The Fundamentals of C#

Methods, arguments, parameters, and **Main**

Objectives:

- What is a method?
- Methods that do not return values (void keyword).
- Methods that return values. (return keyword)
- **Main** method

What is a method?

- A C# method is a collection of statements that are grouped together to perform an operation/task.
- When you call the **Console.WriteLine** method, for example, the system actually executes several statements in order to write your message to the console.
- In C# there are two types of methods:
  - Methods that do not return a value (void keyword)
  - Methods that return a value (return keyword)

What is a method? (continued)

- Every method has a **signature**, which distinguishes one method from another. The **signature** is made up of the following:
  - The **accessibility** (will be covered in a later lesson)
  - Optional **modifiers** (will be covered in a later lesson)
  - The **return type**
  - The **name** of the method
  - Any method **parameters**

- A method also has a **body**, which contains the statements

Methods (void)

- Syntax of a method that does not return a value.

required, will be explained later

No return

Method name

Method parameters (optional)

```
static void Greetings(string name)
{
    Console.WriteLine("Hi: " + name);
}
```

method body

Method definition

Method call (void)

- When we **call** a method we use the **name** of the method.
- If a method takes **parameters** we have to provide **arguments**. Arguments are the information we give to a method.
- Arguments are matched to parameters by position.
- Arguments and parameters **MUST HAVE** the same type
- Arguments **DO NOT** have to have the same name as parameters

Method call and definition (void)

Method call

```
string name = "Carl";
Greetings(name);
```

argument

Method definition

parameter

```
void Greetings(string n)
{
    Console.WriteLine("Hi: " + n);
}
```

name is the **argument**.
n is the **parameter** for our **Greetings** method
Note how the argument matches the parameter both in position and type.
In this case the type of the argument and parameter are both **string**

Methods (void) - example

```
public class Example
{
    public static void Main(string[] args)
    {
        Greetings("students");
        Console.WriteLine("Welcome to this C# session!");
        Console.WriteLine("Bye!!");
    }

    static void Greetings(string value)
    {
        Console.WriteLine("Hello: " + value);
    }
}
```

Method call (passing one argument)

method definition (one parameter)

Greetings method only requires one parameter (*value*) of type string.
We only need one argument “students” when we call the method.

Methods (void) – another example

```
public class Example
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Welcome to this C# session!");
        PrintAsterisks(12);
        DisplayNames("Carl", "Ai", "Daniel");
        PrintAsterisks(15);
    }

    static void PrintAsterisks(int num)
    {
        for (int i = 0; i < num; i++)
        {
            Console.Write("*");
        }
        Console.WriteLine();
    }

    static void DisplayNames(string n1, string n2, string n3)
    {
        Console.WriteLine(n1 + ", " + n2 + ", " + n3);
    }
}
```

Output:

Welcome to this C# session!
****
Carl, Ai, Daniel
****

Methods that return a value (return)

- Often you will want to have a method that returns a value. In order to do this, you have to do the following:
  - Specify the **return type** of the method in the **signature**
  - Use the **return** keyword in the **body** of the method
- The **return** keyword not only returns a value, but **immediately** leaves the method
- When calling a method with a **return** value, we can assign the result of that method call to a **variable**

Methods that return a value (return)

- Syntax of a method that returns a value.

required, will be explained later

Return type

Method name

Method parameters (optional)

```
static int Add(int a, int b)
{
    int result = a + b;
    return result;
}
```

method body

Method definition

The return statement terminates execution of the method and specifies the value that the method returns.

Method call and definition (return)

Method call

```
int x = 3, y = 8;
double z;
z = Average(x, y);
```

Method definition

```
static double Average(int a, int b)
{
    double c;
    c = (a + b) / 2d;
    return c;
}
```

x and **y** are the arguments.
a and **b** are parameters for our **Average** method
Note that the arguments match the parameters both in position and type.
In this case the type of the arguments and parameters are **int**
The value of **c** will be returned from the method and assigned to **z**

Methods (return) - example

```
public class Example
{
    public static void Main(string[] args)
    {
        int num1 = 5, num2 = 3;
        int answer = Add(num1, num2);
        Console.WriteLine(answer);
        answer = Add(7, 25);
        Console.WriteLine(answer);
    }

    static int Add(int a, int b)
    {
        int result = a + b;
        return result;
    }
}
```

Value returned (8) is assigned to variable answer

Value returned (32) is assigned to variable answer

Main method

- Every program needs an entry point, a place where the execution of our program starts.
- When working with **console applications** our entry point is the **Main** method
- This method has a specific **signature** that is recognised by the **.NET Runtime**
- You cannot have two methods with this signature, even if they're in different classes. **.NET** won't know which one to call!

Main method (example)

When you start a console application with the **dotnet run** command, the method below will be called and our program execution will start.

```
public class Hello
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello World");
    }
}
```

Putting methods together

Within a method definition, you can call other methods.

You cannot define (declare) a method within another method.

Putting methods together

```
public class Square
{
    public static void Main(string[] args)
    {
        for (int i = 1; i < 17; i++)
        {
            DisplaySquare(i);
        }
    }

    static void DisplaySquare(int number)
    {
        int square = number * number;
        Console.WriteLine("The square of " + number + " is " + square);
    }
}
```

The class Square has two methods

Putting methods together

```
public class Square
{
    static void DisplaySquare(int number)
    {
        int square = number * number;
        Console.WriteLine("The square of " + number + " is " + square);
    }

    public static void Main(string[] args)
    {
        for (int i = 1; i < 17; i++)
        {
            DisplaySquare(i);
        }
    }
}
```

The order in which methods are written makes no difference

Demonstration

- What is a method?
- Methods that do not return values (void keyword).
- Methods that return values. (return keyword)
- **Main** method