# ”Modern C# for Python Developers”

## Lab Manual

**Jesper Gulmann Henriksen**

**05-09-2025**

v1.0

# Table of Contents

## Exercise types

The exercises in the present lab manual differ in type and difficulty. Most exercises can be solved by applying the techniques from the presentations in the slides in a direct manner. Such exercises are not categorized further.

However, the remaining exercises differs slightly in the sense that they are not necessarily easily solvable. These are categorized as follows:

Labs marked with a single star denote that the corresponding exercises are a bit more loosely specified.

Labs marked with two stars denote that the corresponding exercises contain only a few hints (or none at all!) or might be a bit more difficult or nonessential. They might even require additional searches for information elsewhere than in the slide presentations.

Labs marked with three stars denote that the corresponding exercises are not expected in any way to be solved. These are difficult, tricky, or mind-bending exercises for the interested participants – mostly for fun! ☺

## Prerequisites

The present labs require the course files accompanying the course to be extracted in some directory path, e.g.

```
C:\Code\ModernCSforPyDevs
```

with .NET 8 and a compatible IDE or tools installed on the machine.

We will henceforth refer to the chosen installation path containing the lab files as *PathToCourseFiles* .

# Session 1

None. 😊

# Session 2

## Lab 2.1: "Inheritance"

The purpose of this exercise is to get hands-on experience with the unfamiliar syntax of C# for concepts that are very familiar to you in Python.

- Open the starter project in
  *PathToCourseFiles*`\Session 2\Labs\Lab 02.1\Begin` ,
  which contains a project called `Inheritance`.

This is essentially the example from the presentation cleaned up slightly. Each class has been moved to a separate file and provided a namespace definition.

### a) Make it compile

For some syntactic reason the program doesn't compile now.

- Fix the syntactic error and compile and run the program.

### b) Extend the textual presentation

When the program runs it currently prints the following output to the console:

```
John Doe [25763 C# lines]
John Doe [26250 C# lines]
```

All employees have an employee number which is not printed.

- Make the output appear as

```
100000: John Doe [25763 C# lines]
100000: John Doe [26250 C# lines]
```

### c) Another Employee Type

The fictitious company OODev has several other employee types other than `SoftwareEngineer`.

- Create an appropriate type `SoftwareArchitect` creating Visio drawings at work.

Adjust the main program such that it produces the following output:

```
100001: Jane Doe [176 drawings]
100000: John Doe [25763 C# lines]
100001: Jane Doe [178 drawings]
100000: John Doe [26250 C# lines]
```

## Lab 2.2: "Exception Handling" ( )

The purpose of this exercise is to map a Python program to C# and discover the similarities and problems.

- Open the starter project in
  *PathToCourseFiles*`\Session 2\Labs\Lab 02.2\Begin` ,

which contains a project called `ExceptionHandling`.

The project contains the following Python function:

```
def divide(x,y):
    try:
        result = x/y
    except ZeroDivisionError:
        print("Please make 'y' non-zero")
    except:
        print("Something went wrong")
    else:
        print(f"Your answer is {result}")
    finally:
        print("Aaaaand... We're done!")
```

Your task is now to

- Rewrite it to an equivalent C# method.
- Run the program and test it.

Did you encounter any problems or surprises?

## Lab 2.3: "Object Deconstruction" (⭐⭐)

The purpose of this exercise is to implement object destruction to tuples of a preexisting class.

- Open the starter project in
  *PathToCourseFiles*`\Session 2\Labs\Lab 02.3\Begin` ,

which contains a project called `ObjectDestruction`.

The starter solution consists of two projects – a client project and a class library called `DiscographyLab`.

The class library contains an existing class `Album`. That class is part of an externally supplied API and cannot be modified or derived from:

```
public sealed class Album(string artist, string albumName,
                          DateTime releaseDate)
{
    public Guid Id { get; } = Guid.NewGuid();
    public string Artist { get; } = artist;
    public string AlbumName { get; } = albumName;
    public DateTime ReleaseDate { get; } = releaseDate;
}
```

The client project contains top-level statements with the following code:

```
Album album = new Album(
    "Depeche Mode",
    "Violator",
    new DateTime( 1990, 3, 19 )
);

(_, string summary, int age) = album;
Console.WriteLine( $"{summary} is {age} years old");
```

Currently this code does not compile. You need to fix that.

- Your task is to add an appropriate class and method to make the code compile.
  - Note: You should **not** change anything in the top-level statements or the `Album` class.

When completed, your `Program.cs` should produce the following output:

```
"Violator" by Depeche Mode is 35 years old
```

## Lab 2.4: "Expression-bodied Members" ( )

- Open your completed project in
  *PathToCourseFiles*`\Session 2\Labs\Lab 02.1\Complete` ,
  which contains your `Inheritance` project completed earlier.

Let's update existing methods and properties to be expression-bodied and more functional and "modern".

- Make as much of the solution as you like expression-bodied.
- Are there areas of the code which you would prefer not to be expression-bodied?

If time permits:

- What happens if you make the `Number` property of `Employee` expression-bodied?