İlke Kaş

21803184

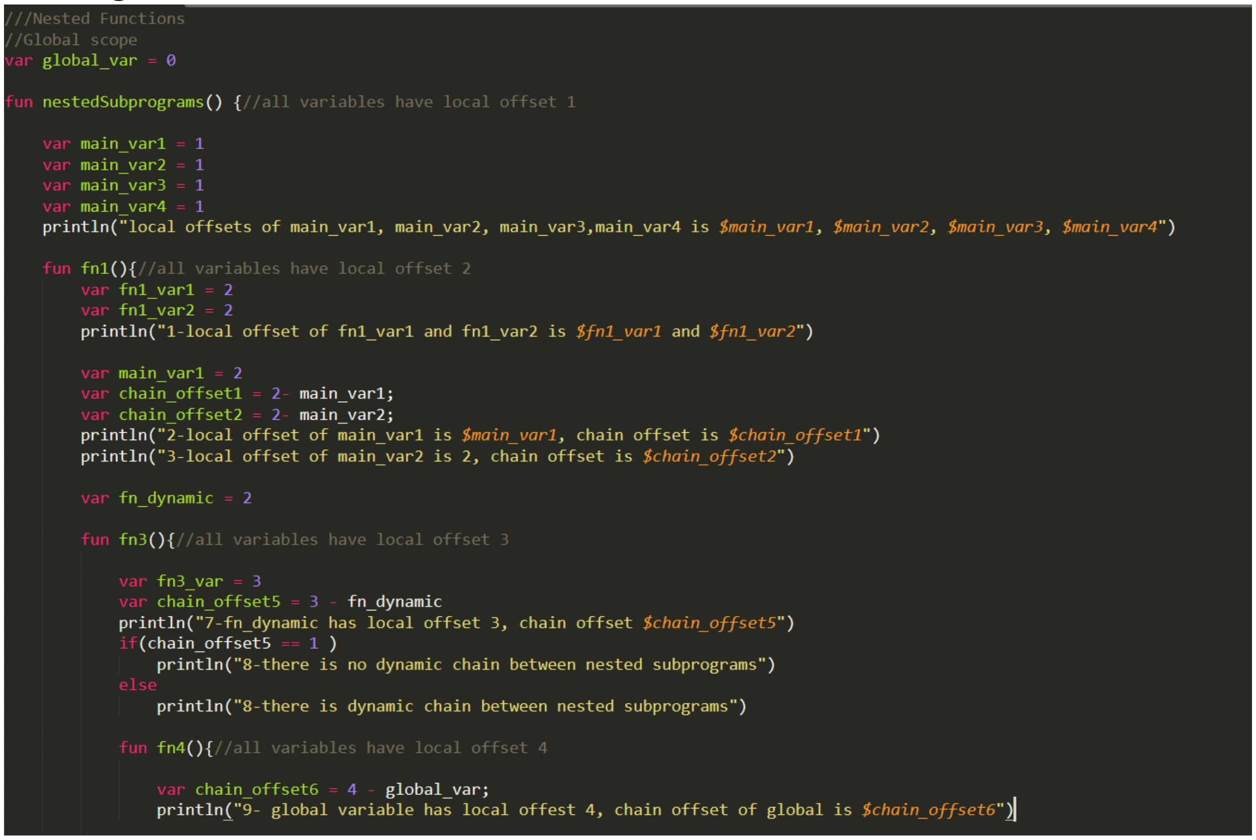
Section 1

REPORT

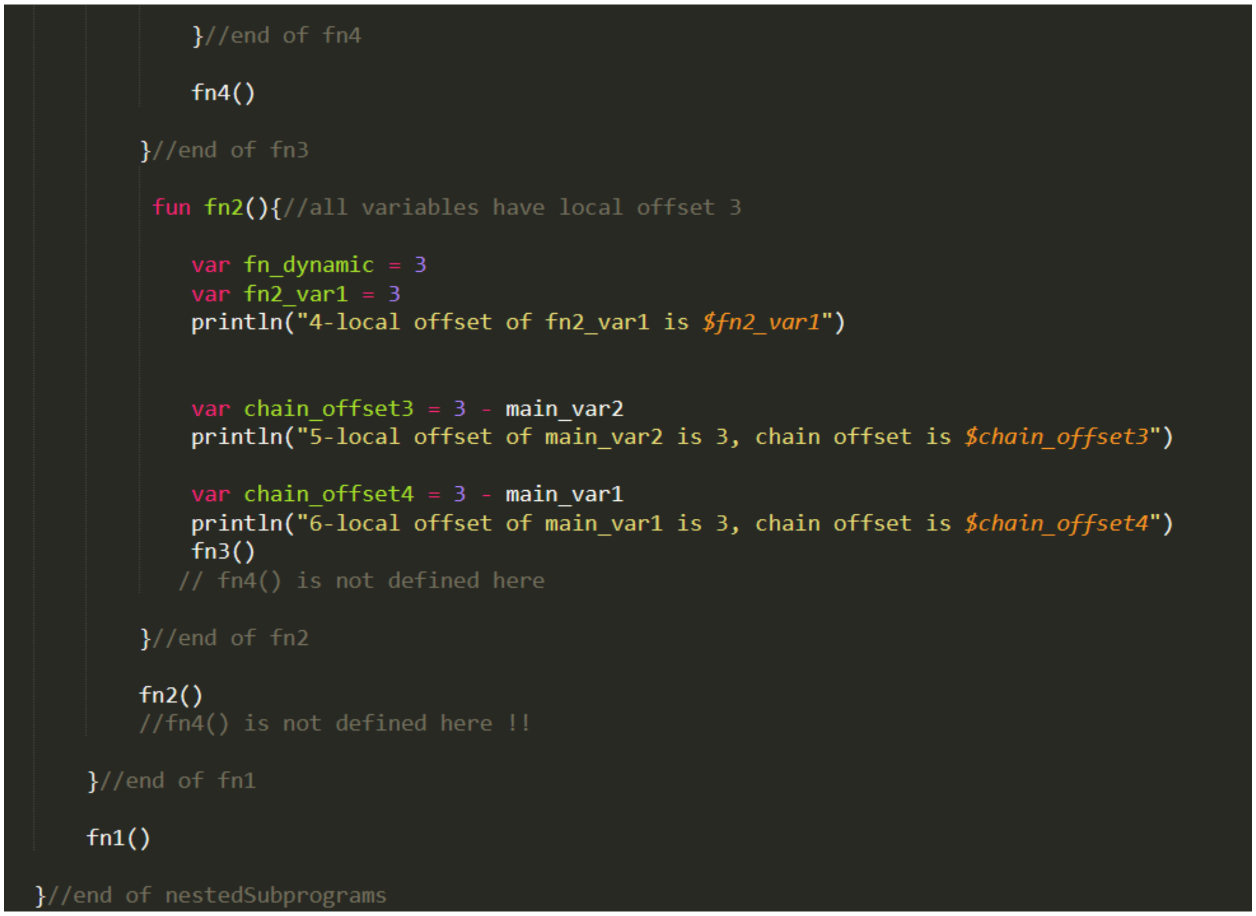
Kotlin

1. Nested subprogram definitions

- **Explanation:** Some languages can support nested subprograms like Kotlin. Static chain is one of the most common ways to implement static scoping languages nested subprograms.
- **Code Segment for:**



```
///Nested Functions
//Global scope
var global_var = 0

fun nestedSubprograms() {//all variables have local offset 1

    var main_var1 = 1
    var main_var2 = 1
    var main_var3 = 1
    var main_var4 = 1
    println("local offsets of main_var1, main_var2, main_var3,main_var4 is $main_var1, $main_var2, $main_var3, $main_var4")

    fun fn1(){//all variables have local offset 2
        var fn1_var1 = 2
        var fn1_var2 = 2
        println("1-local offset of fn1_var1 and fn1_var2 is $fn1_var1 and $fn1_var2")

        var main_var1 = 2
        var chain_offset1 = 2- main_var1;
        var chain_offset2 = 2- main_var2;
        println("2-local offset of main_var1 is $main_var1, chain offset is $chain_offset1")
        println("3-local offset of main_var2 is 2, chain offset is $chain_offset2")

        var fn_dynamic = 2

        fun fn3(){//all variables have local offset 3

            var fn3_var = 3
            var chain_offset5 = 3 - fn_dynamic
            println("7-fn_dynamic has local offset 3, chain offset $chain_offset5")
            if(chain_offset5 == 1 )
                println("8-there is no dynamic chain between nested subprograms")
            else
                println("8-there is dynamic chain between nested subprograms")

            fun fn4(){//all variables have local offset 4

                var chain_offset6 = 4 - global_var;
                println("9- global variable has local offest 4, chain offset of global is $chain_offset6")
```



```
            }//end of fn4

            fn4()

        }//end of fn3

         fun fn2(){//all variables have local offset 3

            var fn_dynamic = 3
            var fn2_var1 = 3
            println("4-local offset of fn2_var1 is $fn2_var1")


            var chain_offset3 = 3 - main_var2
            println("5-local offset of main_var2 is 3, chain offset is $chain_offset3")

            var chain_offset4 = 3 - main_var1
            println("6-local offset of main_var1 is 3, chain offset is $chain_offset4")
            fn3()
           // fn4() is not defined here

        }//end of fn2

        fn2()
        //fn4() is not defined here !!

    }//end of fn1

    fn1()

}//end of nestedSubprograms
```

- **Result Of The Code:**

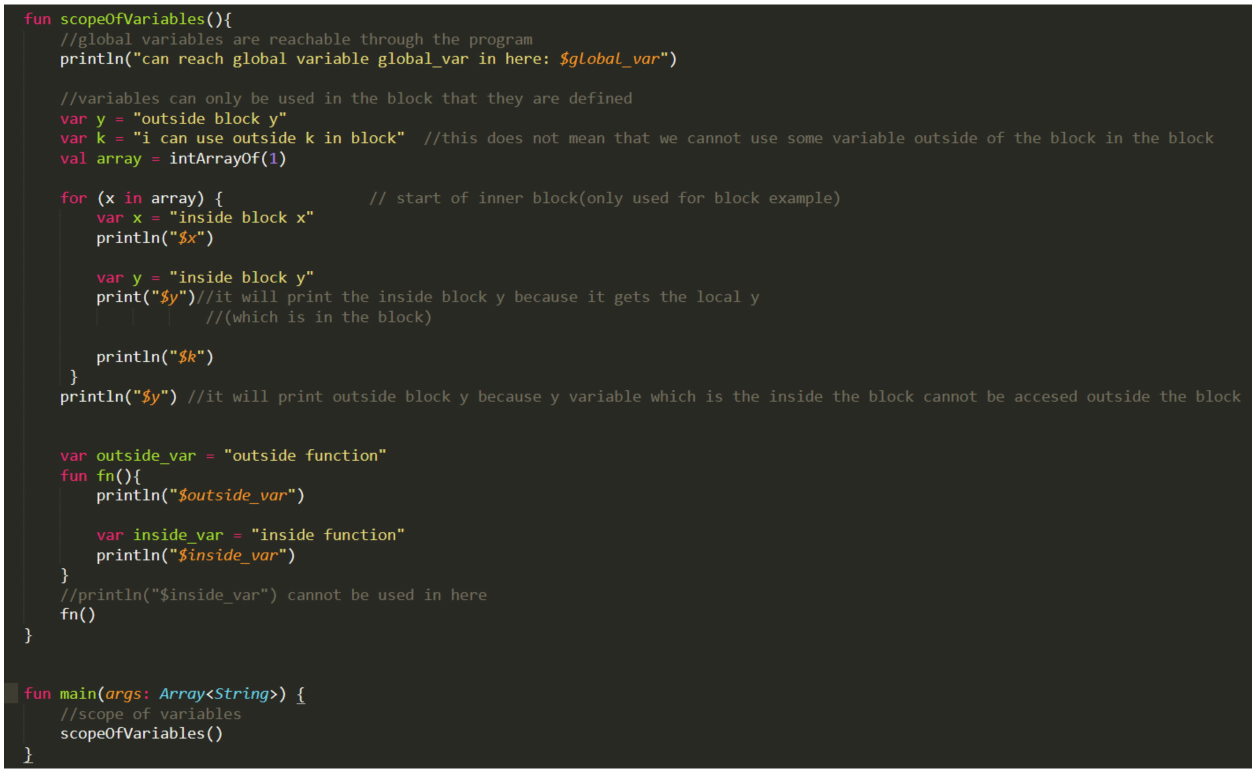
```
local offsets of main_var1, main_var2,
main_var3,main_var4 is 1, 1, 1, 1
1-local offset of fn1_var1 and fn1_var2 is 2
and 2
2-local offset of main_var1 is 2, chain offset
is 0
3-local offset of main_var2 is 2, chain offset
is 1
4-local offset of fn2_var1 is 3
5-local offset of main_var2 is 3, chain offset
is 2
6-local offset of main_var1 is 3, chain offset
is 1
7-fn_dynamic has local offset 3, chain offset 1
8-there is no dynamic chain between nested
subprograms
```

```
9- global variable has local offest 4, chain
offset of global is 4
```

- **Explanation of the code :** In this code example, the main aim is to understand the nested functions in Kotlin language by understanding static chains. Firstly, to observe the scope of the functions I create some variables in these functions. We have one global variable whose local offset is 0. Then, we have main_vars that have local offset 1. I assign the local offset values to variables in here to keep track of the chain offsets through the program. Then, we have a nested function fn1() whose static parent is nestedSubprograms. In fn1(), I also have two fn1 variables whose local offset is 2 and I have another local variable main_var1 that has the same name with one of the variables in its static parent nestedSubprograms. Since main_ar1 is considered as a local variable in fn1 and its local offset is 2, it has chain offset 0. However, main_var2 in fn1 is a nonlocal variable. Even if it has local offset 2 because of the fn1, its chain offset is 1. Then, I created a variable to understand that Kotlin language uses static or dynamic scoping in nested subprograms. Inside fn1, fn2 and fn3 are defined and all of their variables have local offset 3. Fn1 will call fn2. In fn2, I also put variables named fn_dynamic and then fn3. In this way, I will be able to check static chains. When fn2 call fn3, by using chain_offset5, we can determine the chain type. If it uses the fn1's fn_dynamic whose local offset is 2 (also value is 2), there is no dynamic scoping in Kotlin when nested subprogram implementation, otherwise it will use fn2's fn_dynamic variable whose local offset(also value is 1). When we check the result we observe that it uses the fn1's variable. Therefore, it is possible to say that Kotlin uses static chains in nested subprograms.
  Program call:
  nestedSubPrograms calls fn1
  Fn1 calls fn2
  Fn2 calls fn3
  Fn3 calls fn4

2. **Scope of local variables**

- **Explanation:** Kotlin variables have block scope and they can be used and accessed through the scope of the block they are defined after variable declaration.
- **Code Segment:**



```
fun scopeOfVariables(){
    //global variables are reachable through the program
    println("can reach global variable global_var in here: $global_var")

    //variables can only be used in the block that they are defined
    var y = "outside block y"
    var k = "i can use outside k in block"  //this does not mean that we cannot use some variable outside of the block in the block
    val array = intArrayOf(1)

    for (x in array) {                // start of inner block(only used for block example)
        var x = "inside block x"
        println("$x")

        var y = "inside block y"
        print("$y")//it will print the inside block y because it gets the local y
                //(which is in the block)

        println("$k")
     }
    println("$y") //it will print outside block y because y variable which is the inside the block cannot be accesed outside the block


    var outside_var = "outside function"
    fun fn(){
        println("$outside_var")

        var inside_var = "inside function"
        println("$inside_var")
    }
    //println("$inside_var") cannot be used in here
    fn()
}


fun main(args: Array<String>) {
    //scope of variables
    scopeOfVariables()
}
```

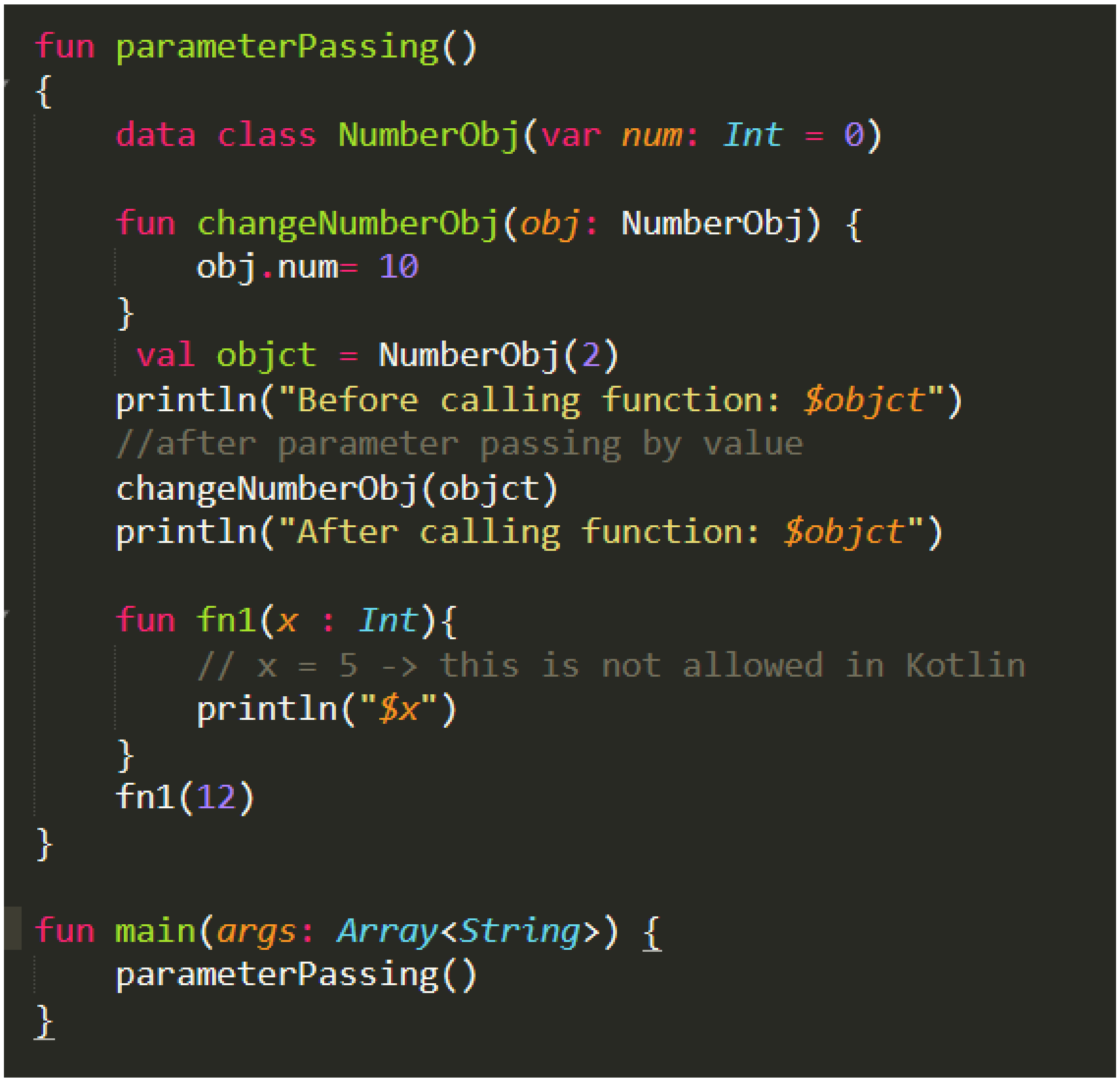
- **Result of Execution:**

```
can reach global variable global_var in here: 0
inside block x
inside block yi can use outside k in block
outside block y
outside function
inside function
```

- **Explanation of the code :** Kotlin variables have block scope. In other words, they can be used and accessed in the block they are defined from the point they are declared. However, there are also global variables. These variables can be referenced at any point of the program from the point they are declared. I used for block to show as an example here. We can use any variable in the for block that we are defined outside of that block. However, out of the block, we cannot use variables that we are declared inside the block.

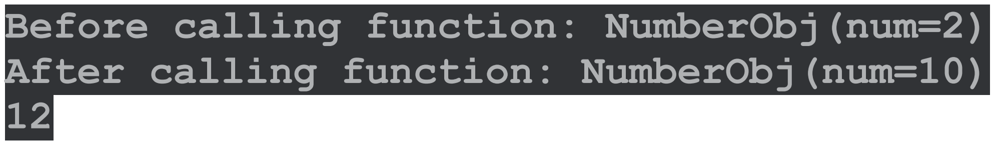
3. **Parameter passing methods**

- **Explanation:** In Kotlin, parameters can be passed to methods by using pass-by-value methods. However, object variables can only be passed by reference like in Java.
- **Code Segment:**



```
fun parameterPassing()
{
    data class NumberObj(var num: Int = 0)

    fun changeNumberObj(obj: NumberObj) {
        obj.num= 10
    }
     val objct = NumberObj(2)
    println("Before calling function: $objct")
    //after parameter passing by value
    changeNumberObj(objct)
    println("After calling function: $objct")

    fun fn1(x : Int){
        // x = 5 -> this is not allowed in Kotlin
        println("$x")
    }
    fn1(12)
}

fun main(args: Array<String>) {
    parameterPassing()
}
```

- **Result of Execution:**



```
Before calling function: NumberObj(num=2)
After calling function: NumberObj(num=10)
12
```

- **Explanation of the code :** Firstly, I want to test the pass objects as parameter to the methods in Kotlin. I created a class called NumberObj that has a single attribute and it is the number num. Then I wrote a function that takes this object as a parameter and changed the value of its num attribute. Then after calling that function, I realized that the attribute of this object is changed as seen in the result. This is a pass-by-reference feature for objects of Kotlin language. However, when I try to do it on Int x it gives compile errors. Firstly, Kotlin does not have primitive types but uses classes like Int as an object wrapper of primitive types. The value of this argument is passed by value to the function and they are immutable [2]. Therefore, we cannot reassign the parameter such as x = 5 in Kotlin.

4. **Keyword and default parameters**

- **Explanation:** Kotlin enables default parameters to the functions. These values should be assigned like in the following

  fun functionName (parameter1 : parameter Type,

  parameter2: parameter Type = defaultValue,

  Parameter3 : parameterType = defaultValue){...}

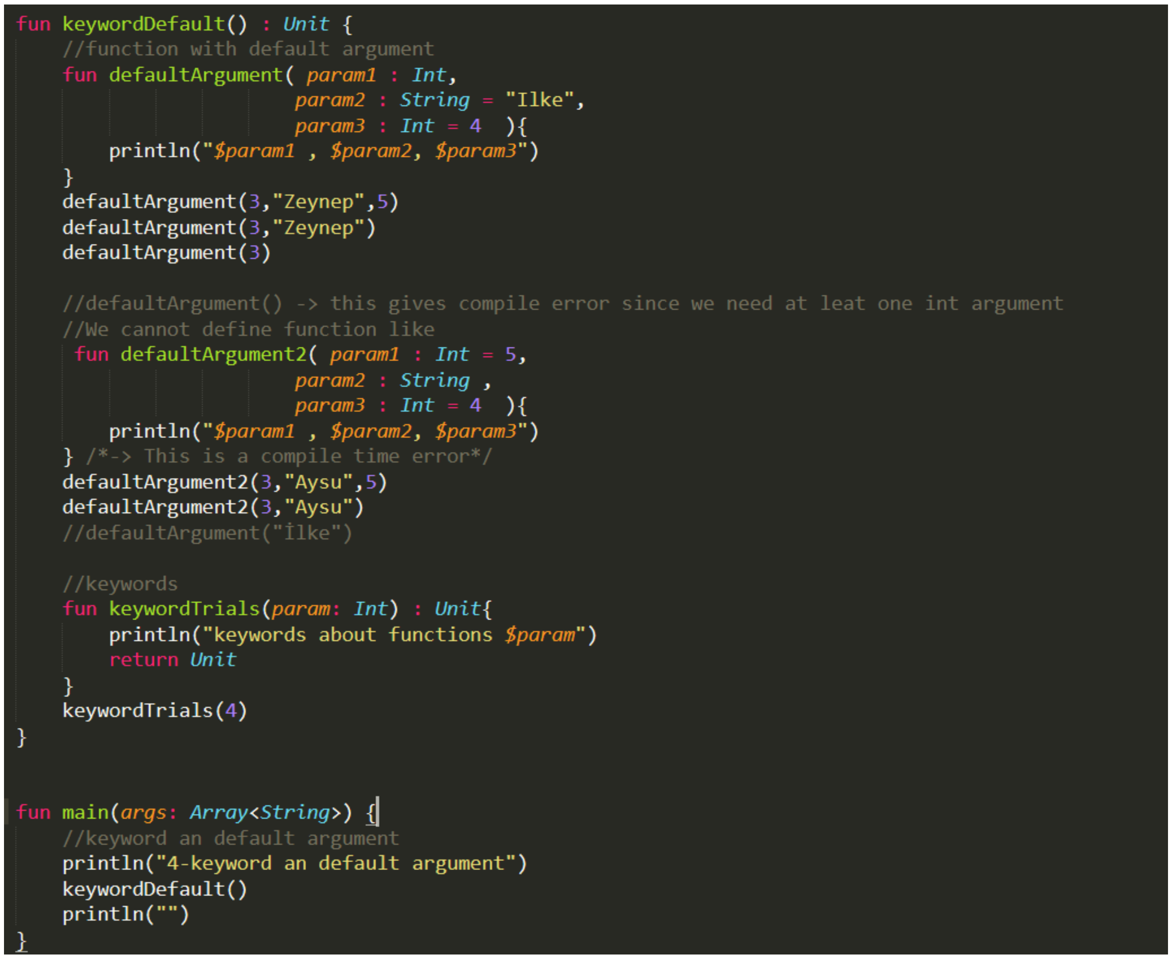
  Keywords about subprograms in Kotlin:

  fun-> fun declares a function

  return-> return from the function

  Unit -> is a class with a single value that is used to return meaningless values from functions such as void. It can be omitted when writing the functions.

- **Code Segment:**



```kotlin
fun keywordDefault() : Unit {
    //function with default argument
    fun defaultArgument( param1 : Int,
                         param2 : String = "Ilke",
                         param3 : Int = 4  ){
        println("$param1 , $param2, $param3")
    }
    defaultArgument(3,"Zeynep",5)
    defaultArgument(3,"Zeynep")
    defaultArgument(3)

    //defaultArgument() -> this gives compile error since we need at leat one int argument
    //We cannot define function like
     fun defaultArgument2( param1 : Int = 5,
                         param2 : String ,
                         param3 : Int = 4  ){
        println("$param1 , $param2, $param3")
    } /*-> This is a compile time error*/
    defaultArgument2(3,"Aysu",5)
    defaultArgument2(3,"Aysu")
    //defaultArgument("İlke")

    //keywords
    fun keywordTrials(param: Int) : Unit{
        println("keywords about functions $param")
        return Unit
    }
    keywordTrials(4)
}


fun main(args: Array<String>) {
    //keyword an default argument
    println("4-keyword an default argument")
    keywordDefault()
    println("")
}
```

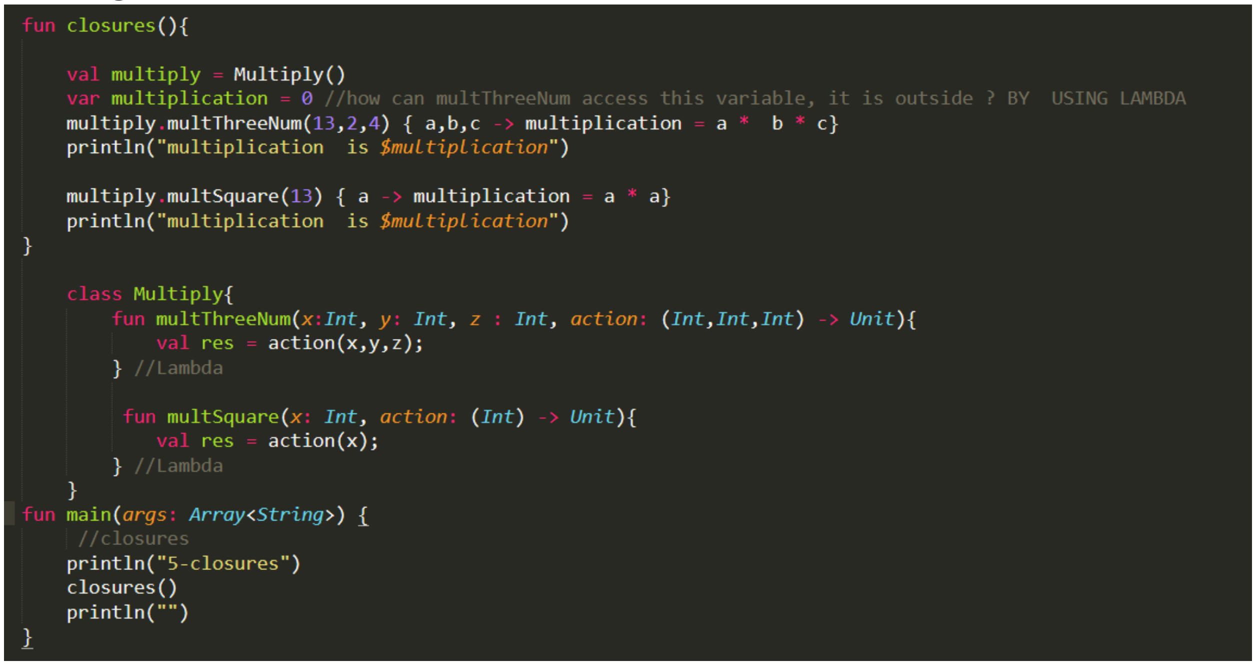
- **Result of Execution:**

```
4-keyword an default argument
3 , Zeynep, 5
3 , Zeynep, 4
3 , Ilke, 4
3 , Aysu, 5
3 , Aysu, 4
keywords about functions 4
```

- **Explanation of the code :** First function, default parameter is the generally written syntax. The parameters with default values are generally written rightmost. However, Kotlin compiler does not give errors if one does not write them rightmost as the defaultParamater2 function. However,in this situation, we cannot write one argument since it should be the second parameter. Also, if we try to write defaultParamater with no argument we will also get error. Last function keywordTrials is written to show keywords about functions in Kotlin. **fun** word is used to define functions, **Unit** is an object that has a single attribute and it is used for meaningless return values in Kotlin. It is not necessary to write it. For example, when we call keywordDefault from the main function we do not return anything but we do not use the return Unit statement.

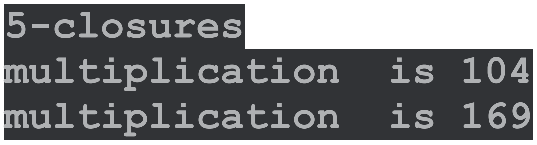
5. **Closures**

- **Explanation:** Closures contain the functions and the referencing environments that they are defined. They can access and change the values that are defined outside the scope of the subprograms. Since these functions have variables and closures can be used through the lifetime of the entire program, their variables also should gain unlimited extent [4]. In Kotlin, lambda expression can access its closure that contains the variables that are declared out of the function[5].
- **Code Segment for:**



```kotlin
fun closures(){

    val multiply = Multiply()
    var multiplication = 0 //how can multThreeNum access this variable, it is outside ? BY  USING LAMBDA
    multiply.multThreeNum(13,2,4) { a,b,c -> multiplication = a *  b * c}
    println("multiplication  is $multiplication")

    multiply.multSquare(13) { a -> multiplication = a * a}
    println("multiplication  is $multiplication")
}

    class Multiply{
        fun multThreeNum(x:Int, y: Int, z : Int, action: (Int,Int,Int) -> Unit){
            val res = action(x,y,z);
        } //Lambda

         fun multSquare(x: Int, action: (Int) -> Unit){
            val res = action(x);
        } //Lambda
    }
fun main(args: Array<String>) {
     //closures
    println("5-closures")
    closures()
    println("")
}
```

- **Result of Execution:**



```
5-closures
multiplication  is 104
multiplication  is 169
```

- **Explanation of the code :** In here first, I create a multiply class and write two functions in it: multThreeNum that multiplies the three numbers given and multSquare that gets the square of the given argument. On the closures function, there is a multiplication variable and normally, both of these functions are not able to change it by passing primitives (not actual primitives but Int, Double wrapper classes) as parameters. However, we can change the outside variable and this is closure.

Conclusion

1. **My Evaluation about language:** Firstly, since Kotlin supports nested subprograms it makes this language hard to read and write in ways such as keep tracking which function includes which one. For example, it is important to be aware of the static parents of functions and their closures. Secondly, since we cannot reassign the parameters of type Int, Double etc. in Kotlin functions, i think it decreases the writability of the language. However, it increases the reliability of the language since it does not allow the change the

parameter value. Nonetheless, I would prefer to pass parameters by value but the ability to reassign the parameter should be in the program. However, since we can use closure in Kotlin to change the outside variables, this issue can be solved even if the concept of closure is not easy to understand. On the other hand, we can easily change the values of objects easily by using pass-by-reference, this increases writability but decreases the reliability. Default parameters provide convenience to programmers by increasing the writability but it may decrease the readability of the program since if we use it too much it may be hard to keep track of the default values that are assigned to parameters.

2. **My learning Strategy:** In this homework, I will search the Internet to get necessary information about Kotlin language. Then by combining them with my other language backgrounds, I tried to test statements and conditions about subprograms. I also use trial and error as seen in my code segments. I compile various types of the codes and research again the reasons for the results from the Internet.
3. **Materials and tools I used:**
   [1]https://www.oreilly.com/library/view/kotlin-programming-by/9781788474542/e7b033a2-bdee-4675-8bab-01cf67fcc175.xhtml
   [2]https://blog.jetbrains.com/kotlin/2013/02/kotlin-m5-1/
   [3]https://kotlinfrompython.com/2018/08/17/closures-methods-static-methods-class-methods-lambdas-and-callbacks/
   [4] Concepts of Programming Languages
   [5]https://kotlinlang.org/docs/lambdas.html#closures

4. **Experiments I performed:**
   As seen from my code segments, there are commented statements. I tried possible variations of statements by using my previous language knowledge. For example, in C++ default parameters must be rightmost parameters. I test this in that language and observe that Kotlin compiler allows that issue.

5. **URL of online compilers:**
   Kotlin Playground: Edit, Run, Share Kotlin Code Online (kotlinlang.org)