# Tutorial III.IV - Input and Output

Energy System Optimization with Julia

# Introduction

Welcome to this interactive Julia tutorial on working with external files! File Input/Output (I/O) operations are crucial in programming and data analysis, allowing us to persist data, share information between programs, and work with large datasets that don't fit in memory. In this tutorial, we'll cover reading and writing text files, handling CSV files, and working with delimited files using various Julia packages. These skills are fundamental for data preprocessing, analysis, and result storage in real-world applications.

Follow the instructions, write your code in the designated code blocks, and validate your results with `@assert` statements.

# Section 1 - Working with Delimited Files

Delimited files, such as CSV (Comma-Separated Values), are a common way to store structured data. Each value in the file is separated by a specific character, often a comma. Julia's DelimitedFiles package makes it easy to work with these files.

> **Tip**
>
> The DelimitedFiles package is part of Julia's Standard Library, which means you can use it without installing anything extra!

```
using DelimitedFiles
```

Now, let's create a simple matrix and save it as a CSV file:

```
# Create a 3x3 matrix
new_data = [10 12 6; 13 25 1; 40 30 7]

# Create a new folder called "ExampleData"
mkdir("$(@__DIR__)/ExampleData")

# Write the matrix to a CSV file
open("$(@__DIR__)/ExampleData/matrix.csv", "w") do io
    writedlm(io, new_data, ',')
end

println("CSV file 'matrix.csv' written successfully to folder ExampleData!")
```

> **Note**
>
> Note that we used the `@__DIR__` macro to get the directory of the current file. This is a convenient way to ensure that the file path is correct, no matter where you run the script from. The reason is that the `@__DIR__` macro returns the directory of the file in which the macro is called, not the working directory of the script you are running.

## Exercise 1.1 - Read a CSV File

Now it's your turn! Let's read the CSV file we just created.

> **Tip**
>
> To learn how to use a Julia function, you can type `?` followed by the function name in the REPL (Julia's command-line interface). For example, `?readdlm` will show you information about the `readdlm()` function.

Use the `readdlm()` function to read the 'matrix.csv' file we just created. Save the result in a variable called `read_matrix`.

```
# YOUR CODE BELOW
# Don't forget to use the @__DIR__ macro to get the correct file path!
```

```
# Test your answer
@assert read_matrix == new_data
println("File 'matrix.csv' read successfully!")
```

# Section 2 - Working with CSV Files and DataFrames

The CSV package in Julia provides powerful tools for reading and writing CSV files to and from DataFrames, a common requirement in data analysis and data science projects. This requires the CSV and DataFrames packages. If you solely followed the course so far, you first have to install the CSV Package before you can start using it:

```
import Pkg; Pkg.add("CSV")
```

## Exercise 2.1 - Write a DataFrame to a CSV File

Write the following given DataFrame to a CSV file `table_out.csv` in the folder `ExampleData`. This can be done by using the function `CSV.write()`. To learn the syntax, ask the inbuild help with `?` and the function name.

```
using CSV, DataFrames
data = DataFrame(Name = ["Elio", "Bob", "Yola"], Age = [18, 25, 29])
csv_file_path = "$(@__DIR__)/ExampleData/table_out.csv"
# YOUR CODE BELOW
```

```
# Test your answer
@assert isfile("ExampleData/table_out.csv") "Sorry, the file could not be found.
    Have you followed all steps?"
println("CSV file 'data.csv' written successfully!")
```

## Exercise 2.2 - Read a CSV File in

Read the CSV file `table_out.csv` in the folder `ExampleData` into the variable `read_data`. Here you can use the function `CSV.read()`, e.g.:

```
read_data = CSV.read("Path/datatable.csv", DataFrame)
```

> **Note**
>
> Note, that you need to provide a sink for the data when using `CSV.read()`, e.g. a DataFrame.

```
# YOUR CODE BELOW
# Again, don't forget to use the @__DIR__ macro to get the correct file path!
```

```
# Test your answer
@assert read_data[1,1] == "Elio"
println("CSV file 'table_out.csv' read successfully!")
```

# Conclusion

Congratulations!  You've successfully completed the tutorial on reading and writing external files in Julia. Continue to the next file to learn more.

# Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week's tutorial. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.