Tutorial - Operators

In this tutorial we're going do discuss operators. What they are, and how we use them.

What Are Operators

Operators, or **Arithmetic Operators**, are one way that we can manipulate variables. You can use them to do basic math, as well as one or two other unique functions. The **Arithmetic Operators** are:

Operator	Name	Description
+	Addition Operator	Adds two values together
-	Subtraction Operator	Subtracts one value from another
*****	Multiplication Operator	Multiplies two values together
/	Division Operator	Divides one value by another value
%	Modulo Operator	Gives the remainder after dividing two whole numbers, or integers
++	Increment Operator	Increases a whole number, or integer, by 1 value
--	Decrement Operator	Decreases a whole number, or integer, by 1 value

Using Operators

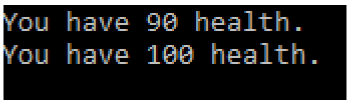
Create a new project in Visual Studio. Remember it's: **File > New > Project > Console App (.NET Framework) …** I'm calling my project **UsingOperators**.

Modify your **Main** function as follows:

```
        ~~static void Main(string[] args)~~
        static void Main()
        {
            int health = 90;
            Console.WriteLine("You have " + health + " health.");

            health = 90 + 10;
            Console.WriteLine("You have " + health + " health.");
            Console.ReadLine();
        }
```

Run the program by pressing **Start.** You should see the following:



```
You have 90 health.
You have 100 health.
```

We set our variable **health** to 90 initially when we declared it, and then we printed that value to the **Console**.

Then we changed the value of the variable to **90 + 10**, and we print the new value, which gives us 100.

You might have expected the value to be 190, but when we change the value of **health**, we are giving it an entirely new value based on our maths. This ignores the variables previous value of **90.**

Now change your code as follows:

```
        static void Main()
        {
            int health = 90;
            int healthBonus = 10;
            Console.WriteLine("You have " + health + " health.");

            health = health + healthBonus;
            Console.WriteLine("You have " + health + " health.");
            Console.ReadLine();
        }
```

Test your program and see what values you get.

You get the same answer as before. This is because when you type a variable on the **right side** of the **=** sign, it reads the value stored inside the variable.

In this case, we are setting **health** to be it's own value**,** which is **90**, plus the value stored inside **healthBonus**, which is **10**.

Wouldn't it be nice if there was an easier way to increase a variables value? Try the following:

```
        static void Main()
        {
            int health = 90;
            int healthBonus = 10;
            Console.WriteLine("You have " + health + " health.");

            health += healthBonus;
            Console.WriteLine("You have " + health + " health.");
            Console.ReadLine();
        }
```

Run your program. You'll get exactly the same values as before.

When we wrote **+=**, what we said was, "**Keep the value of the variable I've already got, then add the value of healthBonus to it.**"
health += healthBonus is exactly the same as ***health = health + healthBonus;***

You can use this shorthand for most of the arithmetic operators. These new operators are called the **Assignment Operators**, because they assign a new value to the variable on their left.

Operator	Name	Example
=	Assignment Operator	**C = A + B** gives the value of **A + B** to **C**
+=	Addition And Assignment	**C += A** is the same as **C = C + A**
-=	Subtract And Assignment	C -= A is the same as C = C - A
*=	Multiply And Assignment	C *= A is the same as C = C * A
/=	Divide And Assignment	C /= A is the same as C = C / A
%=	Modulus and Assignment	C %= A is the same as C = C % A

There are a few other **Assignment Operators** which you don't need to know for the sake of this tutorial.

Other Arithmetic Operators

Let's quickly look at the other arithmetic operators. Delete the previous code from your function and add the following:

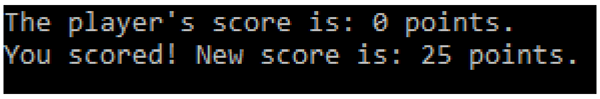
```
        static void Main()
        {
            int score = 0;
            int pointValue = 5;
            int combo = 5;

            // Start of the game
            Console.WriteLine("The player's score is: " + score + " points.");

            score += pointValue * combo;

            Console.WriteLine("You scored! New score is: " + score + " points.");
            Console.ReadLine();
        }
```

Run the program and see what values you get.



```
The player's score is: 0 points.
You scored! New score is: 25 points.
```

We create 3 variables: **score, pointValue,** and **combo**, then print their starting values. Next, we multiply the variable **pointValue** by the variable **combo** and **add** the result of that to the variable **score**. The result is **25.**

Now try the following:

```
        static void Main()
        {
            int score = 0;
            int pointValue = 5;
            int comboMultiplier = 5;

            // Start of the game
            Console.WriteLine("The player's score is: " + score + " points.");

            combo ++;
            score += pointValue * combo;
            Console.WriteLine("You scored! New score is: " + score + " points.");

            combo ++;
            score += pointValue * combo;
            Console.WriteLine("You scored! New score is: " + score + " points.");
            Console.ReadLine();
        }
```

Run the program and see the result. It's 0, 30 then 65.

Writing **comboMultiplier++** increases the **comboMultiplier** variable by 1. Likewise, if we write **comboMultiplier--**, it would decreate the variable by 1.

Now try this:

```
        static void Main()
        {
            int myVariable = 5;
            Console.WriteLine("Value of variable before: " + myVariable);

            myVariable /= 3;
            Console.WriteLine("Value of variable after: " + myVariable);

            Console.ReadLine();
        }
```

Run the program again. You should see **5,** then **1.** You might be surprised if you forgot that **int** variables cannot have decimal values – the decimal is chopped off.

Reviewing Floats and Decimals

To illustrate, try this:

```
        static void Main()
        {
            float myVariable = 5;
            Console.WriteLine("Value of variable before: " + myVariable);

            myVariable /= 3;
            Console.WriteLine("Value of variable after: " + myVariable);

            Console.ReadLine();
        }
```

Run the program. Now you should get this **5,** then **1.666667.** That's because we changed our variable type for **myVariable** from **int** to **float,** and floats can have decimal values.

Try this:

```
        static void Main()
        {
            float myVariable = 5;
            Console.WriteLine("Value of variable before: " + myVariable);

            myVariable = 5 / 3;
            Console.WriteLine("Value of variable after: " + myVariable);

            Console.ReadLine();
        }
```

Run the program. You'll see the same result from earlier: **5,** then **1.**

You might be wondering why, since **myVariable** is a float. This is a tricky situation that can catch even an experienced programmer out if they're not careful.

The problem is, when you type numbers without decimals such as **5 / 3**, the numbers are considered as **integers.** The program then assumes you don't want the answer of **5 / 3** to have decimals either.

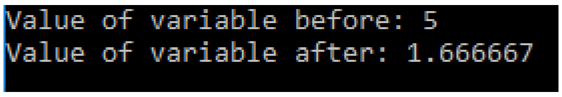
By declaring **myVariable** as a float, it means it **can** take numbers with decimal values, but your equations must still contain either a **float** variable or a number with a **decimal value** to get a decimal in your answer.

The following example will make this concept a bit clearer:

```
        static void Main()
        {
            float myVariable = 5;
            Console.WriteLine("Value of variable before: " + myVariable);

            myVariable = 5.0f / 3;
            Console.WriteLine("Value of variable after: " + myVariable);
            Console.ReadLine();
        }
```

Run the program. You will get a number with a decimal value again.



```
Value of variable before: 5
Value of variable after: 1.666667
```

This is because we explicitly told C# **5.0f** is a **float** so that our answer will have decimal places. We have to put the **f** for float on the end to let the program know that 5.0 is a floating number.

Modulo

Modulo is something you may never need to use. It's uses are fairly specialised, however, it can come in handy in certain cases. One of them being, it's a quick way to work out if a number is odd or even.

To demonstrate:

```
        static void Main()
         {
            int myVariable = 4;
            ~~Console.WriteLine("Value of variable before: " + myVariable);~~
            myVariable = myVariable % 2;
            Console.WriteLine("The remainder is: " + myVariable);

            Console.ReadLine();
         }
```

Run the program. The result should be 0.

We've written the equation the long way to make the process clearer. Modulo **%** is a special **Arithmetic Operator** that tells you what number is left after a division.

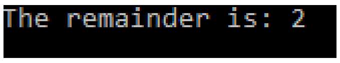
With the equation ***myVariable % 2***, the first thing C# does is divide **myVariable** by 2, so you could re-write it as **myVariable / 2.** In this case, that would be **4 / 2** which is 0. But modulo has an extra step after the division. Some numbers do not divide evenly into each other, and so you get a remainder. Modulo will tell you what that remainder is. In this case, the answer to 4 % 0 is 0, because 2 divides evenly into 4 and so there is no remainder.

For another example, try this:

```
        static void Main()
        {
            int myVariable = 5;
            myVariable = myVariable % 3;
            Console.WriteLine("Value of variable after: " + myVariable);

            Console.ReadLine();
        }
```

Run the program. You should see this:



```
The remainder is: 2
```

The answer of **5 / 3** is **1.** However, that isn't what we get. Because modulo has the extra step which then tells us how much is left over. And after you have divided 5 by 3, you get 2 left over. So **5 % 3** is **2.**

Let's try another example:

```
        static void Main()
        {
            int myVariable = 3;
            myVariable = myVariable % 2;
            Console.WriteLine("The remainder is: " + myVariable);

            Console.ReadLine();
        }
```

Run the program and see what you get. The remainder is 1.

3 / 2 is 1, with 1 left over. This is because 3 is an odd number. Whenever you use **modulo %** on an even number, the result will be **0**, and whenever you use **modulo %** on an odd number, the result will be 1.

You won't use it often, but **modulo** can be useful.

Conclusion

That's all we're going to look at with variables. It gives you a pretty thorough overview of the fundamentals. I recommend looking at the different types of operators and experimenting with different combinations and printing out the results. This will help you to get more familiar with them all.