Chairbear
Digital Learning

C#

Math

Written 17th July 2020

Table Of Contents

Introduction

This book was written for educational purposes, by Jade Lei ©Chairbear, for use by any individual or organization under the BY-NC-ND creative commons licence (Attribution + Noncommercial + NoDerivatives).

This book assumes you understand the basics of C#, however, does iterate over some of the basic syntax and rules of the language in the prerequisite section. If you are not familiar with C#, we would recommend reading our C# Essentials book.

This book covers the methods applicable to C# Math class, and includes some additional information on numbers in C#. It is a guide to use as a reference, and includes examples of usage.

We acknowledge that not everyone learns the same way, and so suggest you find additional resources that accommodate your learning style.

Jade Lei
SEO Technical Specialist,

Junior Software Developer (2020)

Prerequisite C#

Here are some of the fundamental C# rules and syntax that you should know before continuing to explore the rest of this book.

```
using System;


namespace fruits
{
 class Program
 {
   static void Main(string[] args)
   {
       var Fruit = "apple";
      var FRUIT = "apple";
      var fruit = "apple";
      Console.WriteLine(FRUIT);
   }
 }
}
```

Case Sensitivity

C# is case sensitive. The variables ‘fruit’, ‘FRUIT’ and ‘Fruit’ are all different variables.

Line Spacing

C# ignores additional spacing, like the above space after `using System;` but just because you can use lots of spaces and tabs doesn't mean you should, the best practise is to include a space

on either side of operator and assignment signs (such as the = sign), to make the code more human readable.

For this same reason, avoid code lines longer than 80 characters. If a C# statement does not fit on one line, the best place to break it is after an operator.

Ending Statements

All C# statements should end with a semicolon, as best practice.

Comments

Not all C# statements are executed, code after double slashes (//) or between /* and */ is treated as a comment. Comments are useful for adding notes to bits of code to make it easier to understand when revisiting it later.

```
//this is a single line comment

/* this is

a  multi line comment */
```

Math

C# comes built in with a Math class which has many methods that allow you to perform mathematical tasks on numbers.

Methods Reference Table

Method	***Meaning***
Math.Abs(A)	**Math.Abs** Returns the absolute (positive) value of A
Math.Acos(A)	**Math.Acos** Returns the angle whose cosine is the specified number (A). Value passed in must be greater than or equal to -1, but less than or equal to 1, if not it returns NaN (not a number)
Math.Asin(A)	**Math.Asin** Returns the angle whose sine is the specified number (A). Value passed in must be greater

	than or equal to -1, but less than or equal to 1, if not it returns NaN (not a number). A positive return value represents a counterclockwise angle, else a negative value represent a clockwise angle
Math.Atan(A)	**Math.Atan** Returns the angle whose tangent is the specified number (A). If value passed is equal to negative infinity return value is -1.5707963267949, else if value is positive infinity return value is 1.5707963267949. A positive return value represents a counterclockwise angle, else a negative value represent a clockwise angle
Math.Atan2(A,B)	**Math.Atan2** Returns the angle whose tangent is the quotient of two specified numbers, A and B, where A and B are coordinates of a point. The return value is the angle in the Cartesian plane formed by the x-axis, and a vector starting from the origin (0,0) and terminating at the point (A,B).
Math.BigMul(A,B)	**Math.BigMul** Multiples A and B together, and returns the full product
Math.Ceiling(A)	**Math.Ceiling** Rounds A upward to nearest integer value
Math.Cos(A)	**Math.Cos** Returns the cosine of the specified angle (A)
Math.DivRem(A,B, out C)	**Math.DivRem** Calculates a result obtained by dividing one quantity(A) by another(B), and returns the remainder in an output parameter
Math.Exp(A)	**Math.Exp** Returns e (2.71828) raised to the specified power (A)
Math.Floor(A)	**Math.Floor**

	Rounds A downward to nearest integer value
Math.IEEERemainder(A,B)	**Math.IEEERemainder** Returns the remainder of the division of A/B
Math.Log(A,B)	**Math.Log** Returns the logarithm of a specified number (A) in a specified base (B)
Math.Log10(A)	**Math.Log** Returns the logarithm of a specified number (A) in the base 10
Math.Max(A,B)	**Math.Max** Can be used to find the highest value of A and B
Math.Min(A,B)	**Math.Min** Can be used to find the lowest value of A and B
Math.Pow(A,B)	**Math.Pow** Returns a specified number (A) raised to the specified power (B)
Math.Round(A)	**Math.Round** Rounds a number to the nearest whole number
Math.Sign(A)	**Math.Sign** Returns an integer that indicates the sign of a number. Returns -1 if value of A is less than 0. Returns 0 if value is equal to 0, else returns 1 if value is greater than 0
Math.Sin(A)	**Math.Sin** Returns the sine of the specified angle. Value (A) must be passed in radians. To convert the radian return value to degrees multiply it by 180 / Math.PI, else to convert from degrees to radians, multiply value by Math.PI/180
Math.Sqrt(A)	**Math.Sqrt** Returns the square root of A
Math.Tan(A)	**Math.Tan** Returns the tangent of the specified angle. Value (A) must be passed in radians.

Math.Truncate(A)	**Math.Truncate** Rounds a number to the nearest whole number, towards 0

Method Examples

Abs

```
int A = -10, B = -36;
Console.WriteLine(Math.Abs(A));   //10
Console.WriteLine(Math.Abs(B));   //36
```

Acos

```
 double X = Math.PI; //3.14...

 double Y = 0.90;

 Console.WriteLine(Math.Acos(X)); //NaN

 Console.WriteLine(Math.Acos(Y)); // 0.451026811796262

/*To convert the radian return value to degrees multiply it by 180 / Math.PI. */

 Y = Math.Acos(Y)*180/Math.PI;

 Console.WriteLine(Y); // 25.8419327631671
```

Asin

```
   double X = Math.PI; //3.14...

   double Y = 0.90;

   Console.WriteLine(Math.Asin(X)); //NaN

   Console.WriteLine(Math.Asin(Y)); //1.11976951499863

/*To convert the radian return value to degrees multiply it by 180 / Math.PI. */

   Y = Math.Asin(Y)*180/Math.PI;

   Console.WriteLine(Y); //64.1580672368329
```

Atan

```
    double Y = 0.90;

    Console.WriteLine(Math.Atan(Y)); //0.732815101786507

 /*To convert the radian return value to degrees multiply it by 180 / Math.PI. */

    Y = Math.Atan(Y)*180/Math.PI;

    Console.WriteLine(Y); //41.9872124958167
```

BigMul

```
int MaxIntValue  = 2147483647; // the maximum positive value an int can hold

int AnotherMaxIntValue = 2147483647;

long MaxIntsMultiplied = Math.BigMul(MaxIntValue,AnotherMaxIntValue);

Console.WriteLine(MaxIntsMultiplied); //4611686014132420609
```

Ceiling

```
  double X = 23.48;

  double Y = Math.Ceiling(X);

  Console.WriteLine(Y); //24
```

Cos

```
    double X = 90;

    double Y = Math.Cos(X);

    Console.WriteLine(Y); //-0.44807361612917
```

DivRem

```
    int X = 20;

    int Y = 6;

    int Z = 2;

    int A = Math.DivRem(X,Y,out Z);
```


```
    Console.WriteLine(A); //3
```

Exp

```
    double X = 2;

    double A = Math.Exp(X);

    Console.WriteLine(A); //7.38905609893065
```

Floor

```
    double X = 2.89;

    double Y = Math.Floor(X);

    Console.WriteLine(Y); //2
```

IEEERemainder

```
    double X = 20;

    double Y = 6;

    double Z = Math.IEEERemainder(X,Y);

    Console.WriteLine(Z); //2
```

Max

```
int A = 10, B = 20;
Math.Max(A,B); //20
```

Min

```
int A = 10, B = 20;
Math.Min(A,B); //10
```

Pow

```
    double X = 2;

    double Y = 6;

    double Z = Math.Pow(X,Y);

    Console.WriteLine(Z); //64
```


Round

```
double A = 10.99;
Console.WriteLine(Math.Round(A));     //11
Console.WriteLine(Math.Round(36.4));  //36
```

Sign

```
    double X = 2;

    int Y = Math.Sign(X);

    Console.WriteLine(Y); //1
```

Sqrt

```
int A = 10, B = 36;
Console.WriteLine(Math.Sqrt(A));  //3.16227766016838
Console.WriteLine(Math.Sqrt(B));  //6
```

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations on variables and values.

Symbol	Name & Meaning	Example

+	**Addition**
	Adds numbers or variables containing numbers together
		`5 + 5 //10`
		`int x = 10, y = 20;`
		`x + y //30`

-	**Subtraction**
	Subtracts numbers or variables containing numbers together
		`5 - 5 //0`
		`int x = 10, y = 20;`
		`x - y //-10`

*	**Multiplication**
	Multiplies numbers or variables containing numbers together
		`5 * 5 //25`
		`int x = 10, y = 20;`
		`x * y //200`

/	**Division**
	Divides numbers or variables containing numbers
		`10 / 5 //2`
		`int x = 36, y = 6;`
		`x / y //6`

%	**Modulus**
	Returns the division remainder of numbers or variables containing numbers
		`10 % 5 //0`
		`int x = 20, y = 6;`
		`x % y //2`

++	**Increment** Increases the value of a variable by 1	`int x = 20;` `x++; //21`
--	**Decrement** Decreases the value of a variable by 1	`int x = 20;` `x--; //19`

Assignment Operators

Assignment operators are used to assign values to variables.

Symbol	Example
=	`int x = 10; //10` `int y = 20; //20`
+=	`int x = 10;` `x += 5; //15`
-=	`int x = 10;` `x -= 5; //5`
*=	`int x = 10;` `x *= 5; //50`
/=	`int x = 10;` `x /= 5; //2`

%=	

```
int x = 10;
x %= 5; //0
```

Below are some more advanced assignment operators, to understand them you first must understand bitwise.

Bitwise & Bitwise Operators

Bitwise is a level of operation that involves working with individual bits (the smallest units of data in a computer). Each bit has a binary value of 0 or 1. Bits are usually executed in bit multiples called bytes, with most programming languages manipulating groups of 8, 16 or 32 bits.

Bits can be complex for humans to understand, but are far easier to computers, as each bit (aka binary digit) can be represented by an electrical signal which is either on (1) or off (0).

See the below binary conversion table. The binary example below (11010) is equal to 26 in the decimal system (16+8+2=26).

128	64	32	16	8	4	2	1
			1	1	0	1	0

Here is another example:

00001010 is the equivalent of 10 in the decimal system.

128	64	32	16	8	4	2	1
0	0	0	0	1	0	1	0

11110 would be the equivalent of 30.

Note: With binary notation you can perform some math operations very quickly:

- To half any number - simply move the digits 1 place to the right. Ei 11110 (30) -> 1111 (15)
- To double a number - simply add a zero on the end (shifts digits to the left). Ei 11010(26) -> 110100 (52)

Bitwise operators are useful to perform bit by bit operations.

Symbol	Name & Meaning	Example

&	**Bitwise AND**
	This compares each individual bit of the first operand with the corresponding bit of the second operand. If both bits are 1, then the result bit will be 1 otherwise it will be 0
		`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`Console.WriteLine(a & b); //(00000000) equal to 0`
		`Console.WriteLine(a); //(00001010) equal to 10`

&=	The &= operator compares the bits of the operand on its right to the operand on its left, and assigns the result to the operand on its left	`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`a &= b; //assigns result to a`
		`Console.WriteLine(a); //(00000000) equal to 0`

|	**Bitwise OR**
	This compares each individual bit of the first operand with the corresponding bit of the second operand. If either of the bits is 1, then the result bit will be 1 otherwise the result will be 0
		`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`Console.WriteLine(a | b); //(00011110) equal to 30`
		`Console.WriteLine(a); //(00001010) equal to 10`

|=	The |= operator compares the bits of the operand on its right to the operand on its left, and assigns the result to the operand on its left	`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`a |= b;`
		`Console.WriteLine(a); //(00011110) equal to 30`

^	**Bitwise Exclusive OR (XOR)**
	It compares each individual bit of the first operand with the corresponding bit of the second operand. If one of the bits is 0 and the other is 1, then the result bit will be 1 otherwise the result will be 0
		`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`Console.WriteLine(a ^ b); //(00011110) equal to 30`
		`Console.WriteLine(a); //(00001010) equal to 10`

^=	The ^= operator compares the bits of the operand on its right to the operand on its left, and assigns the result to the operand on its left	`int a = 10; //(00001010)`
		`int b = 20; //(00010100)`
		`a ^= b;`
		`Console.WriteLine(a); //(00011110) equal to 30`

<<	**Bitwise Shift Left**
	This shifts bits to the left by the amount specified on the right.
	Ei shift of 1, moves the bits left 1 position each (end will have a zeo
		`int b = 20; //(00010100)`
		`Console.WriteLine(b << 2 ); //(1010000) equal to 80`
		`(2 is equivalent of multiplying by 4)`

	added to it). This is equivalent to multiplication by 2	`int b = 20; //(00010100)` `Console.WriteLine(b << 1 ); //(101000) equal to 40`

<<=	The <<= operator shifts the bits of the operand on its left by the amount specified on the right, and assigns the result to the operand on its left	`int b = 20; //(00010100)`
		`b <<= 1;`
		`Console.WriteLine(b); //(101000) equal to 40`

>>	**Bitwise Shift Right** This shifts bits to the right by the amount specified on the right. Ei shift of 1, moves the bits right 1 position each. This is equivalent to division by 2	`int b = 20; //(00010100)` `Console.WriteLine(b >> 1 ); //(00001010) equal to 10`

>>=	The >>= operator shifts the bits of the operand on its left by the amount specified on the right, and assigns the result to the operand on its left	`int b = 20; //(00010100)`
		`b >>= 1;`
		`Console.WriteLine(b); //(00001010) equal to 10`

The below operand is used to flip bits, ei 0 becomes 1 and vis vera.

Symbol	Name & Meaning	Example

~	**Bitwise Complement**
	This operates on only one operand, and it will invert each bit of the operand (ei it will change 1 to 0 and 0 to 1)
		`int b = 20; //(00010100)`
		`b =~(b);`
		`Console.WriteLine(b); // -21 (11101011)`