Mr. Bellavia

LESSON 04 - DECIMAL VALUES, DOUBLE VARIABLES & TRUNCATION

In this lesson we will discuss how to use double variables to store decimal values. We will also examine how to avoid truncation when using decimal numbers and how to output decimal numbers with a specific number of decimal places.

Sections:

I. DECIMAL VALUES & DOUBLE VARIABLES:

We have seen how we can create variables to hold numbers using **integer variables**. However, if you try to store a **decimal value** into an **integer variable** then then the compiler will give you a **build error.** For example:

```
Console.WriteLine(1.5);
int z = 1.2;
Console.WriteLine(z);
```

Output will give you a **build error!**

Outputting a decimal value with **Console.WriteLine()** works ok, however, the second line of code above will give you a build error because you cannot store **decimal values** into an **integer variable.** The decimal portion **'.2'** is what is causing the error. If we wish for a variable to **hold a decimal** then a simple way to achieve this is to create a **'double'** variable, for example:

```
Console.WriteLine(1.5);
double z = 1.2;
Console.WriteLine(z);
```

Output will be:

```
1.5
1.2
```

A **'double'** variable **can** store **decimal values,** but **'int'** variables **cannot.**

Note: We have now seen 3 different **data types: string, int, & double**

II. AVOIDING TRUNCATION (CASTING):

There are cases when you may be using integer variables and end up with an incorrect decimal value. For example:

```
int x = 5;
int y = 4;
Console.WriteLine(x / y);
```

Output will be:

```
1
```

The answer should have been **'1.25',** but instead we got **'1'.** This is an example of **truncation** where the decimal is being completely discarded. The reason this happened is because C# assumes you are only using integers since both variables 'x' & 'y' are integer variables. Hence, C# will remove the decimal from your calculation. Obviously, this is not good!

Usually, truncation will occur when you are dividing integers that should result as a decimal number. We can try to use a **double** variable when we want to keep the decimal when dividing integers, but we will still get truncation. For example:

```
int x = 5;
int y = 4;
double z = x / y;
Console.WriteLine(z);
```

Output will be:

```
1
```

Truncation occurred again!

Our program calculated the wrong answer because the value of **'x / y'** is calculated **before** it is stored in **'z'.**

How do we fix this? We can sort of 'cheat' the calculation by **multiplying** 'x' by **1.0** before the division occurs:

```
int x = 5;
int y = 4;
double z = 1.0 * x / y;
Console.WriteLine(z);
```

Output will be:

```
1.25
```

Due to BEDMAS the **1.0** will get multiplied by the 'x' before the division by 'y' occurs. This works, but a more appropriate way to avoid truncation is to convert your **data type** using a **cast.** For example:

```
int x = 5;
int y = 4;
double z = (double)x / y;
Console.WriteLine(z);
```

Output will be:

```
1.25
```

The above code demonstrates **casting.** We use the keyword **double** inside **brackets ()** and put it **in front of the variable** being divided. This will convert (cast) the variable **'x'** from an **'int'** to a **'double'** before the division occurs.

Let's look at a more complex example:

```
int x = 5;
int y = 4;
int a = 3;
int b = 7;
int s = 12;
int t = 66;
double z = x / y + 5 * (b / a) + 3 / (s / t);
Console.WriteLine(z);
```

Output will be a **build error!**

You get a build error because **'s / t'** equates to **'0'** in this case and you **cannot divide** by **'3'** by **'0'.**

If you wanted to get the proper answer without any build errors, you will need to cast every variable involved in division. For example:

```
double z = (double)x / y + 5 * ((double)b / a) + 3 / ((double)s / t);
```

Output will be:

```
29.416666666666668
```

III. SET DECIMAL PLACES DURING OUTPUT:

When we output to the screen using **Console.WriteLine()**, it is sometimes necessary to indicate the **number of decimal places** we want displayed for our decimal numbers. A perfect example is outputting money. Money (currency) always has 2 decimal places.

For example, consider the following:

```
double money = 5.43 * 2.4;
Console.WriteLine("You have $" + money);
```

Output will be:

```
You have $13.031999999999998
```

Obviously, we **cannot** have .0319999… cents. To indicate 2 decimal places for **Console.WriteLine()** we can do the following:

```
double money = 5.43 * 2.4;
Console.WriteLine("You have $" + money.ToString("0.00"));
```

Output will be:

```
You have $13.03
```

Now our value for money has 2 decimal places, and it is rounded properly! Notice how we attached a built-in function to our **'money'** variable called **.ToString().** Also notice that we have **"0.00"** **inside** the **.ToString()** function. This indicates that we want to format our decimal number to **two decimal places.** If we wanted, for example, 6 decimals then we would use "0.000000". You will be introduced to many built-in functions as the course progresses.