Department Of Computer Engineering

CS 315 Programming Languages

Homework 3

Berdan Akyürek
21600904
Section 2

1. Sample Codes and Results

Subprograms are called functions in Julia. The following examples show how nested functions, local variables, parameter passing, keyword arguments, default arguments and closures work in functions of Julia.

1.1 Nested subprogram definitions

Nested functions can be simply written inside another function in Julia. A local variable of the outer function is reachable by the inner function as shown in the example below.

1.1.1 Code

```
println("############################################")
println("Testing nested subprograms.")

function nestedFunctionTest()
   x = 3
   println("x inside nestedFunctionTest: ", x)
   function innerFunction()
       # Following line prints 3 because innerFunction
       # is inside nestedFunctionTest after definition
       # of x and the scope of x is between the definition
       # and the end of the function nestedFunctionTest.
       # So innerFunction is inside the scope of x
       println("x inside innerFunction: ", x)
   end
   innerFunction()
end

nestedFunctionTest()
```

1.1.2 Output

```
############################################
Testing nested subprograms.
x inside nestedFunctionTest: 3
x inside innerFunction: 3
```

1.2 Scope of local variables

The scope of a local variable starts with the definition of that variable and ends with the end of that function[1] as shown below. The closure is an exception of this.

1.2.1 Code

```
println("############################################")
println("Testing the scope of a local variable.")

function localVariableTest()
   try
       # Following line will not be executed because the scope of
y
       # is between its definition and the end of the function.
       # This line is outside of this scope.
       println("y inside localVariableTest before definition: ",
y)
   catch e
       # Following line will be executed because the line
       # inside the try block will give an error.
       println("y is not defined.")
   end
   y = 3
   # Following line will be executed because the scope of y
   # is between its definition and the end of the function.
   # This line is between the definition and the end.
   println("y inside localVariableTest after definition: ", y)
end


localVariableTest()
try
   # Following line will not be executed and give an error
   # Because the scope of y is from the definition of y to
   # the end of the function localVariableTest.
   # This line is out of scope
   println("y after function call: ", y)
catch e
   # Following line will be executed because the line
   # inside the try block will give an error.
   println("y is not defined outside foo.")
end
```

1.2.2 Output

```
############################################
Testing the scope of a local variable.
y is not defined.
y inside localVariableTest after definition: 3
y is not defined outside foo.
```

1.3 Parameter passing methods

In Julia, parameters are passed using pass-by-sharing[2]. An example of parameter pasing is as follows.

1.3.1 Code

```julia
println("############################################")
println("Testing parameter passing.")

function parameterPassingTest(z)
   z = 100
   # Following line prints 100 as expected
   println("z inside the function: ", z)
end

z = 10
parameterPassingTest(z)
# Following line prints 10. Calling parameterPassingTest
# Does not make any change. Because Julia use pass-by-sharing
# to pass parameters.
println("z after function call: ", z)
```

1.3.2 Output

```
############################################

Testing parameter passing.

z inside the function: 100

z after function call: 10
```

1.4 Keyword and default parameters

Keyword parameters can be used in Julia as positional arguments[3]. Also, it is possible to use a combination of both positional and keyword arguments[3]. To separate positional arguments and keyword arguments, “;” has to be used in the function definition[3]. Parameters before “;” are positional arguments and parameters after “;” are keyword arguments. If no keyword arguments will be used, there is no need to use “;”. Use of “;”1 in a function call is optional. The program accepts both ways. But a positional argument cannot be used as a keyword argument and a keyword argument cannot be used as a positional argument in Julia.
Also, there are default parameters in Julia. To define default parameters, “=” sign is used[4]. If there is not a value entered for a default parameter in the function call, the default value will be used.
The code below shows how keyword parameters and default parameters work in Julia.

1.4.1 Code

```
println("############################################")
println("Testing keyword parameters and default parameters.")

# This function has one positional(a) and one keyword(b)
parameters.
function keywordParameters(a;b)
   println("b inside keywordParameters: ", b)
end

function defaultParametersTest(v=1917)
   println("v inside defaultParametersTest: ", v)
end

try
   # Below line gives an error because a keyword argument
   # cannot be called with positional arguments.
   keywordParameters(3, 5)
catch
   println("Error!")
end

# This is the right way of calling a keyword argument
keywordParameters(3; b=5)

# Also use of ; is optional. So this line also works
keywordParameters(3, b=5)
```

```
try
   # This line does not execute because a positional
   # argument cannot be used as a keyword argument
   keywordParameters(a=5, b = 3)
catch
   println("Error!")
end
# The following function call prints 1972 as expected.
defaultParametersTest(1972)

# Following line prints 1917 because no arguments passed.
# So the default parameter is used.
defaultParametersTest()
```

1.4.2 Output

```
################################################
Testing keyword parameters and default parameters.
Error!
b inside keywordParameters: 5
b inside keywordParameters: 5
Error!
v inside defaultParametersTest: 1972
v inside defaultParametersTest: 1917
```

1.5 Closures

It is possible to use Closures in Julia. A function can return its inner nested function[5] and by using this function, it is possible to reach a function’s variable outside of that function and outside of the scope of that variable[5]. Closures are an exception of local variable scope. The following code shows how closures work on Julia.

1.5.1 Code

```
println("############################################")
println("Testing closure.")

function outer()
   z = 0
   println("inside outer: ", z)
   function closure()
       z += 1
       println("inside closure: ", z)
       return z
   end
   return closure
end
```

```
# After executing blow line, closure() can be called
# from outside of outer()
v = outer()

# Every time below line is executed, z increments by 1
v()
v()
v()

# t starts from 1 again. A new z will be created
t = outer()

v()
v()
t()
t()
```

1.5.2 Output

```
##################################################
Testing closure.
inside outer: 0
inside closure: 1
inside closure: 2
inside closure: 3
inside outer: 0
inside closure: 4
inside closure: 5
inside closure: 1
inside closure: 2
```

2. Language Evaluations

2.1 Readability

The end keyword makes it hard to read a function in Julia. It is possible to confuse where a function starts and ends especially in nested functions. However, the syntax of keyword parameters and default parameters are clear and easy to read. Also in terms of closure, it is simple and easy to follow. So, Julia can be considered as readable in terms of functions overall.

2.2 Writability

The end keyword reduces writability as it reduces readability too. However, the simplicity of closure, keyword parameters and default parameters makes functions in Julia writable too overall.

3. Learning Strategy

No online compilers are used in this homework. I used Julia 1.5.3 that I downloaded to my system.
I used the Julia website[6] that is given on the homework assignment to learn the usage of functions. After checking each title from the website, I started to write the required function that is asked and I kept going between coding and reading until the end of the assignment.

References

[1]"Scope of Variables · The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/manual/variables-and-scoping/#Local-Scope. [Accessed: 22- Dec- 2020].

[2]"Functions · The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/manual/functions/#Argument-Passing-Behavior. [Accessed: 22- Dec- 2020].

[3]"Functions ·Keyword Arguments· The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/manual/functions/#Keyword-Arguments. [Accessed: 22- Dec- 2020].

[4]"Functions · Optional Arguments · The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/manual/functions/#Optional-Arguments. [Accessed: 22- Dec- 2020].

[5]"Julia Functions · Closures · The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/devdocs/functions/#Closures. [Accessed: 22- Dec- 2020].

[6]"Julia Documentation · The Julia Language", *Docs.julialang.org*. [Online]. Available: https://docs.julialang.org/en/v1/. [Accessed: 22- Dec- 2020].