Tutorial II.III - Comparison Operators

Applied Optimization with Julia

Introduction

This interactive Julia script introduces the basics of comparisons and logical operators. These are fundamental for controlling the flow of programs and making decisions. Follow the instructions, write your code in the designated code blocks, and execute the corresponding code cell.

Section 1 - Comparisons

In Julia, comparison operators allow you to compare values and make decisions based on these comparisons. Common comparison operators include

- == (equal)
- != (not equal)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Comparisons return a boolean value (true or false). You can use these directly or store them in a variable. For example:

```
comparison = 5 == 5 # Store the result in 'comparison'.
println("comparison is $comparison.")
```

```
comparison is true.
```

Exercise 1.1 - Compare if 10 is greater than 5

Compare if 10 is greater than 5 and store the result in `comparison1`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert comparison1 == true
println("comparison1 is ", comparison1)
```

Exercise 1.2 - Compare if x is not equal to y

Define variables `x` with value `"Hello"` and `y` with value `"world"`. Compare if `x` is not equal to `y` and store the result in `comparison2`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert x == "Hello"
@assert y == "world"
@assert comparison2 == true
println("Comparison2 is ", comparison2)
```

Section 2 - Logical Operators

Logical operators combine or invert boolean values. In Julia, use

- `&&` for AND
- `||` for OR
- `!` for NOT

These are useful in complex conditions and controlling program flow.

Exercise 2.1 - Use the AND operator

Use the `AND` operator to check if `10` is greater than `5` and `hello` is equal to `hello`. Store the result in `logic1`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert logic1 == true
println("logic1 is ", logic1)
```

Exercise 2.2 - Use the OR operator

Use the `OR` operator to check if `10` is less than `5` or `hello` is equal to `hello`. Store the result in `logic2`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert logic2 == true
println("logic2 is ", logic2)
```

> **Tip**
>
> Julia uses short-circuit evaluation for `&&` and `||` operators. This means that the second operand is only evaluated if necessary.

Exercise 2.3 - Use the NOT operator

Check whether `10` is greater than `5` and store the result in `logic3`. Then, use the `NOT` operator to invert `logic3`. Store the result in `logic4`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert logic3 == true
@assert logic4 == false
println("logic3 is ", logic3," and logic4 is", logic4)
```

Exercise 2.4 - Chaining Comparisons

Julia allows you to chain multiple comparisons:

```
y = 2
result = 1 < y < 5  # Equivalent to (1 < y) && (y < 5)
println("result is $result.")
```

```
result is true.
```

Check if `x` is between `1` and `10` (exclusive) using a chained comparison. Store the result in `chained_comparison`.

```
x = 5
# YOUR CODE BELOW
```

```
# Test your answer
@assert chained_comparison == true
println("chained_comparison is ", chained_comparison)
```

Conclusion

Excellent work! You've completed the tutorial on Comparisons and logical operators in Julia. You've learned to compare values and use logical operators to combine or invert boolean values. Experiment with the code, try different operations, and understand how Julia handles logic. 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.