Tutorial III.IV - Input and Output

Applied Optimization with Julia

Introduction to Reading and Writing Files in Julia

This interactive Julia script introduces the essentials of interacting with external files. It covers reading and writing text files, handling CSV, and working with delimited files using DelimitedFiles. 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`, can be handled efficiently using the DelimitedFiles package in Julia. This requires the DelimitedFiles package. But as it is part of the Julia Standard Library, you can use it directly without the need to use `Pkg.add()` before.

```
using DelimitedFiles
```

The following code writes a matrix to a CSV file, seperating values with `,`.

```
new_data = [10 12 6; 13 25 1; 40 30 7]
mkdir("ExampleData")
open("ExampleData/matrix.csv", "w") do io
    writedlm(io, new_data, ',')
end
println("CSV file 'matrix.csv' written successfully to folder ExampleData!")
```

Exercise 1.1

Read the just written CSV file 'matrix.csv' using the function `readdlm()`. Save the matrix in the variable `read_matrix`. To learn the syntax, use the inbuild help by typing `?` in the terminal. Afterwards, just type the function name for an explanation.

```
# Read the just written CSV file 'matrix.csv' using the function `readdlm()`. Save the
 ↪  matrix in the variable `read_matrix`. To learn the syntax, use the inbuild help by
 ↪  typing `?` in the terminal. Afterwards, just type the function name for an
 ↪  explanation.
# YOUR CODE BELOW

# 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 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
# 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.

data = DataFrame(Name = ["Alice", "Bob", "Charlie"], Age = [25, 30, 35])
csv_file_path = "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 the CSV file `table_in.csv` in the folder `ExampleData` into the variable `read_data`. Here you can use the function `CSV.read()`. Again, use the inbuild help to familiarize yourself with the syntax. Note, that you need to provide a sink for the data when using `CSV.read()`, e.g. a DataFrame.

```
# Read the CSV file `table_in.csv` in the folder `ExampleData` into the variable
 ↪  `read_data`. Here you can use the function `CSV.read()`. Again, use the inbuild help
 ↪  to familiarize yourself with the syntax. Note, that you need to provide a sink for
 ↪  the data when using `CSV.read()`, e.g. a DataFrame.
# YOUR CODE BELOW

# Test your CSV reading
@assert read_data[1,1] == "Lisa"
println("CSV file 'table_in.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.