Parameters and input

One parameter

We have seen previously, how to define and invoke a Method without Parameters:

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

This is very useful if you want a function that does the same thing every time.

But sometimes, you want a function to do very similar things, but slightly differently every time. For example, we'd like to define a method that prints the double value of a number.

Calculating and returning the double of a number follows the exact same mathematical rules every time. But the number that we'd like to have double of changes every time.

Parameterize it

We can achieve this using a parameterized function:

```
void PrintDouble(int number){
  Console.WriteLine(number*2);
}
```

Above function can be invoked as follows:

```
PrintDouble(3); // 6
PrintDouble(10); // 20
PrintDouble(-2); // -4
```

Defining a parameter

You define a parameter as follows:

```
ReturnType MethodName(ParameterType parameterName){
  // Method Body here
}
```

Invoking a function with a parameter

When you invoke a function with one Parameter, you need to pass an Argument of a matching type to the function:

MethodName(someValue);

E.g:

Abs(-5) // absolute value of a number -5 = 5

If the function requires one argument, but you don't pass one, your code won't compile and show an error:

Abs(); // ERROR: Function requires 1 argument, but is invoked with 0.

And you can also not pass the wrong type of argument:

Abs("Hello"); // ERROR: No overload for Abs(string) exists.

Parameter scope

A parameter can be used within the scope of the function only:

```
void Print(string value){
  Console.WriteLine(value);
}

value = "Hello"; // ERROR: undefined variable `value` (it is only defined within the scope of the method)
```

Parameter scope 2

A parameter can also have the same name as another variable outside its scope:

```
int value = 5;

void Print(string value){
  Console.WriteLine(value);
}

value = 6;

Print("Hello");
```

Passing an argument

The following concept is a very important one and often a topic of confusion for young developers.

When you pass a variable as an argument, its value gets copied to the function. This means, that the original variable will not be affected by any changes within the function:

```
int number = 5;
void Change(int number){
  number = 7;
  Console.WriteLine(number); // 7
}
Change(number);
Console.WriteLine(number); // 5
```

Step by step

- First, the Variable named number is defined and the value 5 is assigned to it
- Then, the function named Change is invoked, passing the value of number (5) as an argument to the function
  - This means, that the value 5 gets copied to the parameter of the function
- When 7 gets assigned to number, this affects the function's parameter only. Not the original variable with the same name
- When printing number within the function Change, it will print the number of the function's parameter, which is 7
- When the function ends..
  - It will pop the parameter **number** and its value of 7 off the stack making room again for the variable **number**
- Console.WriteLine in the Main-procedure also references the variable named number, not the parameter

More parameters

The case for two parameters is not much different from the case with one parameter:

```
void DrawCard(int number){
  // ...
}
```

If you want to have more than one Parameter, you need to specify each Parameter by Type and Name, separated by , (comma):

```
void CreateUser(string firstName, string lastName){
  // ...
}
```

Invoking multiple parameters

When invoking the function, you need to pass arguments with the correct:

- number
- order
- type

CreateUser("John", "Legend");

CreateUser("Doom", "Slayer");

Passed as value

Each argument will be passed as a value to the property at the same position:

- Argument "John" at first position -> Parameter firstName at first position
- Argument "Legend" at second position -> Parameter lastName at second position

Where you can go wrong

A few examples of where you can go wrong:

- Too few arguments are passed into the function
- Too many arguments are passed into the function
- Arguments of the wrong type is passed into the function
- Or arguments are passed into the function in the incorrect order

Goal 1 - First Param

Write a function that takes one string parameter.

Print the parameter to the console.

Invoke the function with the following arguments:

- "Fredrik"
- "Game Programming"
- "Forsbergs"

Goal 2 - Double

Write a function that takes one integer parameter. It prints double the value to the console. Invoke the function with the following arguments:

- 5
- -2
- 0
- 100
- 1000000000 // 1 billion
- 2000000000 // 2 billion

Goal 3 - Cubical

Write a function that takes one integer parameter. It prints the cubical of the number (the number to the power of three) to the console. Invoke the function with the following arguments:

- 1
- 0
- 2
- 3
- -4

Goal 4 - Square

Write a function that takes one rational number parameter. It returns the square of the number as a rational number. Invoke the function and assign the result to a variable, then print the value of the variable to the console for the following arguments:

- 0.5f
- -2f
- 0f
- 4.2f

Goal 5 - Countdown

Implement the countdown again by having a function invoke itself (recursively) printing the numbers while counting down. The function takes one argument of type integer. e.g.:

- Argument: 10
- Output: 10...9...8...7...6...5...4...3...2...1...0...

Goal 6 - Fibonacci

- A classic! Implement a function that implements the nth Fibonacci Number
- Fibonacci number is a number that grows increasingly faster for higher numbers of the sequence
- Fibonacci(0) is defined as: 0
- Fibonacci(1) is defined as: 1
- For every other one:
- Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
- Implement this function as a function that takes n as an argument and returns the correct result.

Sequence Number (n)	Result (Fibonacci(n))
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89