Lesson 18 - Methods III

Overloading

Method **overloading** is when multiple methods have the **same name**, but **different parameters**.

For example, you might have a `Print` method that outputs its parameter to the console window:

```
public static void Print(int a)
{
        Console.WriteLine("Value: " + a);
}
```

This method accepts an integer **argument** only. Overloading it will make it available for other types, such as **double**.

```
public static void Print(double a)
{
        Console.WriteLine("Value: " + a);
}
```

Now the same `Print` method name will work for both integers and doubles.

> When overloading, the definition of the methods must differ. This means that either the type of parameters, or the number of parameters, or both must differ accross overloaded methods.

When we call an overloaded method, the definition relevant to the parameters provided will be used.

Another overload of `Print` showcasing different number of parameters:

```
public static void Print(string label, double a)
{
```

```
        Console.WriteLine(label + a);
}
```

> you CANNOT overload method declarations that differ only by return type. If you try to do that, it will raise an error.

Recursion

A method which calls itself is a recursive method. There are some classic problems that can be solved using recursion, such as taking the factorial of a number. A factorial is given as follows:

```
1! = 1
2! = 2 x 1
3! = 3 x 2 x 1
4! = 4 x 3 x 2 x 1
```

The above equations can be simplified as

```
1! = 1
2! = 2 x 1!
3! = 3 x 2!
4! = 4 x 3!
```

and when you generalize, it becomes

```
n! = n x (n-1)!
```

and we can use this recursion to create a recursive method.

```
int Factorial(int n)
{
        if (n == 1)
                return 1;

        return n * Factorial(n-1);
}
```

So here we need to keep in mind that for a recursive method to work, there must be atleast 2 states, one must be the recursive state, and the other must be the exit state. This is usually defined by a condition, so as long as the condition is in one state i.e.,

true or false, the recursion continues, as soon as the state is switched, we hit the exit state and stop.

Sum example without recursion

```
int Sum(int num)
{
        int sum = 0;
        for (int i = 1; i <= num; i++)
                sum += num

        return sum;
}
```

Sum using recursion

```
int Sum(int num)
{
        if (num == 1)
                return 1;

        return num + Sum(num - 1);
}
```