Tutorial II.IV - Loops

Applied Optimization with Julia

Comprehensive Guide to Loops in Julia

This interactive Julia script is an extensive tutorial introducing you to the robust loop structures in Julia. Improve your skills in for and while loops for performing repetitive tasks, manipulating loop flow with break and continue, and applying nested loops to iterate through multi-dimensional structures. Follow the instructions, input your code in the designated areas, and verify your implementations with @assert statements.

Section 1: For Loops

`for` loops iterate over a sequence such as a range or collection, executing a block of code for each item. Basic syntax:

```
for variable in collection
    Execute code for each element
end
```

Iterating over a range (1 to 3):

```
for i in 1:3
    println(i)
end
```

This will print `1`, `2`, and `3`.

Iterating over an array:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits
    println(fruit)
end
```

This will print each fruit in the fruits array. The `break` statement can be utilized to exit the loop based on a condition. To check some condition, we can use `if` statements. For example:

```
loop_number = 0
for x in 1:10
    loop_number = x
    if loop_number == 4
        break
    end
end
```

This would exit the loop in iteration `4`, as the condition `loop_number == 4` would be true here. We can also chain `if` statements.For example:

```
loop_number = 0
for x in 1:10
    loop_number = x
    if loop_number <= 2
        println(loop_number)
    elseif loop_number == 3
        println("We reached 3!")
    else
        break
    end
end
```

This would print `1`, then `2`, then `We reached 3!`. Afterwards the loop would end, as the break statement kicks in.

Exercise 1.1

Sum the numbers from `1` to `5` in a loop. Don't worry about the `let` here, it just starts a scope block. More on this later in the tutorial on variable scope. The next line initializes `sum_numbers` to `0`. The sum you compute should accumulate in this variable.

```
# Calculate the sum of numbers from 1 to 5 in a loop. Don't worry about the let here, it
↪  just starts a scope block. More on this later in the tutorial on variable scope. The
↪  next line initializes sum_numbers to 0. The sum you compute should accumulate in this
↪  variable.
let
sum_numbers = 0
# YOUR CODE BELOW

# Test your answer
@assert sum_numbers == 15
println("Sum of numbers from 1 to 5: ", sum_numbers)
end # This ends the scope of the previously started scope block.
```

Exercise 1.2

Sum only the even numbers from `1` to `10`. Again, we initialize a variable `sum_evens` to `0`. The sum you compute should accumulate in this variable.

```
# Sum only the even numbers from 1 to 10. Again, we initialize a variable 'sum_evens' to
↪  0. The sum you compute should accumulate in this variable.
let # First, we start a scope block for the loop!
sum_evens = 0
# YOUR CODE BELOW

# Test your answer
@assert sum_evens == 30
println("Sum of even numbers from 1 to 10: ", sum_evens)
end # Here we end the scope block again.
```

Note

Hint: Utilize the remainder operator (%) to verify if a number is even. `number % 2 == 0` implies that `number` is even.

Exercise 1.3

Iterate over each fruit in the `fruits` array, store the current fruit in `current_fruit`, and exit the loop if `current_fruit` is `banana`. The next lines initialize the `fruits` array and `current_fruit` variable.

```julia
# Iterate over each fruit in the 'fruits' array, store the current fruit  in
↪  'current_fruit', and exit the loop if 'current_fruit' is "banana". The next lines
↪  initialize the 'fruits' array and 'current_fruit' variable. But first, another scope
↪  block.
let
fruits = ["apple", "banana", "cherry"]
current_fruit = "None"
# YOUR CODE BELOW

# Test your answer
@assert current_fruit == "banana"
println("The current fruit is: ", current_fruit)
end # Here we end the scope block again.
```

Section 2: While Loops for Conditional Execution

`while` loops execute as long as a specified condition holds true. They're particularly useful when the number of iterations is dynamic or unknown in advance.

Exercise 2.1

Subtract from `10` in increments of `1` until the result is less than `3`. The next line initializes `current_value` to `10`. The result should be in this variable.

```julia
# Subtract from 10 in increments of 1 until the result is less than 3. The next line
↪  initializes current_value to 10. The result should be in this variable. But first, we
↪  startnew scope block.
let
current_value = 10
# YOUR CODE BELOW

# Test your answer
@assert current_value == 2
println("The first value smaller than 3 is: ", current_value)
end # Scope block closed again.
```

Exercise 2.2

Find the first multiple of `7` greater than `50` using an indefinite loop. The next line initializes `first_multiple_of_7` to `0`. The first multiple should be in this variable.

```julia
# Find the first multiple of 7 greater than 50 using an indefinite loop.  The next line
↪  initializes 'first_multiple_of_7' to 0. The first multiple should be in this
↪  variable. But first, we start a new scope block.
let
first_multiple_of_7 = 0
# YOUR CODE BELOW

# Test your answer
@assert first_multiple_of_7 == 56
println("First multiple of 7 greater than 50: ", first_multiple_of_7)
end # End of scope block
```

Tip

‘while true … end’ constructs an infinite loop. You can exit the loop using a ‘break’ statement if a condition is met.

Section 3: Nested Loops

Nested loops are loops within another loop, useful for iterating over multi-dimensional data structures.

Exercise 3.1

Compute the product of each pair of elements from two arrays. The next lines initialize `numbers1`, `numbers2` arrays, and the `products` array to store your results.

```julia
# Compute the product of each pair of elements from two arrays. The next lines initialize
↪  numbers1, numbers2 arrays, and the products array to store your results. But first,
↪  we start a new scope block.
let
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]
products = []
# YOUR CODE BELOW

# Test your answer
@assert products == [4, 5, 6, 8, 10, 12, 12, 15, 18]
println("Products of each pair from two arrays: ", products)
end # End scope block
```

Tip

Remember, you can use push!() to append elements to an array.

Conclusion

Great work! You’ve successfully navigated through the basics of loops in Julia. You’ve seen for and while loops, tackled iterable structure, and worked on nested loops. 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.