Mr. Bellavia

LESSON 02 – METHODS WITH PARAMETERS

In this lesson we will examine how to create methods with parameters in which we can pass arguments to our methods to make them more flexible and useful.

Sections:

I. ARGUMENTS & PARAMETERS:

Our methods so far have been static. We call them, they execute, and then the program continues. However, it is more useful if we can call these methods and have them work on some given data. For example, let's say we wish to calculate the circumference of a circle for a given radius. Our formula would be:

$$C = 2\pi r$$

Therefore, if our radius (r) is set to 5, then we would calculate:

C = 2 * 3.14 * **5**
= 31.4

We can create a method in C# to calculate the circumference of a circle based on any given radius, for example:

```
void CalculateCircumference(double radius)
{
    double c = 2 * Math.PI * radius;
    Console.Write("Circumference of a circle with a radius of " + radius + " is: ");
    Console.Write(c.ToString("0.00"));
}
CalculateCircumference(5);
```

Sample **Output:**

```
Circumference of a circle with a radius of 5 is: 31.42
```

Notice our **method definition.** We still use the keyword **void** with a **method name** of <u>our choice</u>, but inside our **brackets ( )** we have a **parameter** name **'radius'** of type **double:**

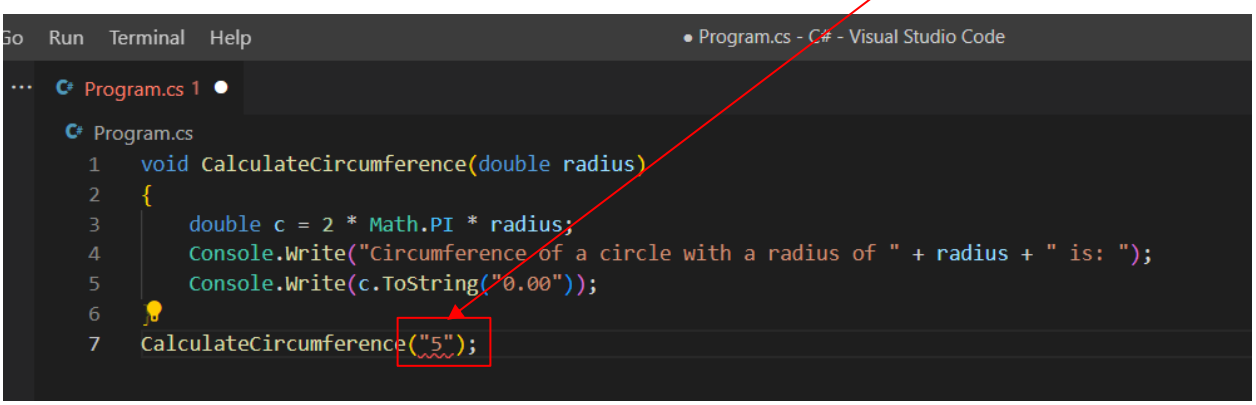
```
void CalculateCircumference(double radius)
```

This indicates that when we **call** this method, we must **pass** a double value to it. The value that we **pass** into the method is called an **argument.** For example:

```
CalculateCircumference(5);
```

The above line of code will call the **ClalculateCircumference()** method and pass in the **argument** value of **'5'** to the method. The method then receives the value of **'5'** and stores it in the **parameter** variable **'radius'.**

We can pass any value we like as an argument, but it must be of the same data type indicated by the method definition. In the example above, the parameter 'radius' is defined as a double, so we must pass a valid double value. If we do not, we will get a **syntax error,** for example:



In this example we tried to pass in the string value of "5" as our argument, which will result in a syntax error since the method is expecting a double value.

Note: If we were to pass in the **char** value of '5', we would **not** get a syntax error. Instead, the ASCII value of character '5' (which equals 53) would be passed into the method and our radius would calculate to '333.01'. Obviously, this in incorrect so we need to be careful. Try it out!

```
CalculateCircumference('5');
```

II. MULTIPLE PARAMETERS:

We are not limited to just one parameter; we can pass in **multiple parameters** of **any type.** For example:

```
void PrintSubTotal(string itemDescription, int qty, double pricePerItem)
{
    double subTotal = qty * pricePerItem;
    Console.WriteLine("Item: " + itemDescription);
    Console.WriteLine("Quantity: " + qty);
    Console.WriteLine("Price per item: " + pricePerItem);
    Console.WriteLine("SUB TOTAL: " + subTotal.ToString("0.00"));
}
PrintSubTotal("Apples", 5, 0.99);
```

Sample **Output:**

```
Item: Apples
Quantity: 5
Price per item: 0.99
SUB TOTAL: 4.95
```

Notice how our **PrintSubTotal()** method has **3 parameters** (itemDescription, qty, and pricePerItem) each of a different **type** (string, int, and double) separated by a **comma ‘,’:**

```
void PrintSubTotal(string itemDescription, int qty, double pricePerItem)
```

Similarly, when calling the method, our arguments consist of values of the proper data type each separated by a **comma ‘,’:**

```
PrintSubTotal("Apples", 5, 0.99);
```

III. INVALID METHOD CALLS:

When calling a method, the **arguments** you pass must **match** the **parameters** defined for that method. For example, consider the method we created in the previous section:

```
void PrintSubTotal(string itemDescription, int qty, double pricePerItem)
```

The following demonstrates **invalid method calls,** and will give you **syntax errors:**

```
PrintSubTotal("Apples", 5);
```

```
PrintSubTotal(5, 0.99, "Apples");
```

```
PrintSubTotal("Apples", "5", "0.99");
```

```
PrintSubTotal();
```

From the above, the first method call does not contain the proper number of arguments, the second method call has the arguments in the wrong order, the third method's last two arguments are invalid data types based on the defined parameters (i.e., should be int and double, not string and string), and the fourth method call does not contain any arguments.

Remember: If you define a method with no parameters then you just use an empty set of **brackets ( ).** For example, consider the following method:

```
void WelcomeMessage()
```

To **call** this method we would have:

```
WelcomeMessage();
```