Module	CS4012 – Representation and Modelling
Day	7
Lab	5
Topic	Variable scope

Summary

(Scope): The section of code where a variable can be accessed.

The Lua interpreter executes code in "chunks". In interactive mode, a chunk is a single executable statement. By contrast, a single lua script and all statements contained therein is considered a chunk in normal mode. Each new chunk that is executed is added to the scope of the preceding chunk. Whether a variable within one chunk is accessible in another depends on its access modifier. By default a variable is implicitly declared with "global" scope, which means that it is accessible from any point in the code after its declaration. A variable may also be declared to have "local" scope, which limits its accessibility to the chunk and block in which it is declared. With this mind, the following focuses on variable scope in the context of both blocks and chunks.

Exercise 1: The "do" block statement

It is common for two or more variables in a program's code to have the same name but have separate purposes. To avoid a name conflict, each variable of the same name should have a different scope. In Lua, a "do" block is typically used to limit the scope of variables. It is delimited as follows:

```
do
        - - > statements
end
```

When encountered it tells the interpreter that any variable declared as local can be accessed only from inside the enclosing block. Conversely, any variable declared (implicitly) as global can be accessed from any point after its declaration.

Consider the following code snippet:

```
aColour = "Red"                 - -> implicitly declared as global
do
        local aColour = "Blue"          - -> explicitly declared as local

        print(aColour)                  - -> outputs "Blue"
end
print(aColour)                  - -> outputs "Red"
```

The variable "aColour" declared (implicitly) as global, is re-declared inside the "do" block as a local variable. This has the effect of <u>hiding the global variable</u> until the end of the block. When the block ends, the local variable "aColour" is now out-of-scope, i.e. not accessible.

Task:

Consider the following code fragments and predict their output. Using the given fragments, write a program to test your predictions.

a.
```
aColour = “Green”
anotherColour = “Chartreuse”
        do
                local anotherColour = “Lime”
                anotherColour = “Sage”
        end
print(aColour)
print(anotherColour)
```

b.
```
aColour = “Green”
anotherColour = “Chartreuse”
        do
                aColour = “Sage”
                local aColour = “Lime”
        end
print(aColour)
print(anotherColour)
```

c.
```
aColour = “Green”
anotherColour = “Chartreuse”
        do
                local aColour = “Lime”
                do
                        print(aColour)
                end
        end
print(aColour)
print(anotherColour)
```

d.
```
aColour = “Green”
anotherColour = “Chartreuse”
        do
                local aColour = “Lime”
                do
                        local aColour = “Sage"
                        anotherColour = “Mint”
                end
        end
print(aColour)
print(anotherColour)
```

Hint: Blocks can be nested, such that all variables declared in the enclosing block are accessible from the nested block.

Exercise 2: Chunks in interactive and normal modes

What constitutes a chunk to the Lua interpreter depends on the mode in which it is run. The interpreter can be considered to behave *greedily* in normal mode and *reluctantly* in interactive mode. Each chunk executed by the interpreter has a discrete scope to which a variable can be local. This is illustrated in table (1) below where chunks are represented by cells; a single cell in normal mode indicates a single chunk containing all code statements.

Table 1

Normal mode	Interactive mode
x = 3	x = 3
local y = 4	local y = 4
print (x) - - > prints 3	print (x)
print(y) - - > prints 4	print(y)

do
    print(x) - -> prints 3
    print(y) - - > prints nil
    local x = 7
    y = 14
    print(x) - - > prints 7
    print(y) - - > prints 14
end
	do
	    print(x)
	    print(y)
	    local x = 7
	    y = 14
	    print(x)
	    print(y)
	end

print(x) - - > prints 3	print(x)
print(y) - - > prints 14	print(y)

Tasks:

1. Consider the output of the above code snippet executed in normal mode; and predict the output of the same code executed in interactive mode. Run the code in interactive mode one statement at a time to test your predictions.

Hint: Each chunk has its own scope and each executable statement in interactive mode is a chunk.

2. Using a single statement, run the same lines of code in interactive mode such the output is the same as when executed in normal mode.

Hint: Variables of an enclosing block are in-scope to nested blocks.