Tutorial II.III - Comparison Operators

Applied Optimization with Julia

Introduction

Imagine you're teaching a computer to make decisions. Just like we compare things in everyday life ("Is it raining?", "Do I have enough money?"), computers need ways to compare values and make choices. This tutorial will show you how to help computers make these comparisons!

Follow the instructions, write your code in the designated code blocks, and execute the corresponding code cell.

Section 1 - Comparisons

In everyday life, we make comparisons all the time:

- Is this coffee too hot?
- Do I have enough money for lunch?
- Is my password correct?

In Julia, we use special symbols to make these comparisons. Comparisons return a boolean value (true or false). You can use these directly or store them in a variable. For example:

```
user_input = "secret123"
password_correct = (user_input == "secret123")
```

```
true
```

```
coffee_temp = 75
temperature_safe = (coffee_temp <= 70)
```

```
false
```

The following are potential real-world examples:

```
# Do I have enough money?
enough_money = (wallet >= price)
# Is the person an adult?
is_adult = (age >= 18)
# Is this person Elio?
same_name = (name == "Elio")
```

Here are all the comparison operators:

Symbol	Meaning	Real-world Example
`==`	Equal to	Is my password correct?
`!=`	Not equal to	Is this a different person?
`<`	Less than	Is it colder than freezing?
`>`	Greater than	Do I have more than $10?
`<=`	Less than or equal to	Can this ride fit in my garage?
`>=`	Greater than or equal to	Am I old enough to vote?

Let's try some examples:

```
# Temperature comparison
temp = 25
is_freezing = (temp <= 0)
println("Is it freezing? $is_freezing")
```

```
Is it freezing? false
```

```
# Price comparison
budget = 50
price = 45
can_afford = (budget >= price)
println("Can I afford it? $can_afford")
```

```
Can I afford it? 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
```

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

Section 2 - Logical Operators

Sometimes we need to combine multiple comparisons in our code, just like in real life decisions:

- "Can I go to the party?" might depend on:
  - (Is it weekend?) AND (Have I finished my studies?)
- "Should I take an umbrella?" might depend on:
  - (Is it raining?) OR (Is it likely to rain?)

Julia uses three main logical operators:

```julia
# AND (&&): Both conditions must be true
can_go_to_party = (is_weekend && homework_done)

# OR (||): At least one condition must be true
need_umbrella = (is_raining || forecast_says_rain)

# NOT (!): Opposite of the condition
is_closed = !is_open
```

Real-world Examples:

```julia
# Can I buy this game?
age = 25
money = 60
game_price = 50
can_buy = (age >= 18) && (money >= game_price)
println("Can I buy the game? $can_buy")
```

```
Can I buy the game? true
```

```julia
# Should I wear a coat?
temperature = 5
is_raining = true
need_coat = (temperature < 10) || is_raining
println("Should I wear a coat? $need_coat")
```

```
Should I wear a coat? true
```

> Tip
>
> Think of those logical operators as follows:
>
> - `&&` (AND) as a strict condition - everything must be perfect
> - `||` (OR) as a lenient condition - any good reason is enough
> - `!` (NOT) as the opposite of something

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 has a neat feature that lets you write comparisons the way you think about them:

```
# Instead of writing:
age >= 13 && age <= 19    # Is age between 13 and 19?

# You can write:
13 <= age <= 19           # Much more natural!
```

Real-world examples:

```
# Is the body temperature normal?
normal_temperature = (36.5 <= body_temp <= 37.5)

# Is the current hour within working hours?
working_hours = (9 <= current_hour < 17)
```

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.

Bibliography