net) we learned about multi-dimensional arrays in C# .NET. Learning C# .NET actually starts from now on, however, this online course of the most basic constructs of the language will be finished today. I'm glad that we've successfully reached this point. The next online course is focused on object-oriented programming. We'll create really interesting applications and even one game. We'll end this course with a simple article about mathematical functions that will certainly come in handy in our future programs.

In .NET, basic mathematical functions are included in the **Math class**. The class provides 2 fundamental constants for us: PI and E. PI is for sure the number Pi (3.1415...), and E is Euler's number, the base of the natural logarithm (2.7182...). I'm sure you'll get how to work with it, but just to be sure let's print these constants to the console:

[Run code](https://www.ict.social/) Click to edit

* Console.WriteLine("Pi: {0} \ne: {1}", Math.PI, Math.E);
* Console.ReadKey();

We see that we call everything in the Math class. There is nothing interesting about this code except that we've used the special character \n, which breaks a line.

Console application

**Pi: 3.14159265358979**

**e: 2.71828182845905**

Now, let's go over the methods that the Math class provides:

**Math class methods**

**Min(), Max()**

Let's start with the simple methods :) Both functions take two numbers of any data type as parameters. Min() returns the smallest number, Max() returns the greatest one.

**Round(), Ceiling(), Floor() and Truncate()**

All three functions are related to rounding. Round() takes a decimal number as parameter and returns the rounded number **of the double data type** in the way we learned in school (from 0.5 it rounds upwards, otherwise downwards). Ceiling() upwards and Floor() rounds downwards no matter what. Truncate() cuts the decimal part off and leaves whole number part intact (does not round whatsoever).

We'll certainly be using Round() very often. I practically used the other functions e.g. in determining the number of pages of a guestbook. When we've 33 comments and we print only 10 comments per page, they'll, therefore, occupy 3.3 pages. The result must be rounded up since there will be actually 4 pages.

If you think that Floor() and Truncate() do the same thing, think again! They behave differently for negative numbers. Floor() rounds negative numbers down to the next "more negative" number, Truncate() always rounds to zero when the input is negative.

We round decimal numbers and store them in int variables like this:

**double** d = 2.72;

**int** a = (**int**)Math.Round(d);

Casting to int is necessary despite of the Round() method returns a whole number. However, it's still of the doubletype, due to the fact that all mathematical functions work with double.

**Abs() and Sign()**

Both methods take a number of any type as a parameter. Abs() returns its absolute value and Sign() returns a number based on its sign, -1, 0 or 1 (for a negative number, zero and a positive number).

**Sin(), Cos(), Tan()**

Classic trigonometric functions, all take an angle as a double, which has to be entered in radians (not degrees if your country uses them). To convert degrees to radians we multiply them by (Math.PI / 180). The return value is also a double.

**Acos(), Asin(), Atan()**

Inverse trigonometric (arcus, sometimes cyclometric) functions, which return the original angle according to the trigonometric value. The parameter is a double and the returned angle is in radians (also as double). If we wish to have an angle in degrees, we have to divide the radians by (180 / Math.PI).

**Pow() and Sqrt()**

Pow() takes two double parameters. The first is the base of the power and the second is the exponent. If we wanted to calculate eg. 23, the code would be as following:

[Run code](https://www.ict.social/) Click to edit

Console.WriteLine(Math.Pow(2, 3));

Sqrt() is an abbreviation of SQuare RooT, which returns the square root of the number given as a double. Both functions return a double as the result.

**Exp(), Log(), Log10()**

Exp() returns the Euler's number raised to a given exponent. Log() returns the natural logarithm of a given number. Log10() returns the decadic logarithm of a number.

Hopefully, you noticed that the method list lacks any general root function. We, however, can calculate it using the functions the Math class provides.

We know that roots work like this: 3rd root of 8 = 8^(1/3). So we can write:

[Run code](https://www.ict.social/) Click to edit

Console.WriteLine(Math.Pow(8, (1.0/3.0)));

It's very important to write at least one number with a decimal point when we are dividing, otherwise, C# will assume that we want it to apply whole-number division, and the result would have been 8 ^ 0 = 1 in this case.

**Division**

Programming languages often differ in how they perform the division of numbers. You need to be aware of these issues to avoid being, unpleasantly, surprised afterwards. Let's write a simple program:

[Run code](https://www.ict.social/) Click to edit

**int** a = 5 / 2;

**double** b = 5 / 2;

**double** c = 5.0 / 2;

**double** d = 5 / 2.0;

**double** e = 5.0 / 2.0;

// int f = 5 / 2.0;

Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}", a, b, c, d, e);

Console.ReadKey();

We divide 5/2 for several times in the code, which is mathematically 2.5. Nonetheless, the results will not be the same in all cases. Can you guess what we'll get in each case? Go ahead, give it a try :)

The code wouldn't compile because of the line with the variable f, which we commented. The problem is that in this case is that one of the results is a decimal number, which we're trying to assign to an int variable. The program output will be the following:

Console application

2

2

2.5

2.5

2.5

We see the result of this division is sometimes decimal and sometimes whole. It doesn't really matter what the data type of the variable we're assigning the result to is. What really matters is the data type of the numbers we divide by. If one of the numbers is decimal, the outcome will always result in a decimal number. Division of 2 integers always returns an integer. Keep in mind that if you compute the average and want a decimal result, at least one variable must be cast to double.

**int** sum = 10;

**int** count = 4;

**double** average = (**double**)sum / count;

For example, [the PHP language](https://www.ict.social/php) always returns the decimal result of the division. When you divide in different programming languages make sure you check how division works there first before you use it.

**The remainder after division**

In our applications, we often need the remainder after integer division (i.e. modulo). In our example 5/2, the integer result is 2 and modulo is 1 (what left over). Modulo is often used to determine whether a number is even (remainder of division by 2 is 0). Modulo is useful if you want, for example, to draw a checkerboard and fill in the fields based on whether they are even or odd, to calculate the deviance of your position from some square grid, and so on.

In C# .NET and C-like languages in general, modulo is a percent sign, i.e. %:

[Run code](https://www.ict.social/) Click to edit

Console.WriteLine(5 % 2); // prints 1

Well, that's all I've got for you in this course. If you'd like to learn more about the [Basic constructs of C# .NET](https://www.ict.social/csharp/basics) or feel like you need more practice, take another look at the articles and lesson-specific exercises. Our C# course will be continued in [Basics of object-oriented programming in C# .NET](https://www.ict.social/csharp/oop). In the "next lesson": csharp/oop/in­troduction-to-object-oriented-programming-in-csharp we'll introduce you to an object-oriented world. We'll get acquainted with many things that have been kept secret from us until now :)

Download

[Download MathConstants.zip](https://www.ict.social/api/Articles-Article/file/56d59b3d2442f)

Downloaded 31x (33.36 kB)   
 Application includes source codes in language C#

Article has been written for you by [David Capka](https://www.ict.social/portfolio/1)

[](https://www.ict.social/portfolio/1)

Do you like this article?

Top of Form

Bottom of Form

1 votes

The author is a programmer, who likes web technologies and being the lead/chief article writer at ICT.social. He shares his knowledge with the community and is always looking to improve. He believes that anyone can do what they set their mind to.

The author learned IT at the [**Unicorn College**](http://www.unicorncollege.cz/en) - a prestigious college providing education