Tutorial II.V - Variable Scope

Applied Optimization with Julia

Introduction

This interactive Julia script is a look on variable scopes in Julia. Understand the intricacies of local and global scopes, comprehend the significance of `local` and `global` keywords, and discern the scope behaviors in different contexts. Follow the structured instructions, implement your code in the designated blocks, and affirm your comprehension with `@assert` statements.

Section 1 - The Local and Global Scope

In Julia, variables within a function or a loop are encapsulated in a local scope, meaning their visibility and lifespan are confined to that block of code. Variables declared in the main body of a script are in the global scope, making them accessible and modifiable across your entire program. We have seen this in the previous tutorial on Loops, where we have created scope blocks with `let ... end` to prevent us from creating global variables.

Exercise 1.1 - Experiment with Local Variables in a `for` Loop

Experiment with local variables in a `for` loop and try to execute the following block of code.

```
println("Defining a local variable within a loop:")
for i in 1:3
    loop_variable = i
    println("Inside the loop, loop_variable is: ", loop_variable)
end
println("Outside the loop, loop_variable is: ", loop_variable)

# Test your answer
@assert loop_variable == 3
println("Now, the loop works and the value of the 'loop_variable' is: ", loop_variable)
```

> **Note**
>
> Attempting to access `loop_variable` here should result in an error because its scope is local to the loop. Can you come up with a way to solve this issue? Extend the code above to allow the execution of the code.

Exercise 1.2 - Experiment with Local Variables in a Nested Loop

Experiment with local variables in a nested loop. Try to execute the following block of code.

```
println("Defining a local variable within a loop:")
for i in 1:3
    nestedloop_variable = 0
    for j in 1:3
        nestedloop_variable = i * 3
        println("Inside the nested loop, loop_variable is: ", nestedloop_variable)
    end
end
println("Outside the nested loop, loop_variable is: ", nestedloop_variable)
```

```
# Test your answer
@assert nestedloop_variable == 9
println("Now, the loop works and the value of the 'nestedloop_variable' is: ",
 ↪  nestedloop_variable)
```

> **Note**
>
> Attempting to access `nestedloop_variable` here should result in an error because its scope is only accessible in the first loop, where it is defined, and the second loop below. Can you come up with a way to solve this issue? Extend the code above to allow the execution of the code.

Section 2 - Working with `global` and `local` Keywords

Use `global` to designate a global variable inside a local scope. Conversely, `local` explicitly defines a local variable, particularly in contexts where global inference might occur.

Exercise 2.1 - Experiment with Local Variables in a Potentially Global Context

Experiment with local variables in a potentially global context using `local`. The objective is to redefine the value of `explicit_local_variable` to "end" within the following loop. Try to execute the following block of code.

```
explicit_local_variable = "start"
println("At the start of the loop, explicit_local_variable is: ",
 ↪  explicit_local_variable)
for i in 1:3
    local explicit_local_variable = "end"
    println("Inside the loop, explicit_local_variable is: ", explicit_local_variable)
end
println("Outside the loop, explicit_local_variable is: ", explicit_local_variable)

# Test your answer
@assert explicit_local_variable == "end" "Currently, the value of
 ↪  'explicit_local_variable' is $explicit_local_variable."
```

> **Note**
>
> Attempting to access `explicit_local_variable` here should result in an error because its scope is local to the loop. Can you come up with a way to solve this issue? Extend the code above to allow the execution of the code.

Conclusion

Great! You've just navigated through the variable scopes in Julia. Although this is quite complicated at first, it will save you much time later on if you understand this concept. 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.