Lesson 17 - Methods II

Recall,

> Methods can have a list of parameters to work with. They are variables that accept values during method call, and use them to perform any function. The method `body` uses that value and then discards it when the method call is complete.

```
int Square(int num)
{
    int squaredNum = num * num;
    return squaredNum;
}

int num1 = 3;
int num2 = 4;

// sum of squares of the two numbers
int result = Square(num1) + Square(num2);
Console.WriteLine(result);
```

Optional Parameters

When defining a method, you can specify a **default value** for optional parameters. These must be defined *after* required parameters. If the method is called without specifying valued for their values, then the default values for optional parameters is used.

```
int Power(int num, int exponent = 2)
{
    int result = 1;
    for (int i = 0; i < exponent; i++)
        result *= num;

    return result;
```

```
}

Console.WriteLine(Power(2));
Console.WriteLine(Power(2, 3));
```

```
OUTPUT:
4
8
```

> Note: Method *parameters* are also sometimes called method **arguments**.

Named Parameters

When we `declare` and `call` a method, in both cases the order of the parameters should be the same. What would happen if we have a lot of parameters and forget or mismatch the order in which to write then in call?

It can become very cumbersome to keep track of slightly complex methods, so in those cases we can use `named` parameters, and provide parameters values *in any order*.

```
Console.WriteLine(Power(num: 2, exponent: 3));
Console.WriteLine(Power(exponent: 3, num: 2));
```

```
OUTPUT:
8
8
```

Passing Parameters

There are three ways to pass arguments to a method.

1. By **value**
2. By **reference**
3. As **output**

By value

Copies the argument **value** into the method's parameter variable. Changes made to the parameter inside the method do not have any effect on the original variable. This is used by **default** to pass arguments. So, the method operates on the *value*, not on the actual *variable*.

```
int ByValueDemo(int num1, int num2)
{
    num1 += 1;
    num2 += 1;
    return num1 + num2;
}

int num1 = 1;
int num2 = 2;

Console.WriteLine("Initial Values:");
Console.WriteLine($"Num1: {num1}");
Console.WriteLine($"Num2: {num2}\n");

Console.WriteLine($"Demo: {ByValueDemo(num1, num2)}");
Console.WriteLine($"Num1: {num1}");
Console.WriteLine($"Num2: {num2}");
```

```
OUTPUT
Initial Values:
Num1: 1
Num2: 2

Demo: 5
Num1: 1
Num2: 2
```

By reference

Copies the argument's **memory address** into the method's parameter variable. Since using the memory address we can access the actual variable, any changes made to the parameter inside the method affect the original variable. To pass by reference, the `ref` keyword is used in both **call** and **declaration**.

```
int ByReferenceDemo(ref int num1, ref int num2)
{
    num1 += 1;
    num2 += 1;
    return num1 + num2;
}

int num1 = 1;
int num2 = 2;
```

```
Console.WriteLine("Initial Values:");
Console.WriteLine($"Num1: {num1}");
Console.WriteLine($"Num2: {num2}\n");

Console.WriteLine($"Demo: {ByReferenceDemo(ref num1, ref num2)}");
Console.WriteLine($"Num1: {num1}");
Console.WriteLine($"Num2: {num2}");
```

```
OUTPUT
Initial Values:
Num1: 1
Num2: 2

Demo: 5
Num1: 2
Num2: 3
```

As output

These are like reference parameter, but they transfer data *out* of the method rather than in. They are defined using the `out` keyword. This is useful when you want to return multiple values from a method. `out` must be used both when declaring and calling the method.

```
void AsOutputDemo(int number, out int square, out int cube)
{
    square = number * number;
    cube = number * number * number;
}

int x = 2;
int x2;
int x3;

AsOutputDemo(x, out x2, out x3);

Console.WriteLine(x);
Console.WriteLine(x2);
Console.WriteLine(x3);
```

```
OUTPUT:
2
```

```
4
8
```