# Lesson 16 - Methods I

A method is a group or `block` of statements that performs a particular task. C# has some built-in methods, like some array methods we discussed in last lecture,

> ```
> int[] numbers = { 1, 2, 3, 4 };
>
> int i = 0;
> while (i < numbers.Length)  // length: 4, indexes: 0, 1, 2, 3
> {
>         Console.WriteLine(numbers[i++]);
> }
>
> Console.WriteLine($"Sum: {numbers.Sum()}");
> ```

In addition to these built in methods, you can define your own as well. Advantages of using methods are:

1. Reusable code.
2. Easy to test.
3. Modifications do not affect calling program.
4. One method can accept many different inputs.

## Declaring Methods

To use a method, you must first `declare` it, then `call` it. The general syntax of a method declaration is as follow,

```
returnType MethodName(paramType paramName)
{
        // code to run when method is called
}
```

**returnType:** data type of the result value of a method. It can be any valid `C#` type, like `int`, `bool`, `string` etc, and for special cases when a method does not have any result or `return` value, it has a returnType of `void`.

### Example

```
void PrintHello()
{
    Console.WriteLine("Hello");
}

PrintHello();
```

```
OUTPUT: Hello
```

## Method Parameters

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);
```

```
OUTPUT: 25
```

The `return` statement is used to send a result back to the method `call` so that it can be used where the method is called, such as assigned to a variable or printed to the console.

## Multiple Parameters

You can have as many parameters as needed for a method by separating them by commas in the definition.

```
int Sum(int num1, int num2)
{
    return num1 + num2;
}
```

```csharp
int sum = Sum(5, 8);
```

## Interesting Examples

```csharp
int[] InitializeIntArray(int sizeOfArray)
{
    Console.WriteLine($"Please enter {sizeOfArray} values.");

    int[] array = new int[sizeOfArray];
    for (int i = 0; i < array.Length; i++)
        array[i] = Convert.ToInt32(Console.ReadLine());

    return array;
}

void PrintIntArray(int[] array)
{
    Console.WriteLine();

    foreach (var item in array)
        Console.WriteLine(item);

    Console.WriteLine();
}

int[] numbers = InitializeIntArray(5);
PrintIntArray(numbers);

numbers = InitializeIntArray(7);
PrintIntArray(numbers);
```

```csharp
Console.WriteLine("Welcome to the program.");

void MessageConstructor(string messageBody, string authorName)
{
    string message = $"{messageBody} by {authorName}";
    Console.WriteLine(message);
}

// Message by Talha
MessageConstructor("Hello World!", "Talha");

// Message by Amad
MessageConstructor("Hello Moon!", "Amad");

Console.WriteLine("End of program.");
```