BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS315

Homework Assignment 3

Subprograms in Julia

Bulut Gözübüyük 21702771

1.0 Nested subprogram definitions

In the Julia programming language, nested subprograms (functioned) can be defined as below. The last expression of the inner function is returned.

1.1 Code

```
function outer(x)
    function inner()
        val + 2
    end

    val = x
    return inner()
end

print(outer(5))
```

1.2 Output

```
7
```

2.0 Scope of local variables

In Julia programming language, writing "local x" declares a new local variable in that scope in any local scope, irrespective of whether or not there is already a variable called x in the outer scope [1]. Julia applies the 3 following rules when "x = <value>" exists in a local scope:

- **Existing local**: If x is already a local variable, it is assigned a current local x [1].
- **Hard scope**: If x is not already a local variable and the assignment happens within any "hard scope construct" the assignment scope generates a new local called x [1].
- **Soft scope**: The characteristics depend on whether the global variable x is specified if “x is not already a local variable and all the scope constructs containing the assignment are” soft scopes [1].

2.1 Code

```
function outer2()
    outerLocalVar = 5 # new local variable for outer2
    println("Outer function outer local variable: 
",outerLocalVar)
    function inner2()
        innerLocalVar = 2 # new local variable for inner2
        println("Inner function inner local variable: ", 
innerLocalVar)
        println("Inner function outer local variable: ", 
outerLocalVar)
        outerLocalVar = 3 # change outer local
        println("Inner function outer local variable: ", 
outerLocalVar)
    end
    inner2()
    println("Outer function outer local variable: 
",outerLocalVar)
    # error if code below is uncommented out of scope
    #println("Inner function local variable call from 
outer: ", innerLocalVar)
end

outer2()
```

2.2 Output

```
Outer function outer local variable: 5
Inner function inner local variable: 2
Inner function outer local variable: 5
Inner function outer local variable: 3
Outer function outer local variable: 3
```

3.0 Parameter passing methods

The Julia function parameters implement a principle often referred to as "pass-by-sharing" which means that when they are passed to functions, values are not copied [2]. Modifications made inside a function to mutable values would be visible to the caller [2]. That's the same behavior observed, among many other programming languages, in Python, Ruby, and Perl [2].

3.1 Code

```
function test(param1)
    param1 = 5
    println("In test function Param1 is ", param1)
end

param1 = 10
println("Param1 is ", param1)
test(param1)
println("After calling test, Param1 is ", param1)
```

3.2 Output

```
Param1 is 10
In test function Param1 is 5
After calling test, Param1 is 10
```

4.0 Keyword and default parameters

In the Julia programming language, the keyword parameters hold the value by a specified keyword. On the other hand, the default (positional) arguments hold the value by position.

4.1 Code

```
println("------------keyword")

function keyWord(;arg1,arg2)
    return arg1 + 2 * arg2
end

println("Keyword: ", keyWord(arg1=5,arg2=2))
println("Keyword reverse: ", keyWord(arg2=2, arg1=5))

# default parameters

println("------------default")

function defaultP(arg1,arg2)
    return arg1 + 2 * arg2
end

println("Default: ", defaultP(5,2))
println("Default reverse: ", defaultP(2,5))
```

4.2 Output

```
------------keyword
Keyword: 9
Keyword reverse: 9
------------default
Default: 9
Default reverse: 12
```

5.0 Closures

In Julia programming language, closure is a callable object with “field names” matching the variables captured [1].

5.1 Code

```
function closureTest(x)
    return other -> x - other
end

println(closureTest(10)(5))
```

5.2 Output

```
5
```

6.0 Evaluation of Julia programming language

The Julia programming language seems to be a pretty readable and writable programming language in the context of subprograms. On the other hand, closures could be better in terms of readability. Functions are using the end keywords and it makes subprograms to be easier to write and read. To sum up, it can be concluded that the Julia programming language has a successful subprogram syntax.

7.0 Learning Strategy

While learning the Julia programming language, I tried to adhere to the official documentation of that programming language. If I have problems after looking at the official documents, I also search from stack overflow, then I find solutions to my issues. These two approaches allow me to complete this homework so that they can be said to be effective.

URLs of the official documentation:

- Functions https://docs.julialang.org/en/v1/manual/functions/
- Argument Passing Behavior https://docs.julialang.org/en/v1/manual/functions/#Argument-Passing-Behavior
- Scope of Variables https://docs.julialang.org/en/v1/manual/variables-and-scoping/
- Keyword Arguments https://docs.julialang.org/en/v1/manual/functions/#Keyword-Arguments
- Closures https://docs.julialang.org/en/v1/devdocs/functions/#Closures
- Scope of Default Values https://docs.julialang.org/en/v1/manual/functions/#Evaluation-Scope-of-Default-Values
- Closures https://docs.julialang.org/en/v1/devdocs/functions/#Closures

URL of the online compiler/interpreters:

- https://repl.it/languages/julia

References

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

[2] “Functions,” Functions · The Julia Language. [Online]. Available: https://docs.julialang.org/en/v1/manual/functions/. [Accessed: 22-Dec-2020].