Functions

Is it functioning?

void MakeLasagna();

Here, we will learn about parameter-less functions without returned values.

Functions allow us to pack Blocks of Code into reusable packages.

Functions solves problems:

Think about writing code to prepare food, maybe a lasagna, and its specific set of instructions that goes through step by step on how to create this delicious meal.

```
Console.WriteLine("Fry meat.");
Console.WriteLine("Cut onions.");
Console.WriteLine("Fry onions.");
Console.WriteLine("Add tomato sauce.");
Console.WriteLine("Add spices.");
Console.WriteLine("Boil.");
```

The problem:

Now imagine that in our code, there's multiple occasions in which we want to make Lasagna:

```
Console.WriteLine("Time for breakfast.");
if(isHungry) {
  // insert code for making lasagna here.
}

Console.WriteLine("Time for lunch.");
if(isHungry) {
  // insert code for making lasagna here.
}

Console.WriteLine("Time for dinner.");
if(isHungry) {
  // insert code for making lasagna here.
}
```

… am I repeating myself?

As you can see from the image, the code would be quite hefty and we have repeated ourselves a lot.

Imagine if we were to add another set of instructions like including garlic, then there would be numerous places we need to add our newfound addition to our recipe!

If only there was a way to combat this?

```
Console.WriteLine("Time for breakfast.");
if(isHungry)
{
  Console.WriteLine("Fry meat.");
  Console.WriteLine("Cut onions.");
  Console.WriteLine("Fry onions.");
  Console.WriteLine("Add tomato sauce.");
  Console.WriteLine("Add spices.");
  Console.WriteLine("Boil.");
}
// ...
Console.WriteLine("Time for lunch.");
if(isHungry)
{
  Console.WriteLine("Fry meat.");
  Console.WriteLine("Cut onions.");
  Console.WriteLine("Fry onions.");
  Console.WriteLine("Add tomato sauce.");
  Console.WriteLine("Add spices.");
  Console.WriteLine("Boil.");
}
// ...
Console.WriteLine("Time for dinner.");
if(isHungry)
{
  Console.WriteLine("Fry meat.");
  Console.WriteLine("Cut onions.");
  Console.WriteLine("Fry onions.");
  Console.WriteLine("Add tomato sauce.");
  Console.WriteLine("Add spices.");
  Console.WriteLine("Boil.");
}
```

The solution:

We could define a function like this:

```
void MakeLasagna()
{
  Console.WriteLine("Fry meat.");
  Console.WriteLine("Cut onions.");
  Console.WriteLine("Fry onions.");
  Console.WriteLine("Add tomato sauce.");
  Console.WriteLine("Add spices.");
  Console.WriteLine("Boil.");
}
```

Invoke it

Then we could simply invoke (call it) the function like so:

MakeLasagna();

Everytime you invoke this function it will execute it's set of instructions, line by line, and then return to where your code left off before its call.

```
MakeLasagna(); // This will first execute the Lasagna-Recipe Line-By-Line
// And then continue here:
Console.WriteLine("I'm done making Lasagna!");
MakeLasagna(); // This will again execute the Lasagna-Recipe Line-By-Line
Console.WriteLine("Oops, I did it again.");
```

Much better..

This new function allows us update our previous code sample to look like this:

```
Console.WriteLine("Time for breakfast.");
if(isHungry) {
  MakeLasagna();
}

Console.WriteLine("Time for lunch.");
if(isHungry) {
  MakeLasagna();
}

Console.WriteLine("Time for dinner.");
if(isHungry) {
  MakeLasagna();
}
```

The syntax:

You define a function like this:

RETURN_TYPE FUNCTION_NAME(PARAMETER_LIST)

{ // Function Body/Scope Start

// <- Put the Code of the Function here

} // Function Body/Scope End

void

In the easiest case for the Return Type, it is of Type **void**, which stands for "nothing":

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

This means, that the function does not return anything. Which again means, that there is no result that you could assign to a variable:

```
int result = SayHello(); // Error, Type `void` can not be case to Type `int`
```

Now you might try to define a variable of Type void in order to fix above error, but that won't work either:

```
void result = SayHello(); // Error, can not define variable of Type `void`
```

This proves: **void** means that there is **no result** from this function.

Empty parameters

In the easiest case, the Parameter-List is empty. This means, that no Argument NEEDS TO and no Argument CAN be passed to the function:

SayHello(); // OKAY

Arguments for Parameters can be passed within the Parentheses. But only, if the function requires them:

SayHello(5); // Error: SayHello takes 0 Arguments, but 1 was given.

You have done this already, when using **Console.WriteLine;** You passed an Argument of Type string to the function:

Console.WriteLine("Hello");

Goal 1 - Hello World as function

- Write a C# program that defines a parameter-less void function to print a simple "Hello, World!" message.
- Call this function from the main program.
- **Output: Hello, World!**

Goal 2 - Supporter

- Write a C# program that defines a parameter-less void function called AskForHelp().
- It prints "Asking for help." to the console and then invokes another function named IAmTheHelp.
- That function prints "I am the help!".
- Invoke the function AskForHelp() from the main program.

Goal 3 - Random

- Develop a C# program that contains a parameter-less void function to generate and print a random number between 1 and 100.
- Call this function multiple times to display different random numbers.

Goal 4 - Movement

- You are only allowed to print:
  - "Move Forward."
  - "Turn Right."
- Using those two prints, make a character:
  - run three steps forward.
  - then turn around and run three steps forward
  - then turn left and run three steps forward.
  - then turn left and walk one step forward.
  - then turn around and walk one step forward.
- Make sure to introduce functions in order to avoid repetitive code!

```
Output:Move Forward.
Output:Move Forward.
Output:Move Forward.
Output:Turn Right.
Output:Turn Right.
Output:Move Forward.
Output:Move Forward.
Output:Move Forward.
Output:Turn Right.
Output:Turn Right.
Output:Turn Right.
Output:Move Forward.
Output:Move Forward.
Output:Move Forward.
Output:Turn Right.
Output:Turn Right.
Output:Turn Right.
Output:Move Forward.
Output:Turn Right.
Output:Turn Right.
Output:Move Forward.
```