How do we know if we need parameters in a method? Why do we need them?

As yourself the following question, when writing a method with or without parameters:

Does the method have access to the data it needs to perform its task?

- If yes:
  - No parameters
- Else
  - Which value(s) the method needs to have to do its task.
    - For each value, determine the type and then declare a parameter for it.
    -

Example: BMICalculator

We have to calculate BMI and for that we need to know the height, the weight and the name of the person whose BMI is to be calculated. Height and weight can have decimal values (double type) and name is text (string). We write a class **BMICalculator**.

We write one or more methods to calculate BMI. These methods MUST have the above values because they must come from the **MainForm** (or another class that uses **BMICalcualtor**).

Two ways to go:

1. We write methods with parameters to receive the above.
   ```
   public  double CalculateBMI (string name, double height, double weight)
   {
     //code
   }
   ```

   Every time the method is called, the values must be provided by the caller.

2. We let the class have instances variables to store the values. Once the values are in an instance of the class, then all methods of the **BMICalculator** will have access to them and can use them in their calculations.

Which alternative to choose? The second one is easier and better if the data are input to the class. They belong to the class and each object must know its input data (like every person must know its own properties, name, height, weight, etc).

```
class BMICalculator
{
   private string name = "No Name";
   private double height = 0;   //m, inch
   private double weight = 0;   //kg, lb
   private UnitTypes unit;
```

But there is a problem. The above data must come from the client of the class (**MainForm** which in turn takes the values from the user of the application), but **MainForm** does not have access to the private fields! We actually don't want them to have direct access!

What to do?

Write a public method and let the caller send each of the value as a parameter. We check the values and if they are good, then save them in the above.

In Assignment 3, we use a standard method (setter).

In Assignment 4, we use Properties instead.

We can choose to have more than one parameter in a method (all the three) or work with each parameter separately. Use the latter (when it is about input):

```
public void SetName(string value)
{
    //validate value before accepting it!
    if (!string.IsNullOrEmpty ( value ))
        name = value;
}

public void SetHeight (double value)
{
    if (value >= 0)    //validate before accepting
        height = value;
}

public void SetUnit(UnitTypes value)
{
    //No validation needed as the caller cannot
    //give any wrong value
    unit = value;
}
```

The main purpose for BMICalculator is to receive the input in some way. This is how it has to be done. Once you understand the above, then you will have a clear understanding of why we can use Properties instead (next module).

The class defines method (so called setter methods) to receive input from a client object (MainForm), but the client may lose the above values because it assumes that **BMICalculator** maintains them. If **MainForm** or any other object wants to know which height, name or unit values stored in an **BMICalculator** object, they should have access to the values (if you, the programmer think it is ok). To give them access to the values, we write getter methods:

```
public double GetHeight()
{
    return height; //no need for any manipulation
}

public string GetName ( )
{
    return name;
}
```

Etc.

The idea behind method parameters (argument) is to pass values to the methods or receive values (in addition to the return value). In the latter case, we use out (or ref) type of variables (as in TryParse methods).

Now the caller (MainForm) can use the above setter methods to furnish the object of **BMICalculator** it uses with input, and whenever it needs to know about the value again, it calls the getter methods.

The **MainForm** class:

```
public partial class MainForm : Form
{
    // create an instance of BMICalculator
    private BMICalculator bmiCalc = new BMICalculator ( );
```

MainForm gets its instance of the **BMICalculator** and use it, just as you buy a hammer and use its capabilities.

MainForm will pass input to the bmiCalc (assume for a while that we send the value without any processing and that the value is a good number)

```
private bool ReadHeight ( )
{
    double outValue = 0;

    if (double.TryParse ( txtHeight.Text, out outValue ); //user's value
       bmiCalc.SetHeight ( outValue);  //setter method to save the value in the
                                       //object
```

The method **SetHeight** is the setter method in the **BMICalculator**.
You can go on with the other input in the same manner.

We are not using the getter methods but they are good to have for the future.

Sometimes you will pass values between methods of the same class, the same argumentation is to be used to decide if parameters are needed. In fact in all cases, decide the need for method parameters based on whether the methods need extra values in addition to what they already have access to. Sometimes you may need to have method parameters for both input to and output from the same method. A good example is the TryPars (which .NET programmers have designed).In the C# language specification, the method has the following structure:

```
public static bool TryParse(
        string s,
        out int result
)
```

s is input to the method result is output from the method. It has a bool return type too! In C# you can only have one return value. If you need more values from a method, you can use out-parameters as in above.

```
bool ok = double.TryParse ( txtHeight.Text, out outValue );
```

We send the text (ex "123.5" five chars) and get its corresponding double value (123.5, a double). In addition, we also receive a true value (if outValue = 123.5) or a false value (if something goes wrong) as the return value of the method.