In The Name of God

Shiraz University

Programming Languages : Kotlin

Teacher : Dr. Morteza Keshtkaran

Provided by : Hossein Dehghanipour

Fall 2019

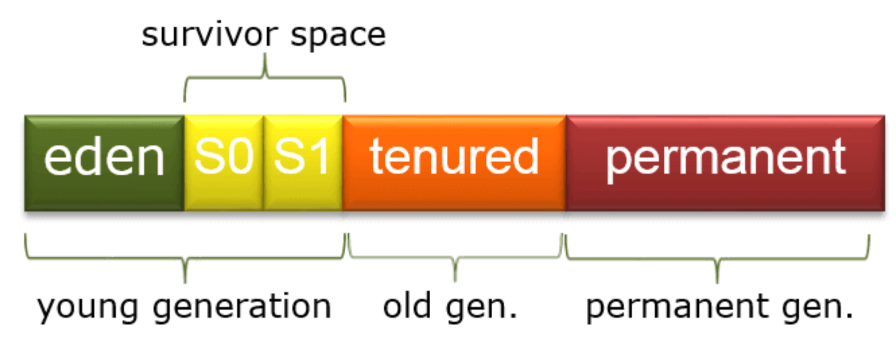
Introduction

- The syntax is pretty much similar to Java
- Kotlin is based on JVM
- Better performance and small runtime
- No Static Declaration – Kotlin does not have usual static handling modifier like Java

Garbage Collector :

Garbage Collection (5)

- Kotlin is run in JVM so it uses the same garbage collector as Java or any other JVM based language. Oracle's HotSpot is by far the most common GC .
- Java garbage collection is an automatic process. The programmer does not need to explicitly mark objects to be deleted. The garbage collection implementation lives in the JVM. Each JVM can implement garbage collection however it pleases.
- All of HotSpot's garbage collectors implement a generational garbage collection strategy that categorizes objects by age. The rationale behind generational garbage collection is that most objects are short-lived and will be ready for garbage collection soon after creation.

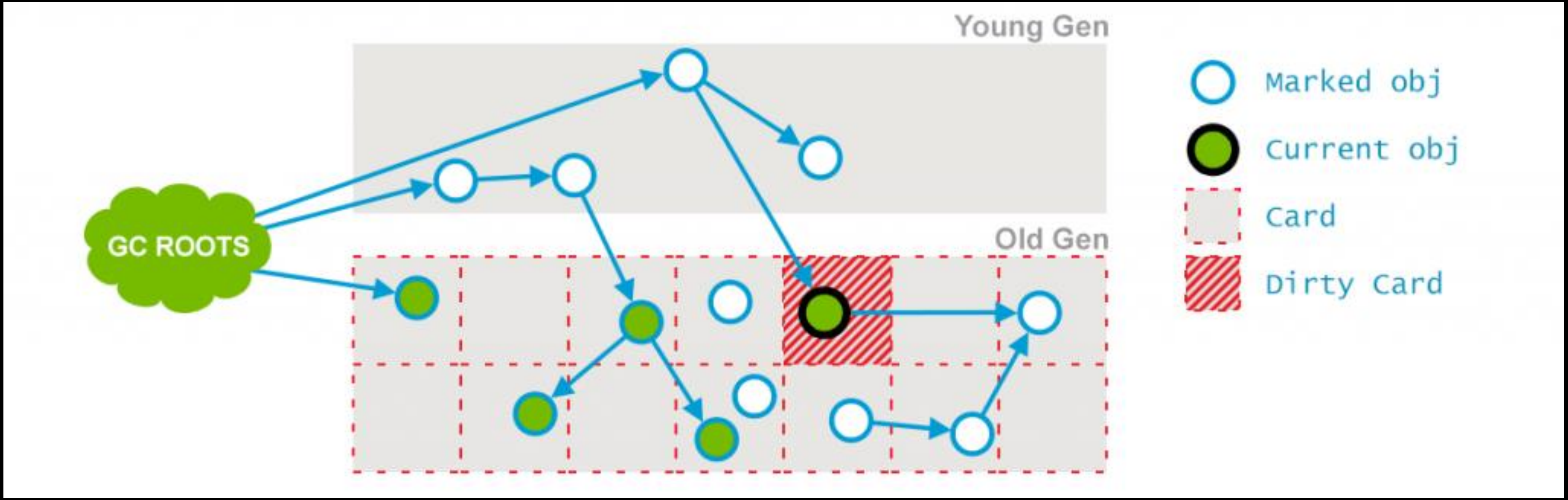


HotSpot GC :

- HotSpot has four garbage collectors:
- Serial: All garbage collection events are conducted serially in one thread. Compaction is executed after each garbage collection.
- Parallel: Multiple threads are used for minor garbage collection. A single thread is used for major garbage collection and Old Generation compaction. Alternatively, the Parallel Old variant uses multiple threads for major garbage collection and Old Generation compaction.
- CMS (Concurrent Mark Sweep): Multiple threads are used for minor garbage collection using the same algorithm as Parallel. Major garbage collection is multi-threaded, like Parallel Old, but CMS runs concurrently alongside application processes to minimize “stop the world” events (i.e. when the garbage collector running stops the application). No compaction is performed.
- G1 (Garbage First): The newest garbage collector is intended as a replacement for CMS. It is parallel and concurrent like CMS, but it works quite differently under the hood compared to the older garbage collectors.

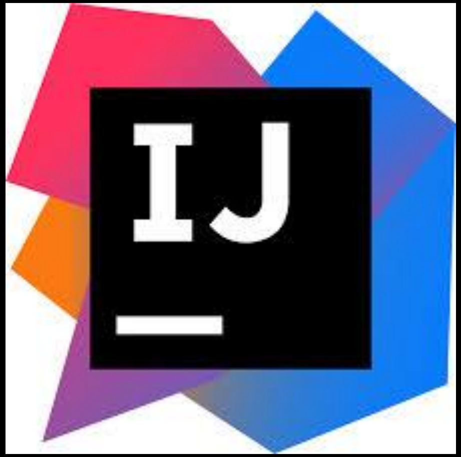
Kotlin In Android : GC

- Android uses the most common type of garbage collection, known as tracing garbage collection with the CMS algorithm. CMS stands for Concurrent Mark-Sweep.



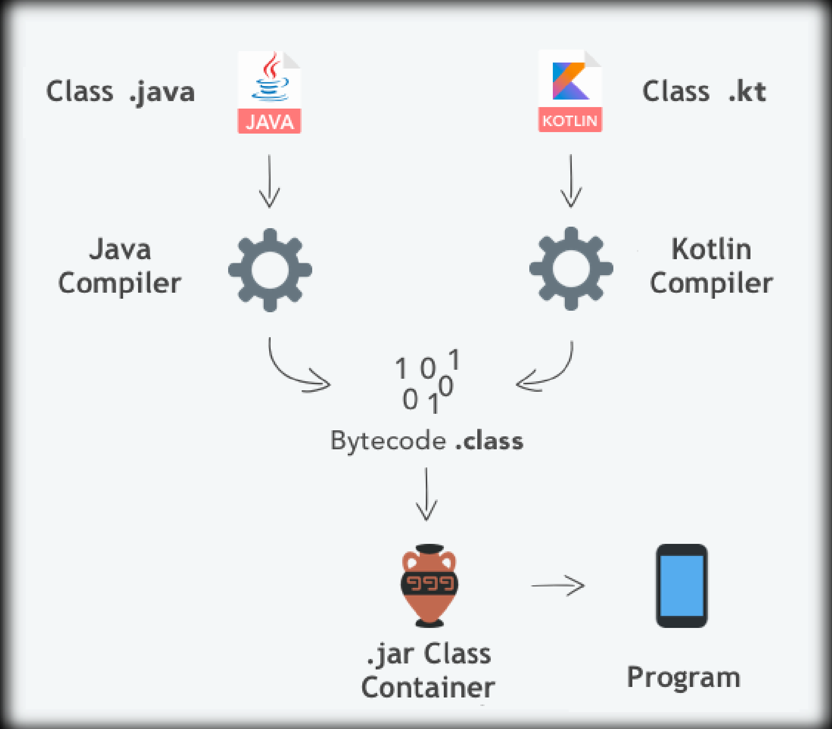
IDEs :

- Eclipse , Intellij , Android Studio , TryKotlin , Vim , Sublime Text

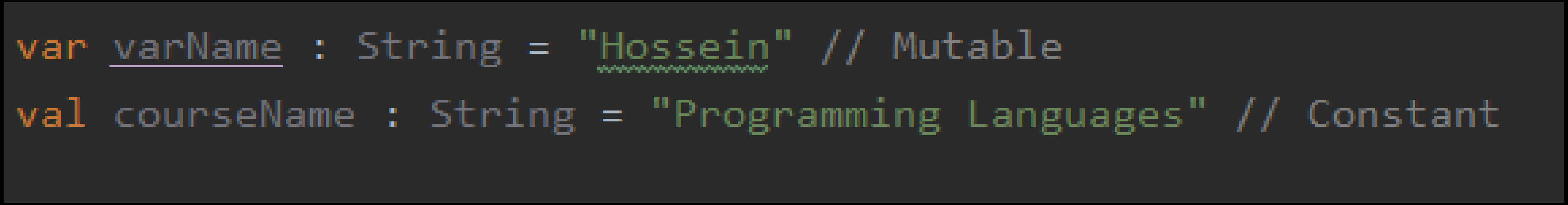


Compiling Process

- Kotlin compiler creates a byte code and that byte code can run on the JVM, which is exactly equal to the byte code generated by the Java .class file
- You can also use all Java libraries in Kotlin.

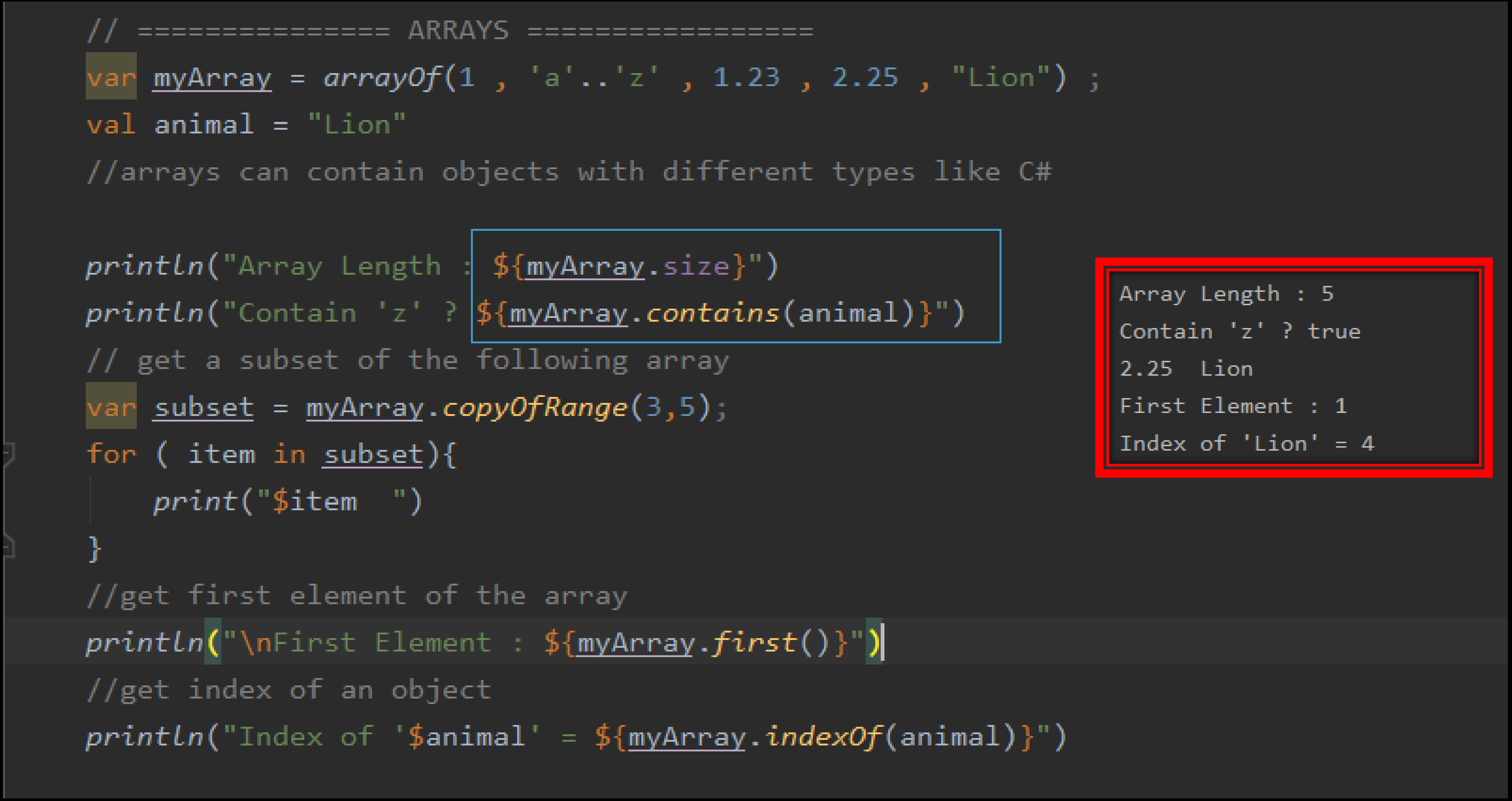


Variables



```
var varName : String = "Hossein" // Mutable
val courseName : String = "Programming Languages" // Constant
```

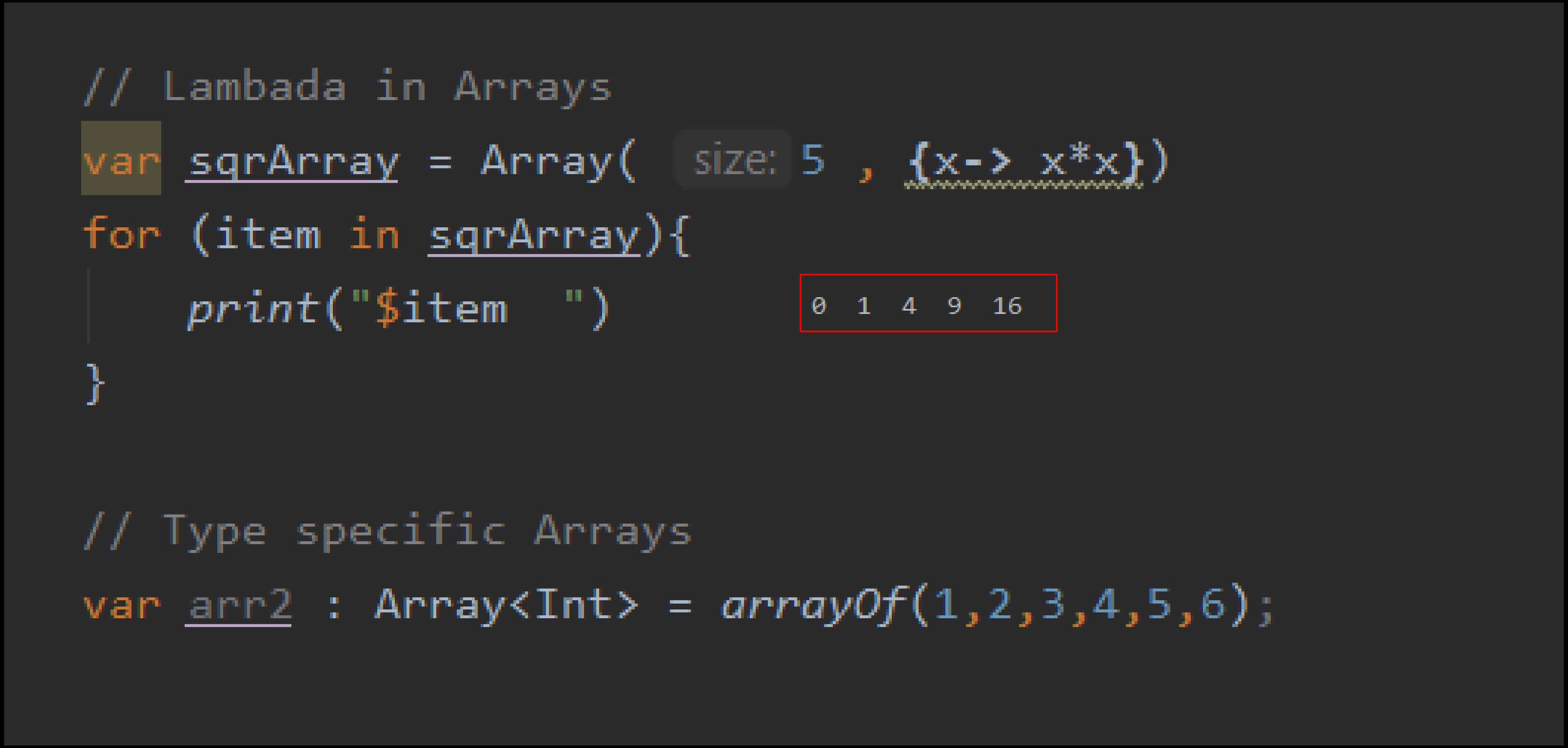
Arrays : Not Type Specified



```
// =============== ARRAYS =================
var myArray = arrayOf(1 , 'a'..'z' , 1.23 , 2.25 , "Lion") ;
val animal = "Lion"
//arrays can contain objects with different types like C#

println("Array Length : ${myArray.size}")
println("Contain 'z' ? ${myArray.contains(animal)}")
// get a subset of the following array
var subset = myArray.copyOfRange(3,5);
for ( item in subset){
    print("$item  ")
}
//get first element of the array
println("\nFirst Element : ${myArray.first()}")
//get index of an object
println("Index of '$animal' = ${myArray.indexOf(animal)}")
```

```
Array Length : 5
Contain 'z' ? true
2.25  Lion
First Element : 1
Index of 'Lion' = 4
```

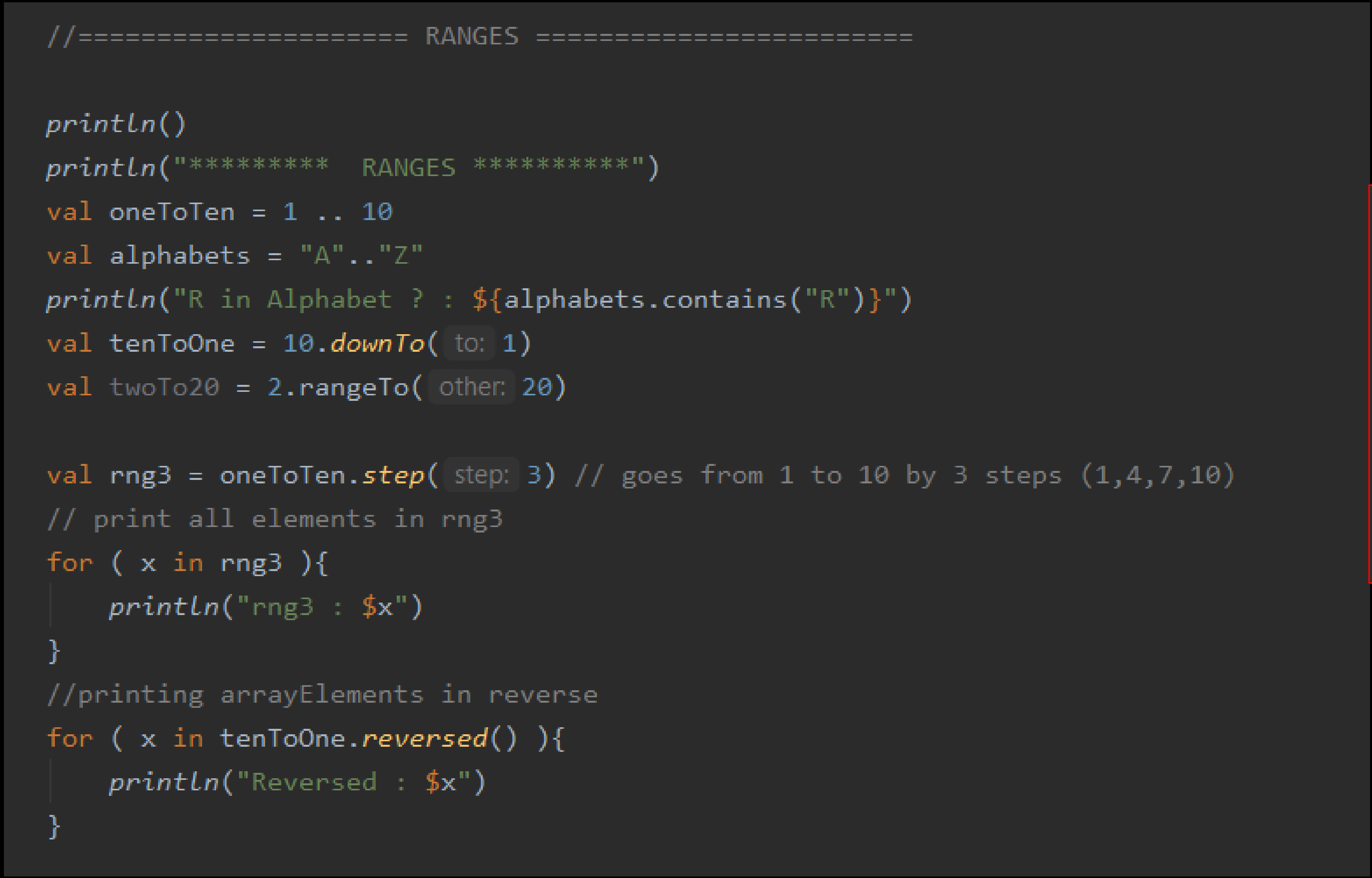
Arrays -> Type Specification + lambda



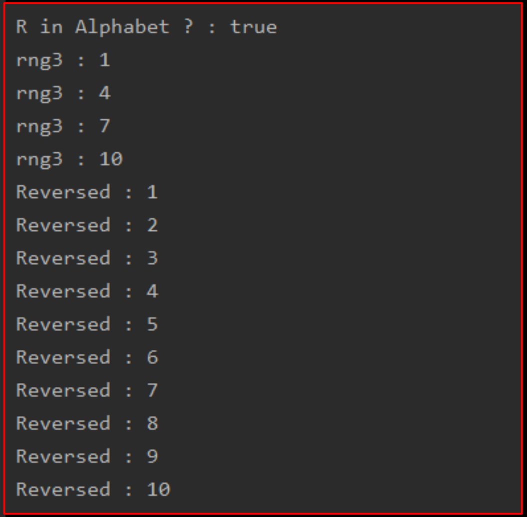
```
// Lambada in Arrays
var sqrArray = Array( size: 5 , {x-> x*x})
for (item in sqrArray){
    print("$item  ")
}

// Type specific Arrays
var arr2 : Array<Int> = arrayOf(1,2,3,4,5,6);
```

```
0  1  4  9  16
```

Ranges



```kotlin
//===================== RANGES ========================

println()
println("*********  RANGES **********")
val oneToTen = 1 .. 10
val alphabets = "A".."Z"
println("R in Alphabet ? : ${alphabets.contains("R")}")
val tenToOne = 10.downTo( to: 1)
val twoTo20 = 2.rangeTo( other: 20)

val rng3 = oneToTen.step( step: 3) // goes from 1 to 10 by 3 steps (1,4,7,10)
// print all elements in rng3
for ( x in rng3 ){
    println("rng3 : $x")
}
//printing arrayElements in reverse
for ( x in tenToOne.reversed() ){
    println("Reversed : $x")
}
```

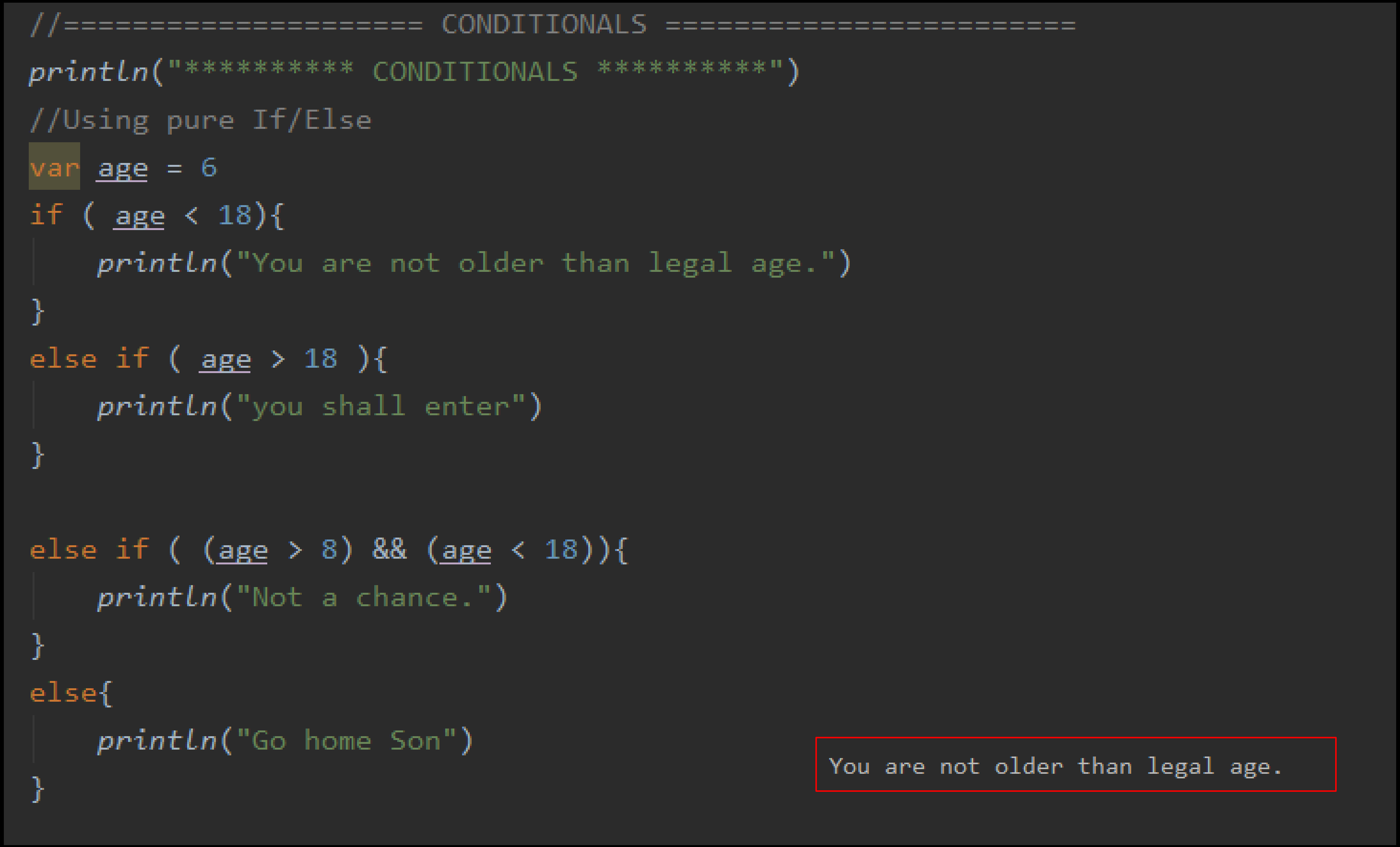


```
R in Alphabet ? : true
rng3 : 1
rng3 : 4
rng3 : 7
rng3 : 10
Reversed : 1
Reversed : 2
Reversed : 3
Reversed : 4
Reversed : 5
Reversed : 6
Reversed : 7
Reversed : 8
Reversed : 9
Reversed : 10
```

Control Flow

- If/Else
- When/Else

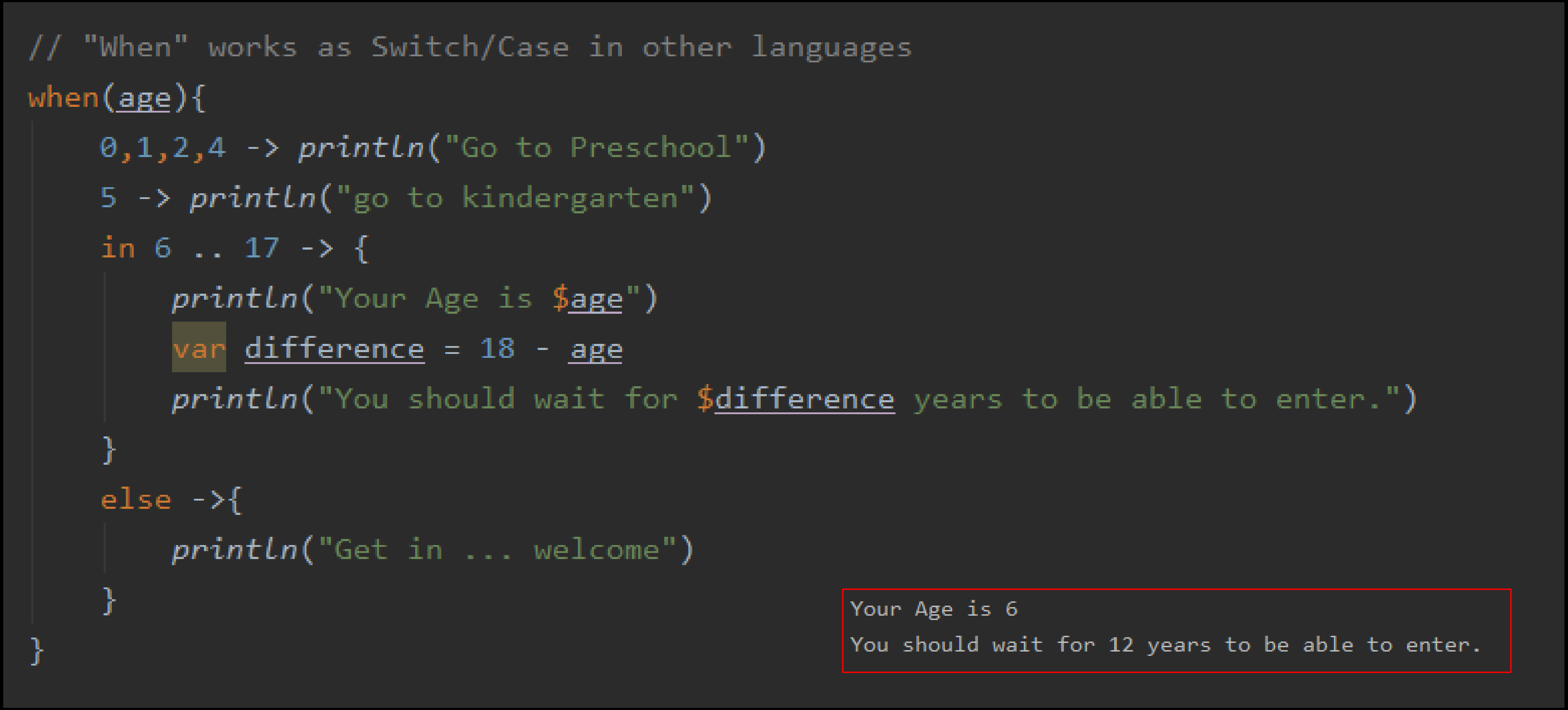
IF/Else



```
//===================== CONDITIONALS ========================
println("********** CONDITIONALS **********")
//Using pure If/Else
var age = 6
if ( age < 18){
    println("You are not older than legal age.")
}
else if ( age > 18 ){
    println("you shall enter")
}

else if ( (age > 8) && (age < 18)){
    println("Not a chance.")
}
else{
    println("Go home Son")
}
```

```
You are not older than legal age.
```

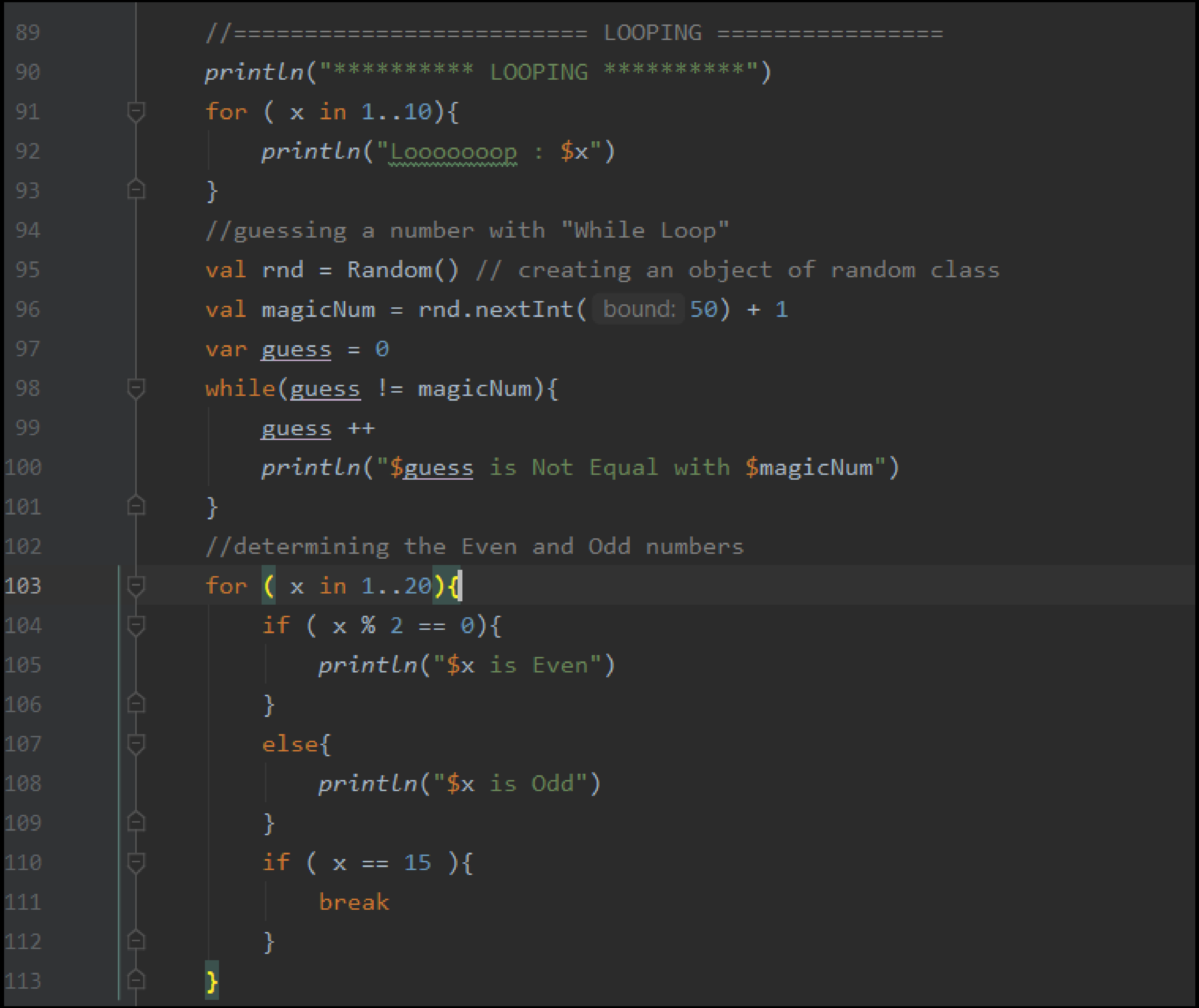
When



```
// "When" works as Switch/Case in other languages
when(age){
    0,1,2,4 -> println("Go to Preschool")
    5 -> println("go to kindergarten")
    in 6 .. 17 -> {
        println("Your Age is $age")
        var difference = 18 - age
        println("You should wait for $difference years to be able to enter.")
    }
    else ->{
        println("Get in ... welcome")
    }
}
```

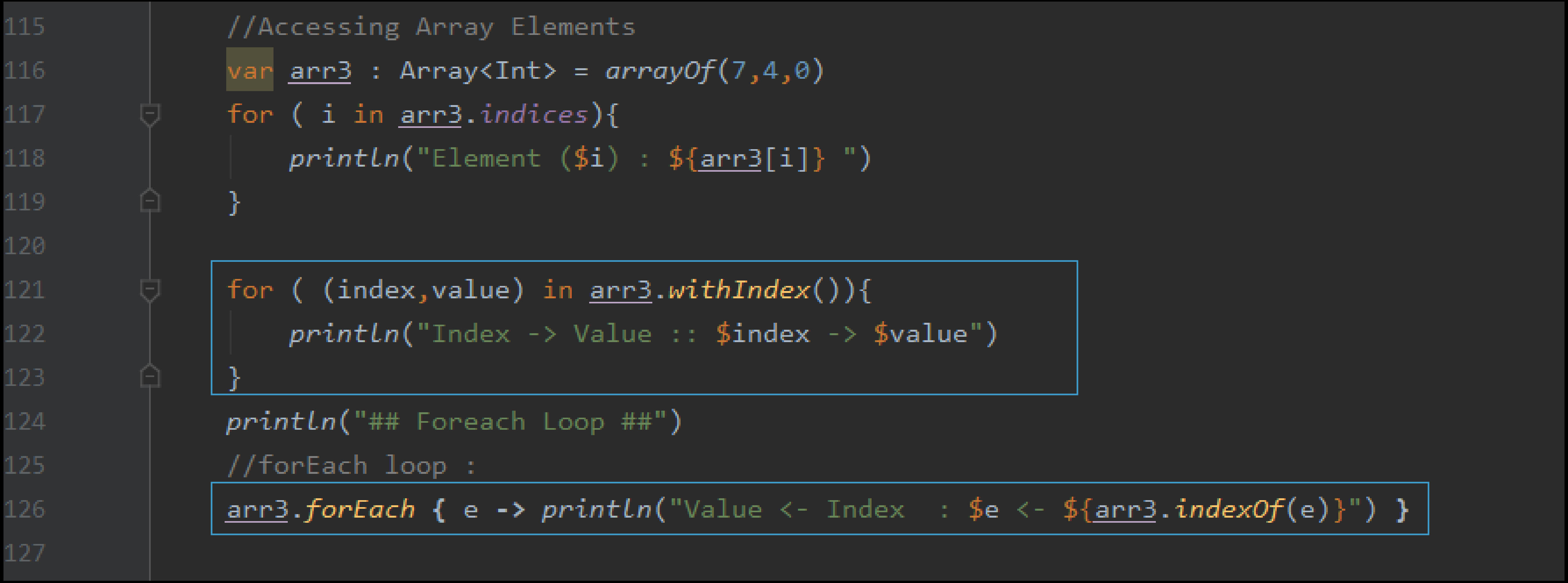
```
Your Age is 6
You should wait for 12 years to be able to enter.
```

Loops



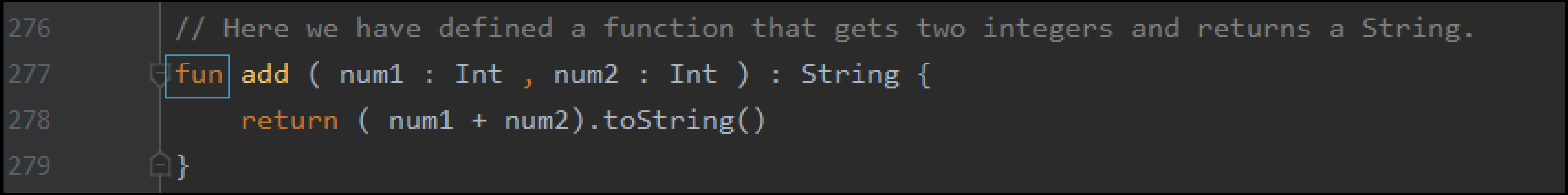
```
//======================== LOOPING ================
println("********** LOOPING **********")
for ( x in 1..10){
    println("Looooooop : $x")
}
//guessing a number with "While Loop"
val rnd = Random() // creating an object of random class
val magicNum = rnd.nextInt( bound: 50) + 1
var guess = 0
while(guess != magicNum){
    guess ++
    println("$guess is Not Equal with $magicNum")
}
//determining the Even and Odd numbers
for ( x in 1..20){
    if ( x % 2 == 0){
        println("$x is Even")
    }
    else{
        println("$x is Odd")
    }
    if ( x == 15 ){
        break
    }
}
```

Accessing Array Elements

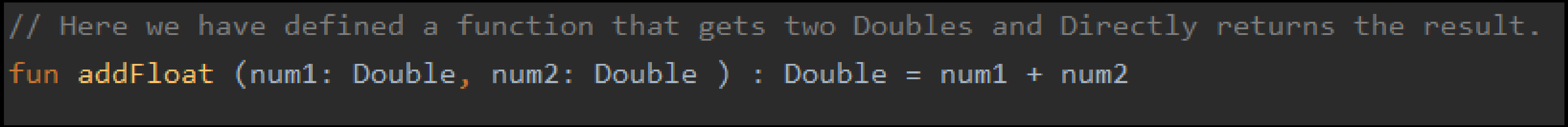


```kotlin
//Accessing Array Elements
var arr3 : Array<Int> = arrayOf(7,4,0)
for ( i in arr3.indices){
    println("Element ($i) : ${arr3[i]} ")
}

for ( (index,value) in arr3.withIndex()){
    println("Index -> Value :: $index -> $value")
}
println("## Foreach Loop ##")
//forEach loop :
arr3.forEach { e -> println("Value <- Index  : $e <- ${arr3.indexOf(e)}") }
```

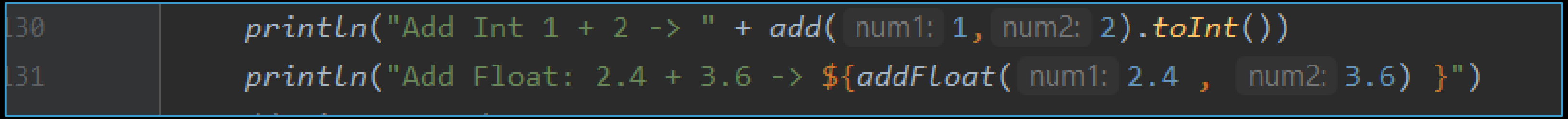
Functions -> نحوه ی تعریف تابع(6)



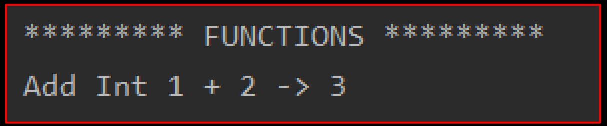
```kotlin
// Here we have defined a function that gets two integers and returns a String.
fun add ( num1 : Int , num2 : Int ) : String {
    return ( num1 + num2).toString()
}
```



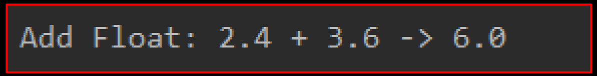
```kotlin
// Here we have defined a function that gets two Doubles and Directly returns the result.
fun addFloat (num1: Double, num2: Double ) : Double = num1 + num2
```



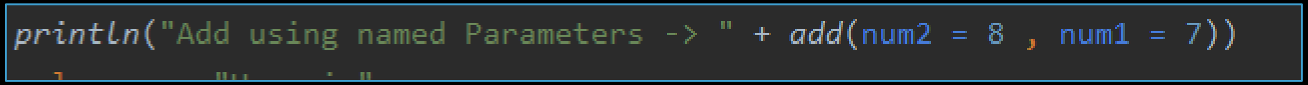
```kotlin
        println("Add Int 1 + 2 -> " + add( num1: 1, num2: 2).toInt())
        println("Add Float: 2.4 + 3.6 -> ${addFloat( num1: 2.4 , num2: 3.6) }")
```



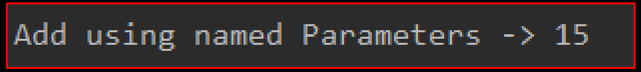
```
********* FUNCTIONS *********
Add Int 1 + 2 -> 3
```



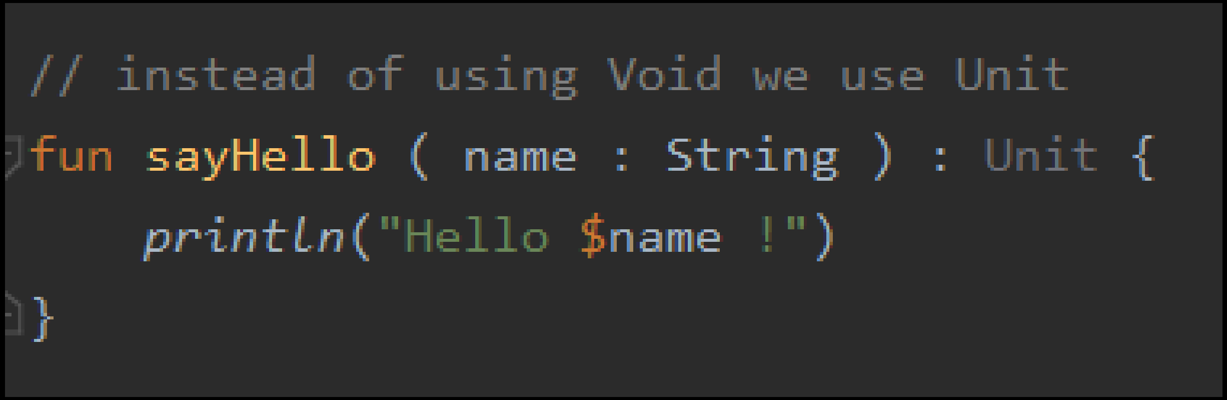
```
Add Float: 2.4 + 3.6 -> 6.0
```



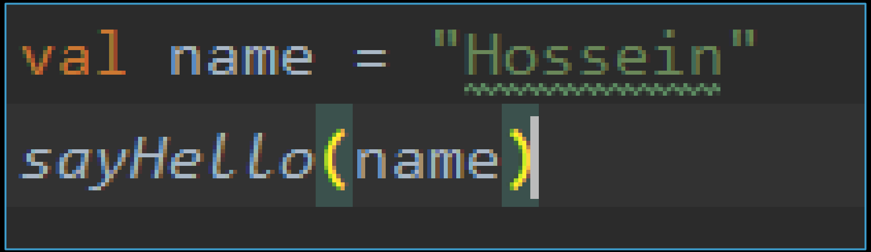
```kotlin
println("Add using named Parameters -> " + add(num2 = 8 , num1 = 7))
```



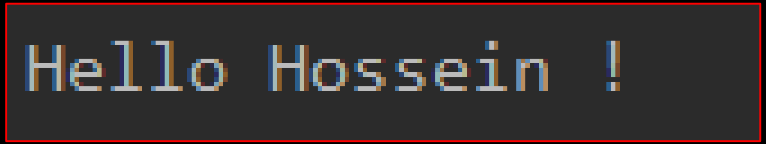
```
Add using named Parameters -> 15
```



```kotlin
// instead of using Void we use Unit
fun sayHello ( name : String ) : Unit {
    println("Hello $name !")
}
```

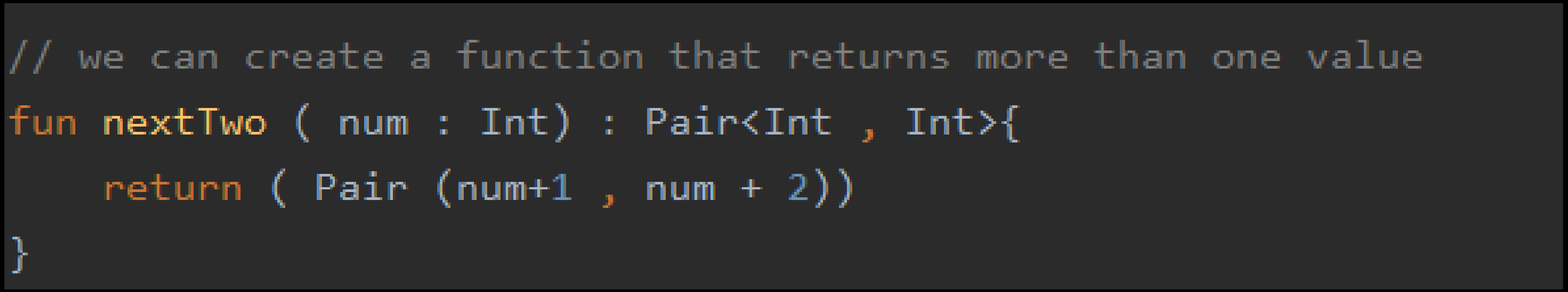


```kotlin
val name = "Hossein"
sayHello(name)
```

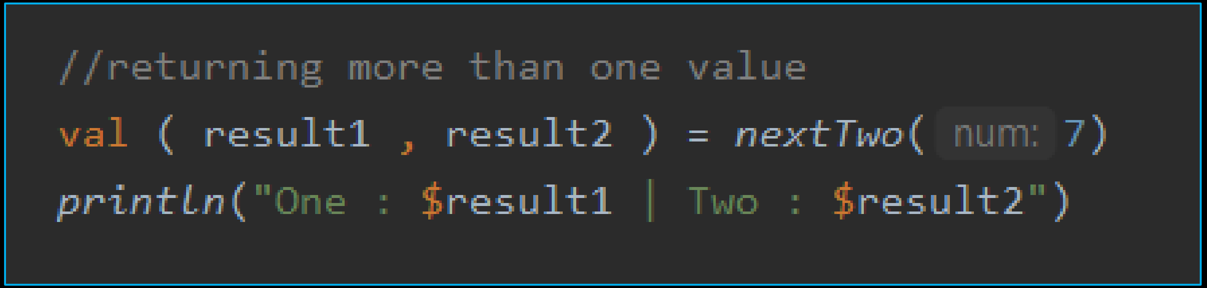


```
Hello Hossein !
```

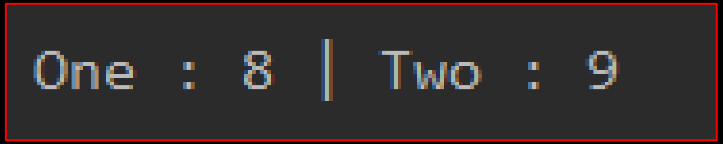
Functions (II)



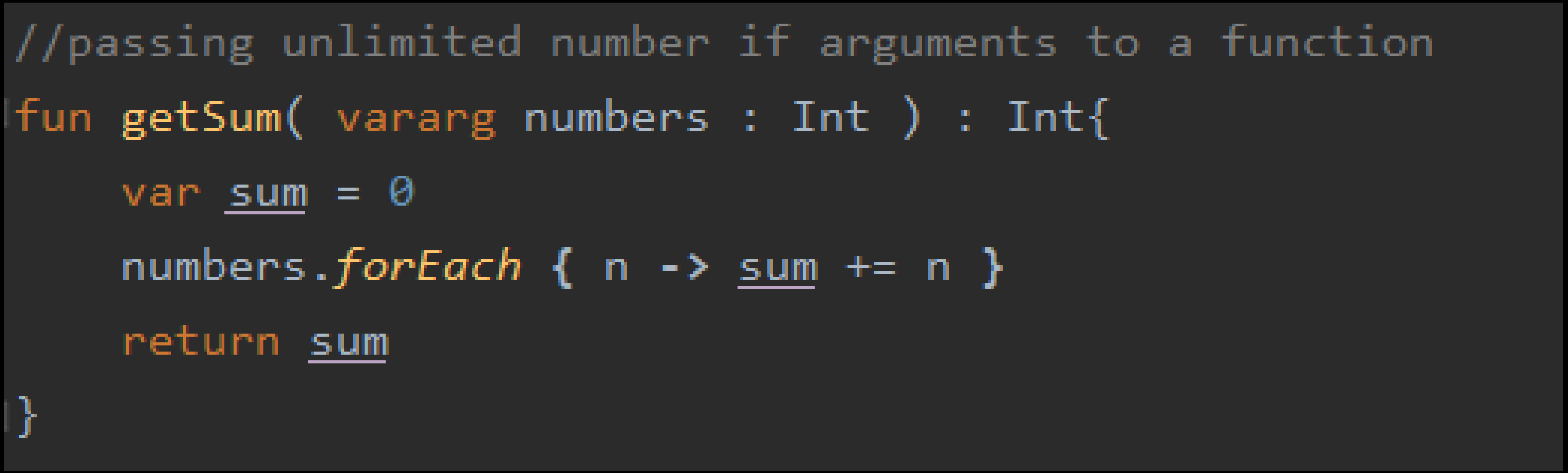
```
// we can create a function that returns more than one value
fun nextTwo ( num : Int) : Pair<Int , Int>{
    return ( Pair (num+1 , num + 2))
}
```



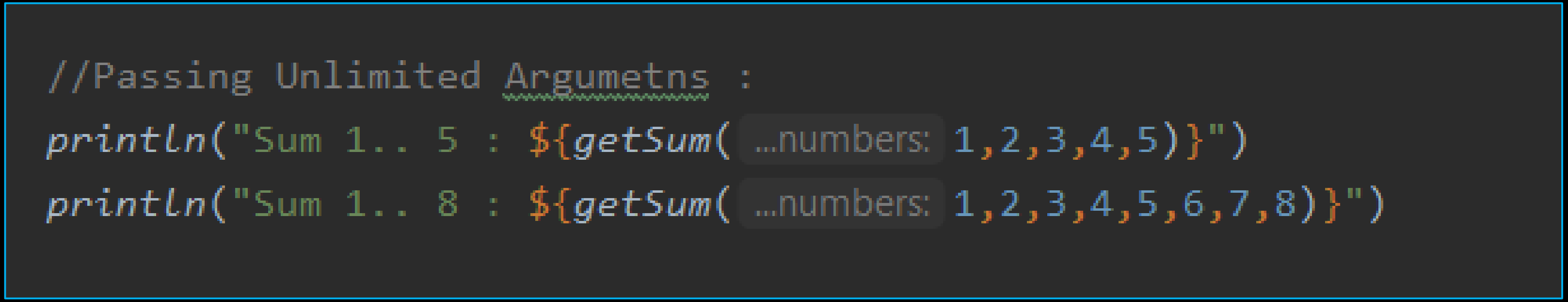
```
//returning more than one value
val ( result1 , result2 ) = nextTwo( num: 7)
println("One : $result1 | Two : $result2")
```



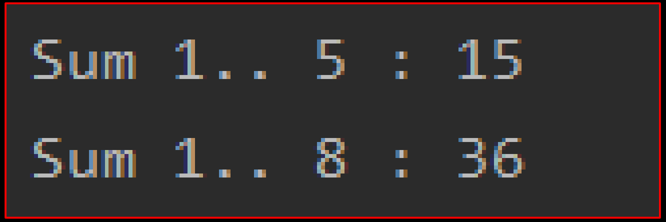
```
One : 8 | Two : 9
```



```kotlin
//passing unlimited number if arguments to a function
fun getSum( vararg numbers : Int ) : Int{
    var sum = 0
    numbers.forEach { n -> sum += n }
    return sum
}
```



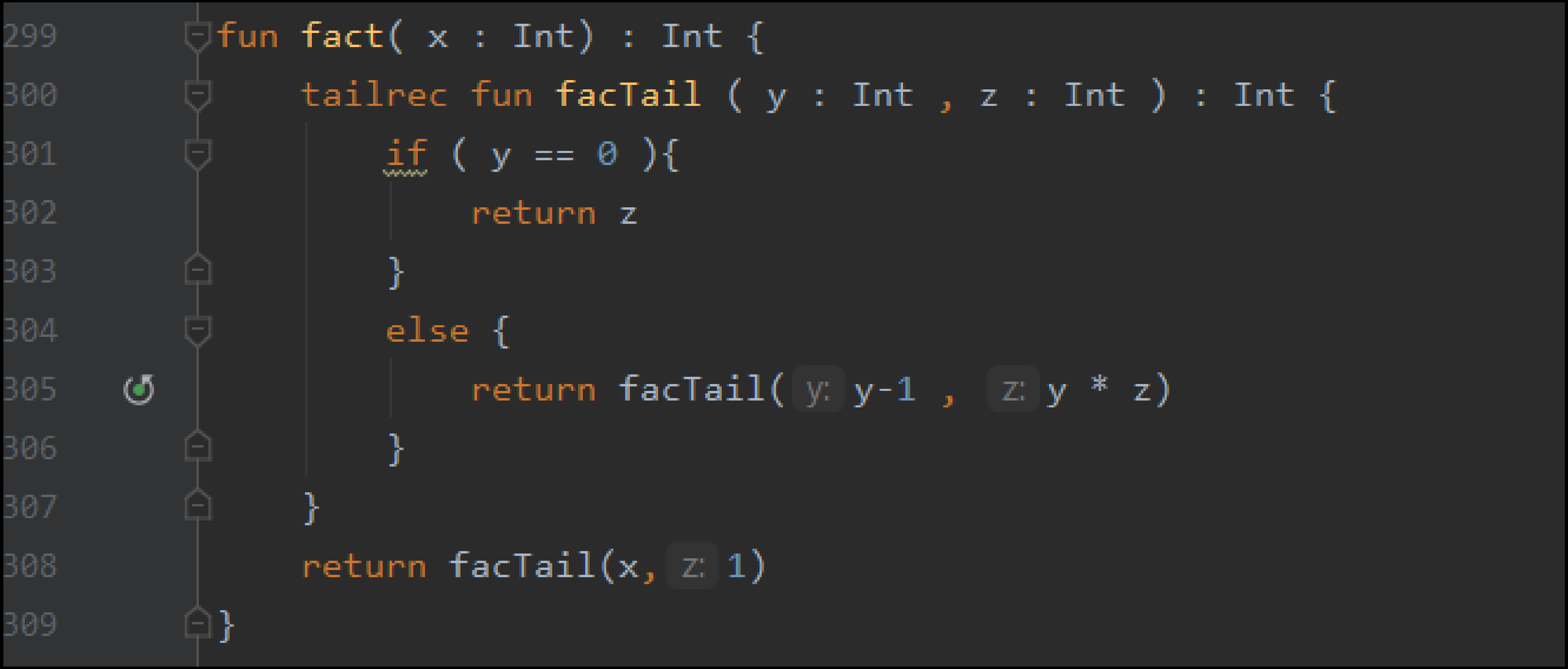
```kotlin
//Passing Unlimited Argumetns :
println("Sum 1.. 5 : ${getSum( …numbers: 1,2,3,4,5)}")
println("Sum 1.. 8 : ${getSum( …numbers: 1,2,3,4,5,6,7,8)}")
```



```
Sum 1.. 5 : 15
Sum 1.. 8 : 36
```

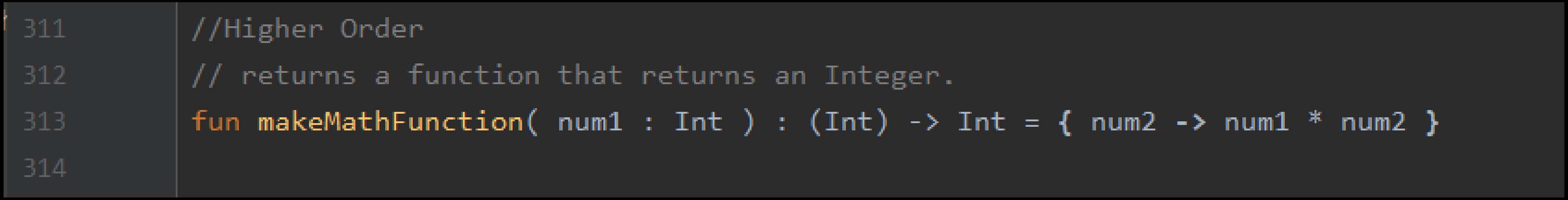
(9)Variable number of parameters

Tail Recursive Functions :

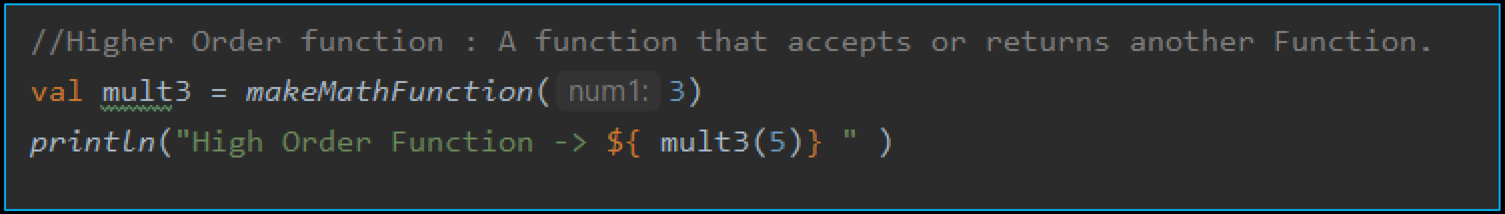


```
fun fact( x : Int) : Int {
    tailrec fun facTail ( y : Int , z : Int ) : Int {
        if ( y == 0 ){
            return z
        }
        else {
            return facTail( y: y-1 , z: y * z)
        }
    }
    return facTail(x, z: 1)
}
```

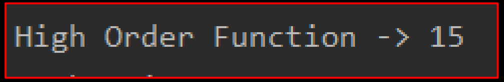
Higher Order Functions



```kotlin
//Higher Order
// returns a function that returns an Integer.
fun makeMathFunction( num1 : Int ) : (Int) -> Int = { num2 -> num1 * num2 }

```

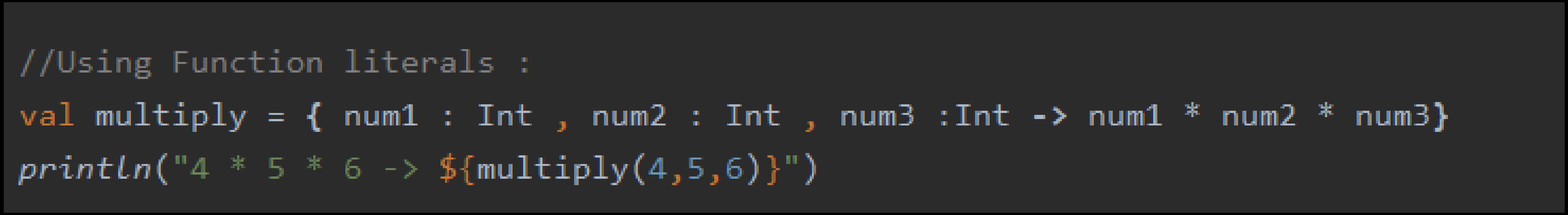


```kotlin
//Higher Order function : A function that accepts or returns another Function.
val mult3 = makeMathFunction( num1: 3)
println("High Order Function -> ${ mult3(5)} " )
```

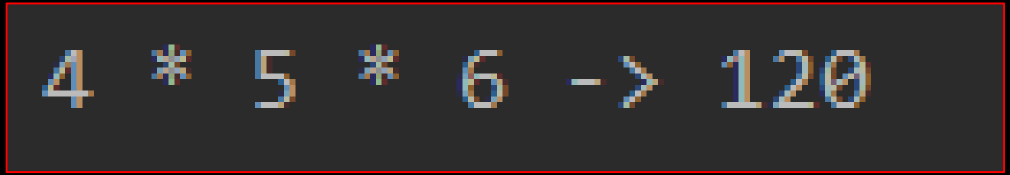


```
High Order Function -> 15
```

Lambda

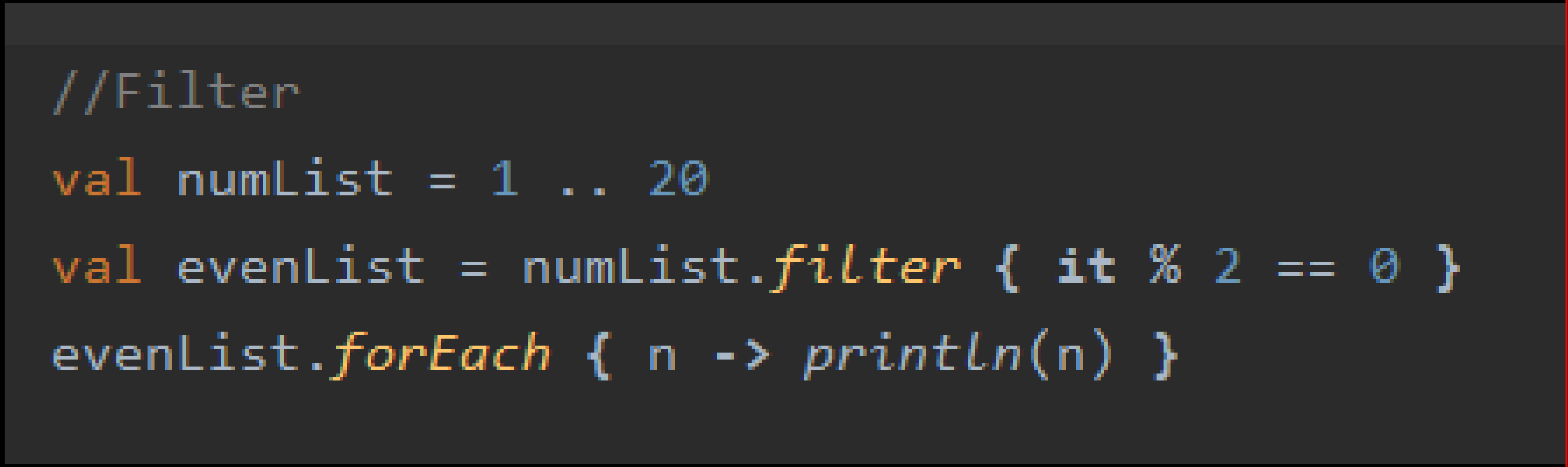


```
//Using Function literals :
val multiply = { num1 : Int , num2 : Int , num3 :Int -> num1 * num2 * num3}
println("4 * 5 * 6 -> ${multiply(4,5,6)}")
```

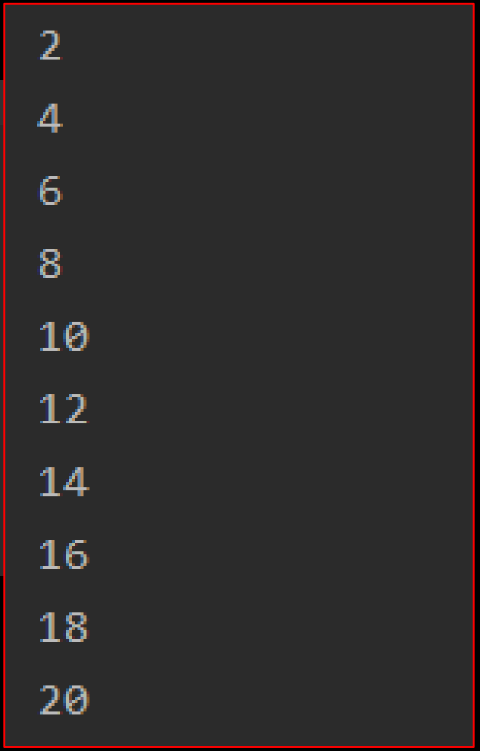


```
4 * 5 * 6 -> 120
```

Filter

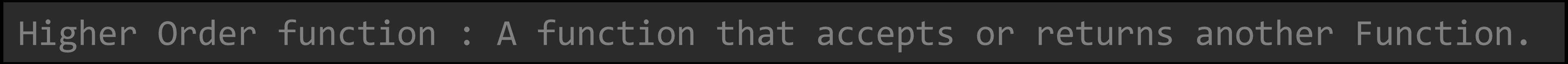


```
//Filter
val numList = 1 .. 20
val evenList = numList.filter { it % 2 == 0 }
evenList.forEach { n -> println(n) }
```

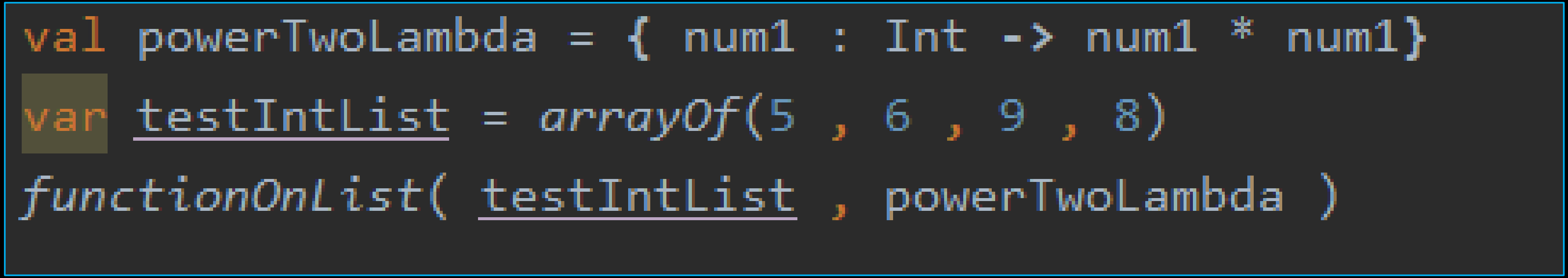


```
2
4
6
8
10
12
14
16
18
20
```

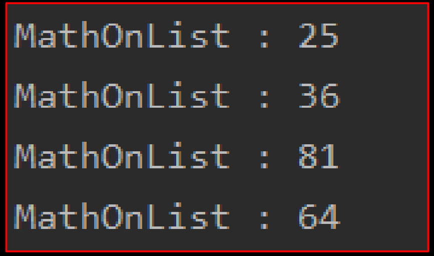
What is a Higher Order Function ?



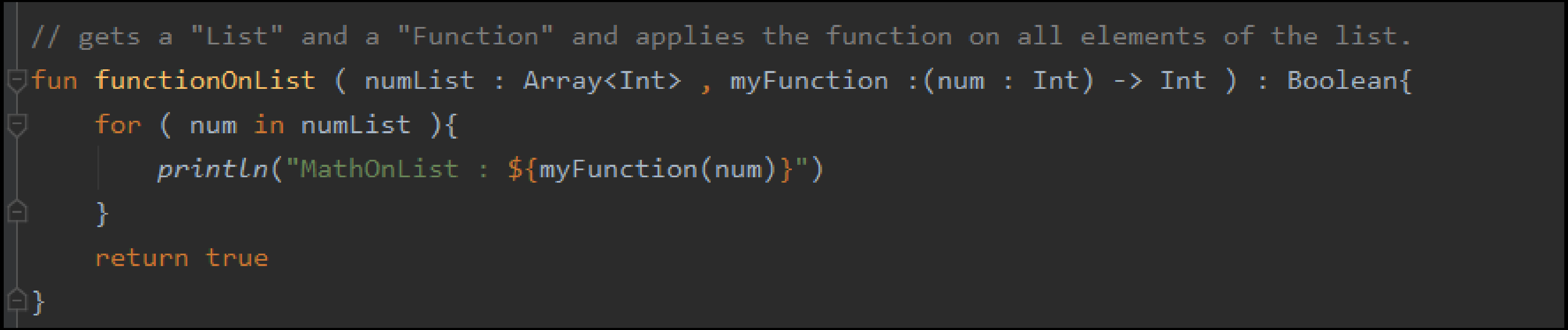
```
Higher Order function : A function that accepts or returns another Function.
```



```kotlin
val powerTwoLambda = { num1 : Int -> num1 * num1}
var testIntList = arrayOf(5 , 6 , 9 , 8)
functionOnList( testIntList , powerTwoLambda )
```



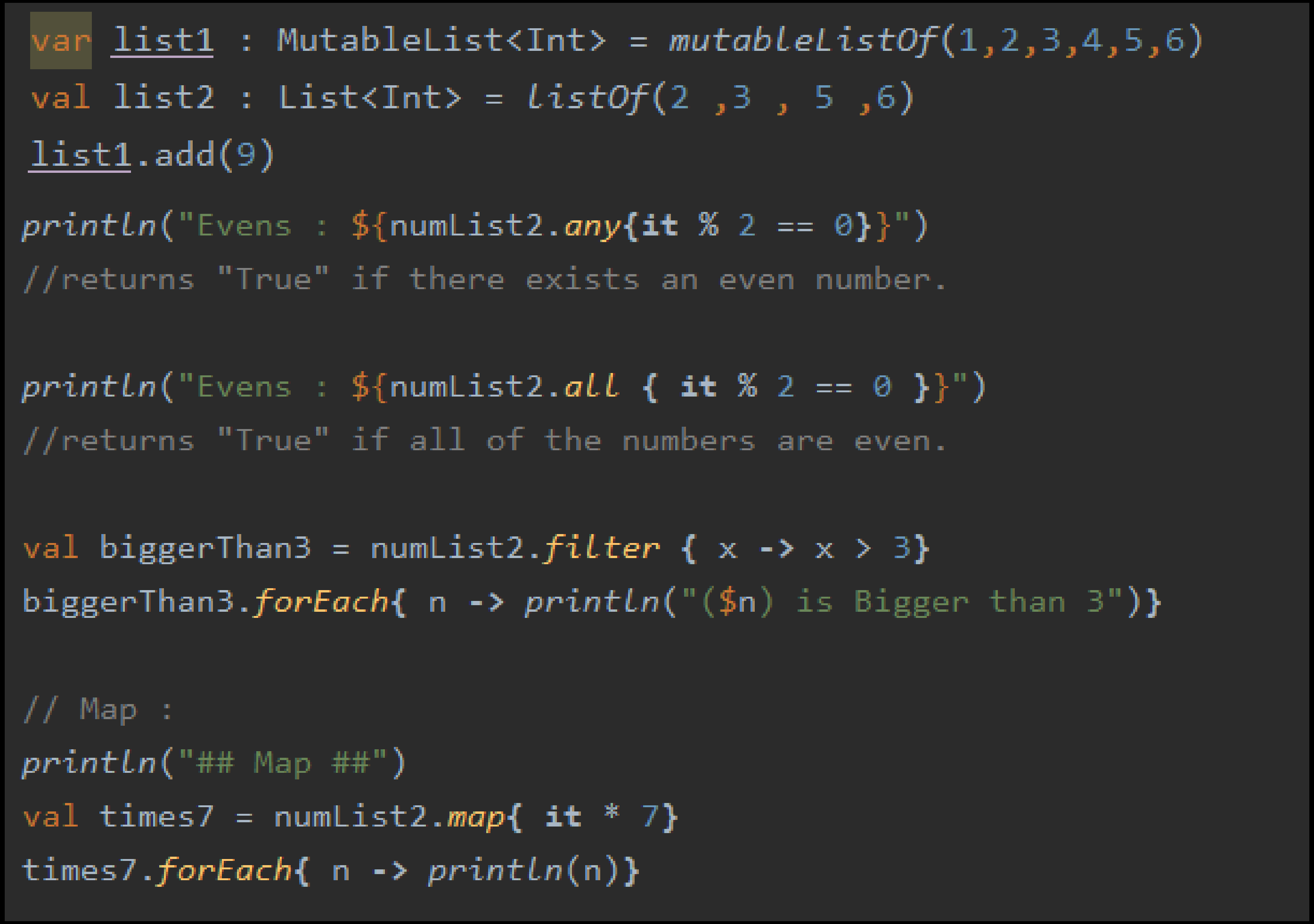
```
MathOnList : 25
MathOnList : 36
MathOnList : 81
MathOnList : 64
```



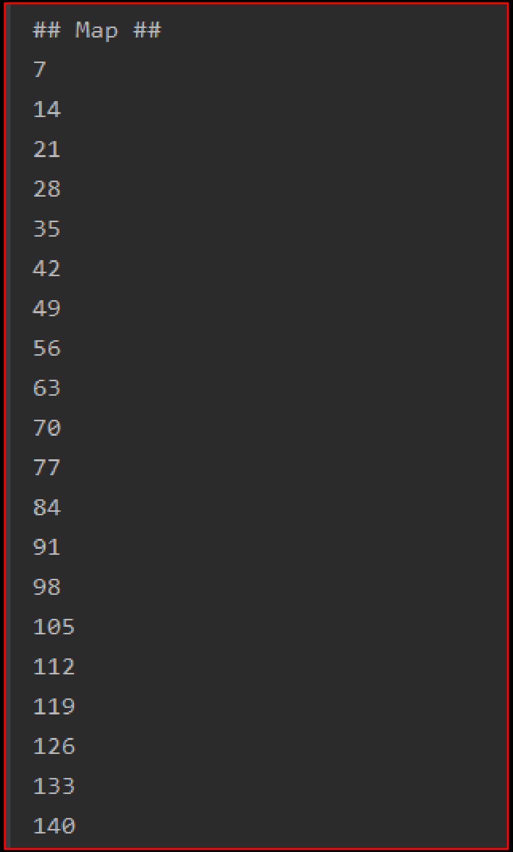
```kotlin
// gets a "List" and a "Function" and applies the function on all elements of the list.
fun functionOnList ( numList : Array<Int> , myFunction :(num : Int) -> Int ) : Boolean{
    for ( num in numList ){
        println("MathOnList : ${myFunction(num)}")
    }
    return true
}
```

ارسال تابع به عنوان
پارامتر تابع دیگر
(11)

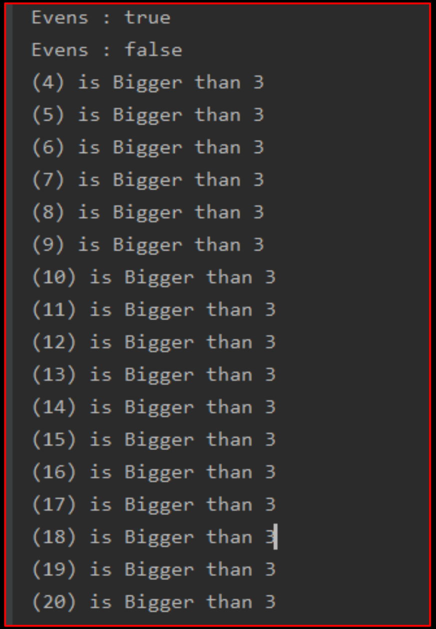
Collection Methods



```
var list1 : MutableList<Int> = mutableListOf(1,2,3,4,5,6)
val list2 : List<Int> = listOf(2 ,3 , 5 ,6)
list1.add(9)
println("Evens : ${numList2.any{it % 2 == 0}}")
//returns "True" if there exists an even number.

println("Evens : ${numList2.all { it % 2 == 0 }}")
//returns "True" if all of the numbers are even.

val biggerThan3 = numList2.filter { x -> x > 3}
biggerThan3.forEach{ n -> println("($n) is Bigger than 3")}

// Map :
println("## Map ##")
val times7 = numList2.map{ it * 7}
times7.forEach{ n -> println(n)}
```

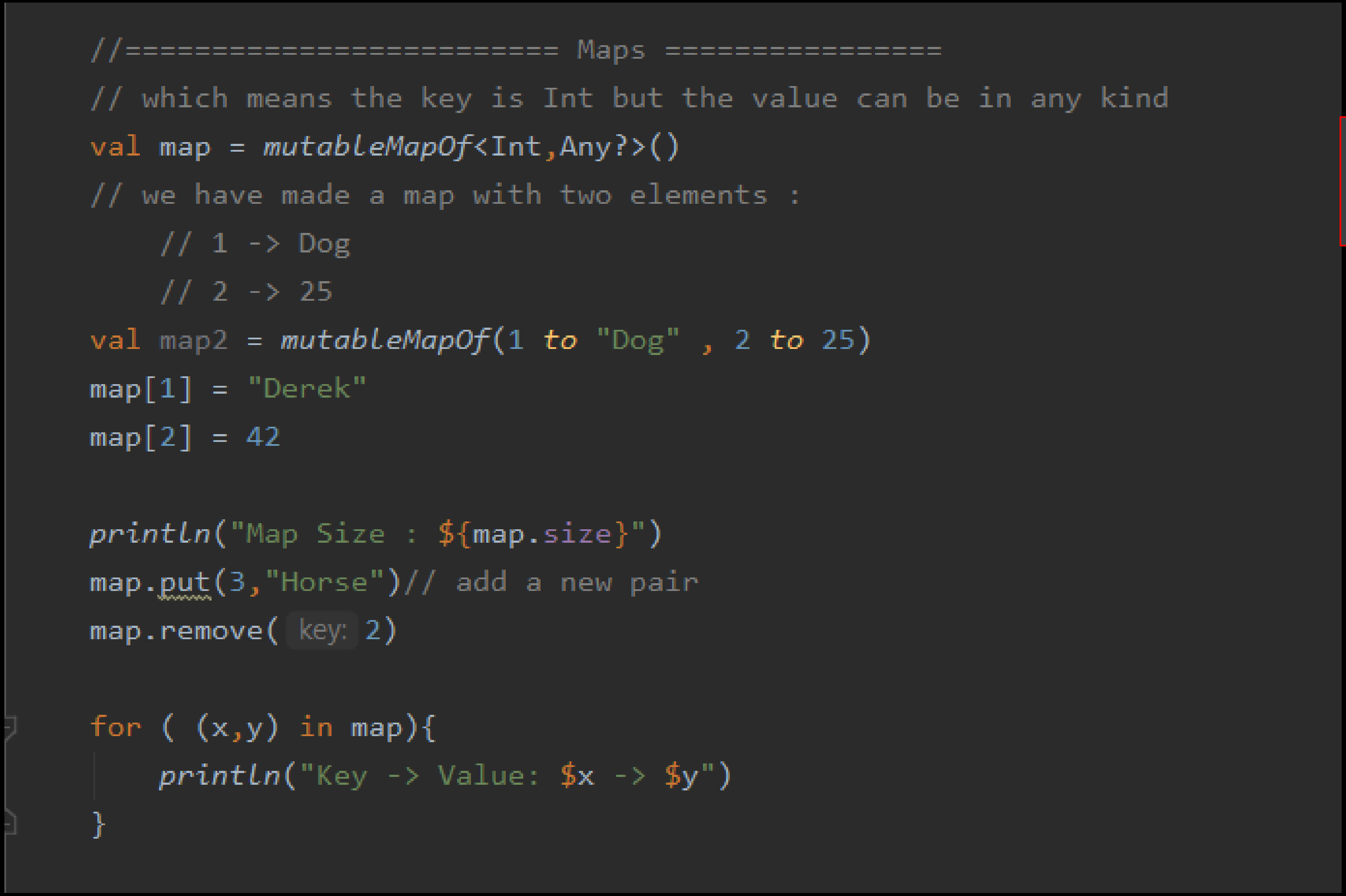


```
## Map ##
7
14
21
28
35
42
49
56
63
70
77
84
91
98
105
112
119
126
133
140
```

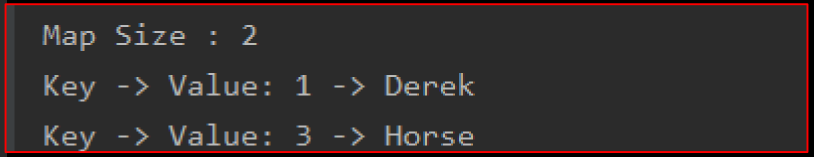


```
Evens : true
Evens : false
(4) is Bigger than 3
(5) is Bigger than 3
(6) is Bigger than 3
(7) is Bigger than 3
(8) is Bigger than 3
(9) is Bigger than 3
(10) is Bigger than 3
(11) is Bigger than 3
(12) is Bigger than 3
(13) is Bigger than 3
(14) is Bigger than 3
(15) is Bigger than 3
(16) is Bigger than 3
(17) is Bigger than 3
(18) is Bigger than 3
(19) is Bigger than 3
(20) is Bigger than 3
```

Maps (Data Structure)



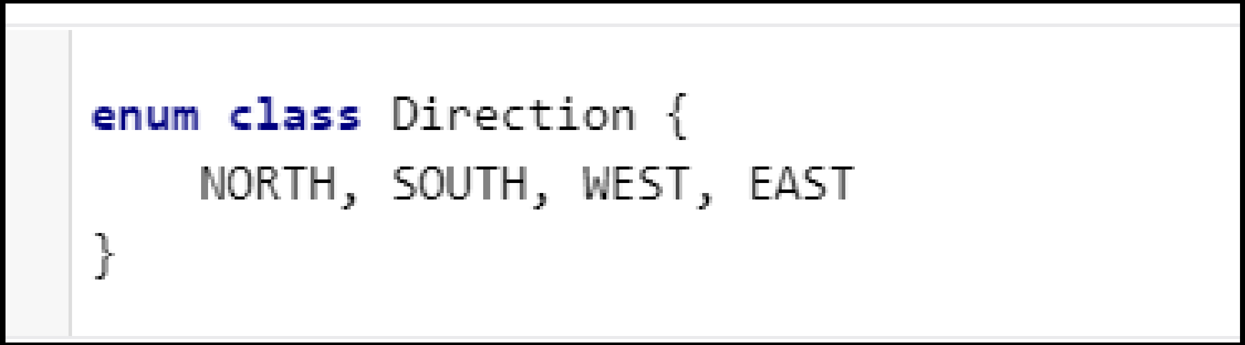
```
//======================== Maps ================
// which means the key is Int but the value can be in any kind
val map = mutableMapOf<Int,Any?>()
// we have made a map with two elements :
    // 1 -> Dog
    // 2 -> 25
val map2 = mutableMapOf(1 to "Dog" , 2 to 25)
map[1] = "Derek"
map[2] = 42

println("Map Size : ${map.size}")
map.put(3,"Horse")// add a new pair
map.remove( key: 2)

for ( (x,y) in map){
    println("Key -> Value: $x -> $y")
}
```



```
Map Size : 2
Key -> Value: 1 -> Derek
Key -> Value: 3 -> Horse
```

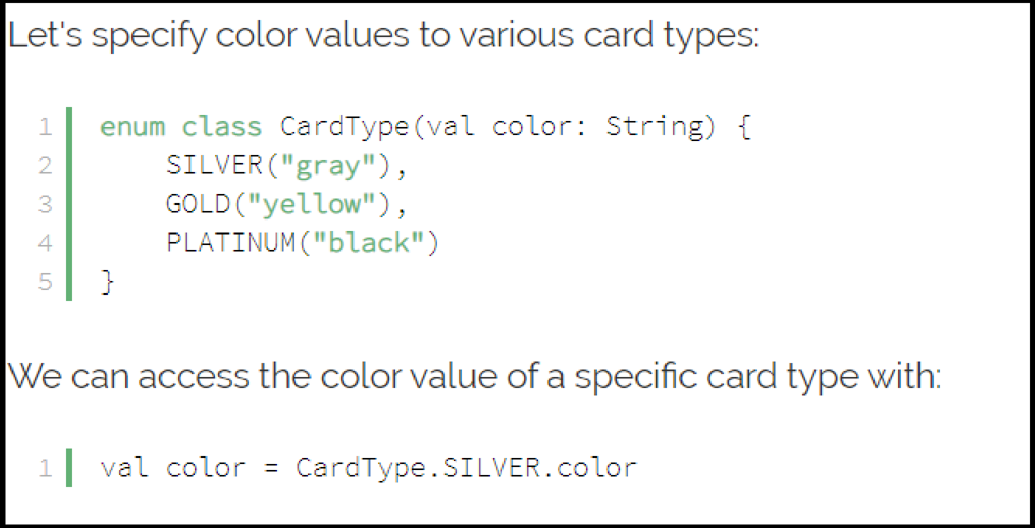
Associative Array (Dictionary) .(3)

Enumerations :



```kotlin
enum class Direction {
    NORTH, SOUTH, WEST, EAST
}
```

Reference : https://kotlinlang.org/



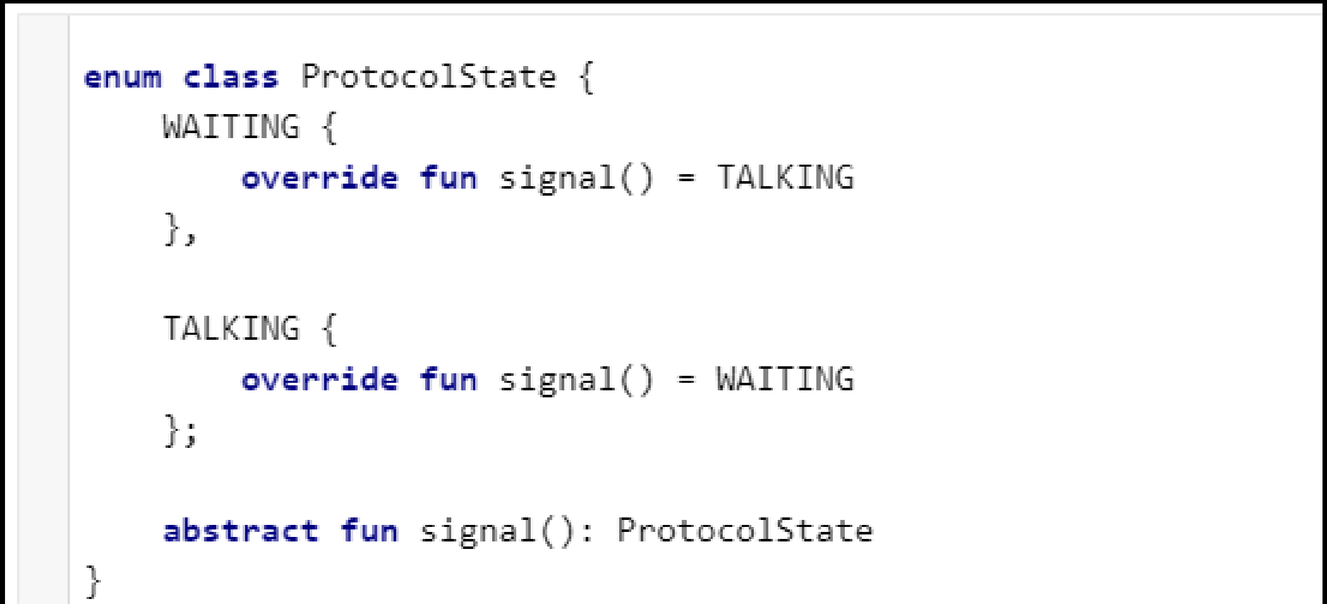
Let's specify color values to various card types:

```kotlin
enum class CardType(val color: String) {
    SILVER("gray"),
    GOLD("yellow"),
    PLATINUM("black")
}
```

We can access the color value of a specific card type with:

```kotlin
val color = CardType.SILVER.color
```

Reference : https://www.baeldung.com/kotlin-enum

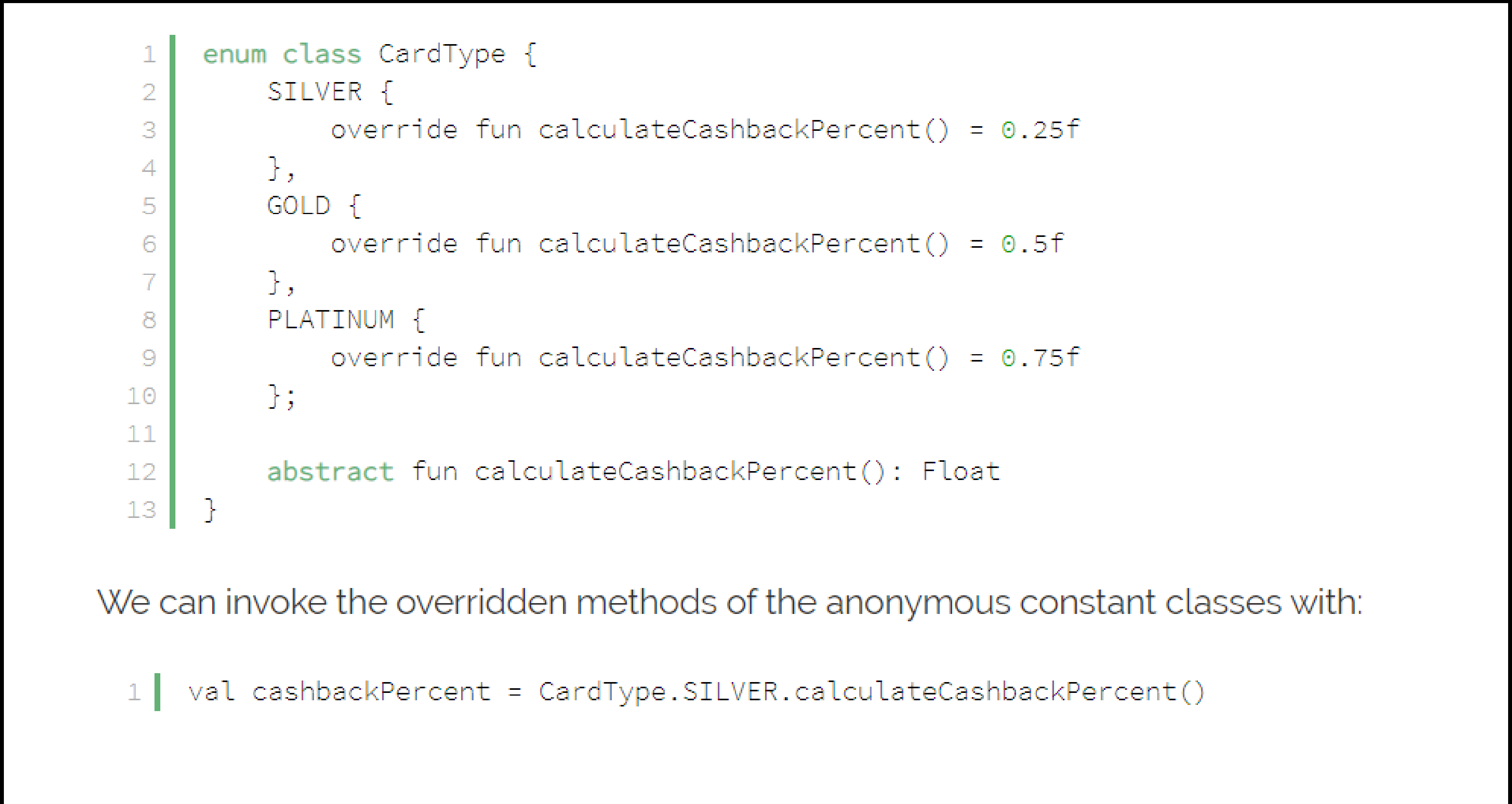


```kotlin
enum class ProtocolState {
    WAITING {
        override fun signal() = TALKING
    },

    TALKING {
        override fun signal() = WAITING
    };

    abstract fun signal(): ProtocolState
}
```

Reference : https://kotlinlang.org/

.Enums(2)

بررسی محدودیت در انجام عملیات بر روی آن

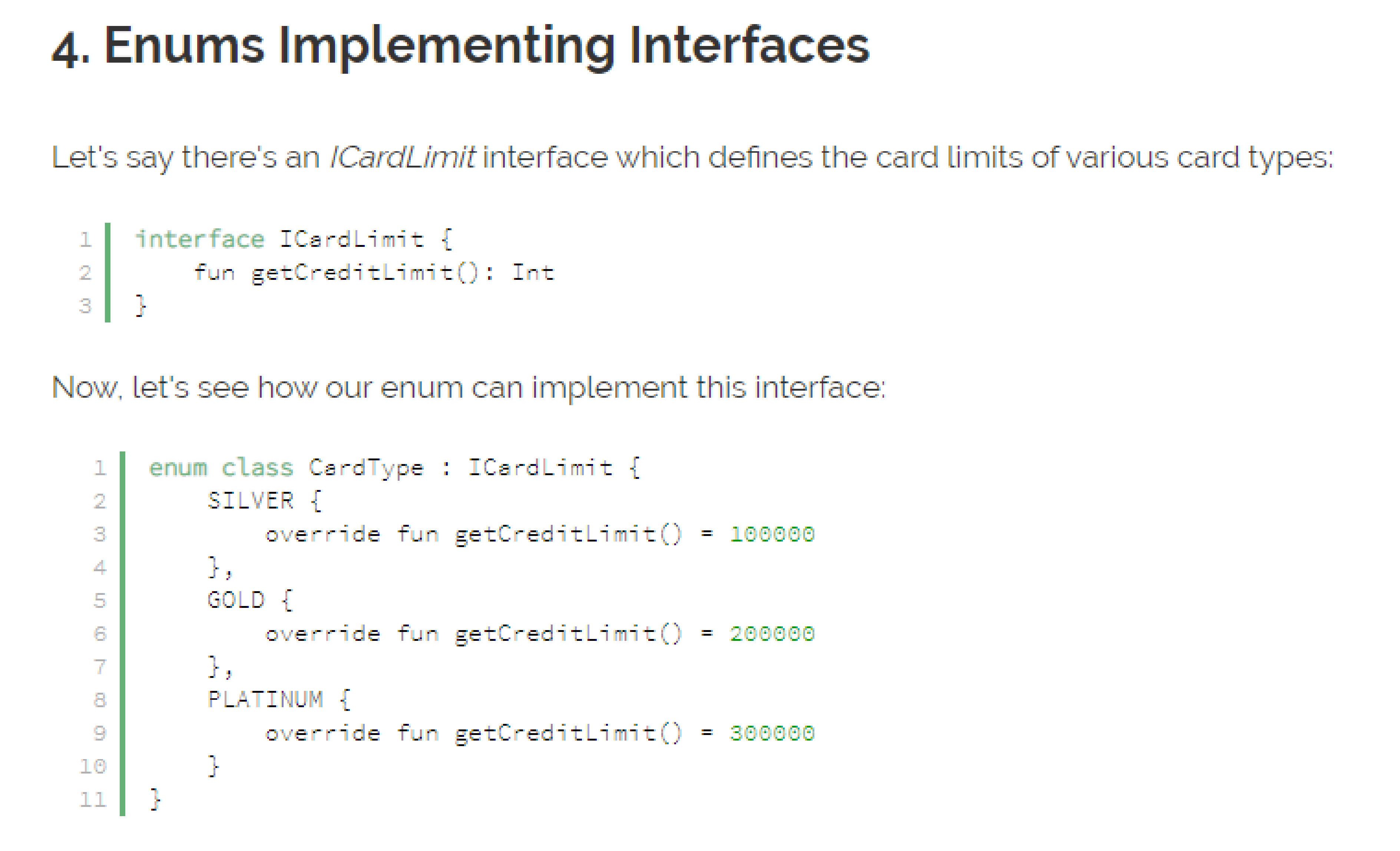


```kotlin
enum class CardType {
    SILVER {
        override fun calculateCashbackPercent() = 0.25f
    },
    GOLD {
        override fun calculateCashbackPercent() = 0.5f
    },
    PLATINUM {
        override fun calculateCashbackPercent() = 0.75f
    };

    abstract fun calculateCashbackPercent(): Float
}
```

We can invoke the overridden methods of the anonymous constant classes with:

```kotlin
val cashbackPercent = CardType.SILVER.calculateCashbackPercent()
```

Reference : https://www.baeldung.com/kotlin-enum



4. Enums Implementing Interfaces

Let's say there's an *ICardLimit* interface which defines the card limits of various card types:

```kotlin
interface ICardLimit {
    fun getCreditLimit(): Int
}
```

Now, let's see how our enum can implement this interface:

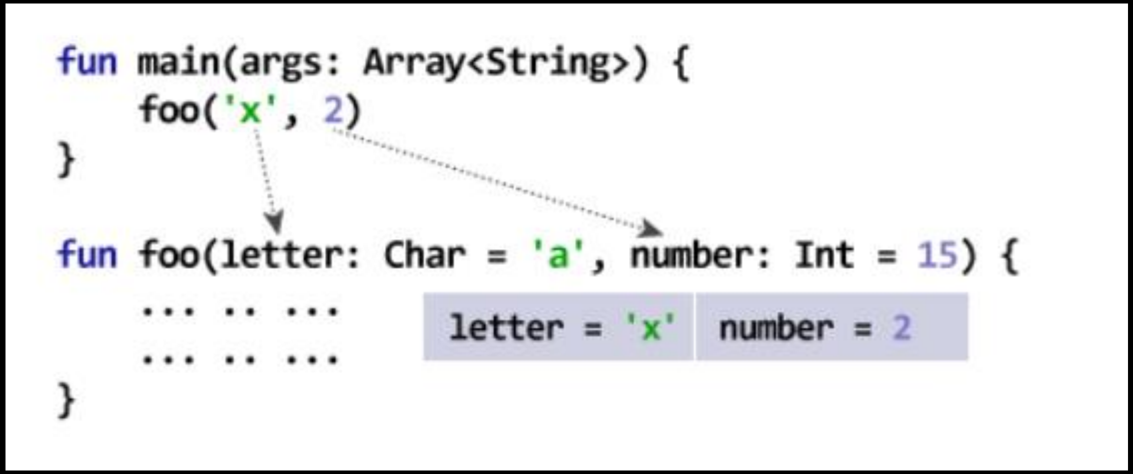
```kotlin
enum class CardType : ICardLimit {
    SILVER {
        override fun getCreditLimit() = 100000
    },
    GOLD {
        override fun getCreditLimit() = 200000
    },
    PLATINUM {
        override fun getCreditLimit() = 300000
    }
}
```

Reference : https://www.baeldung.com/kotlin-enum

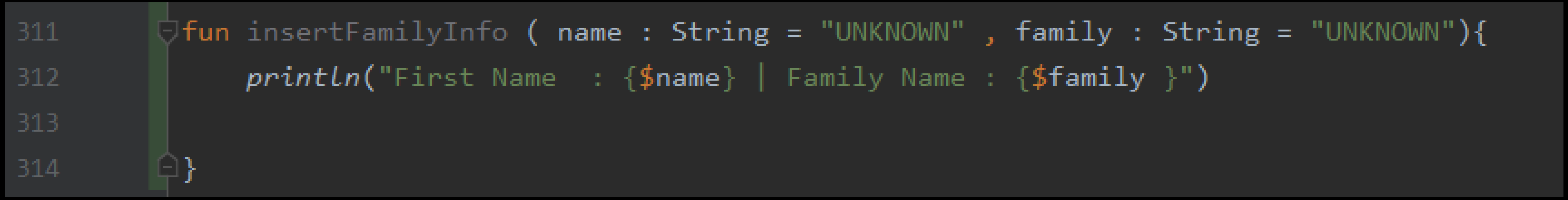
Default Values :

Reference : www.programiz.com

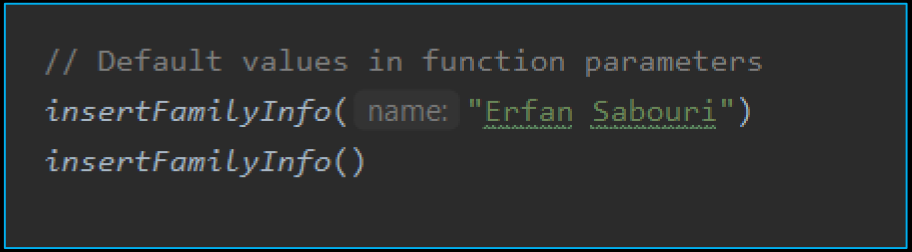
#Default Value(5)



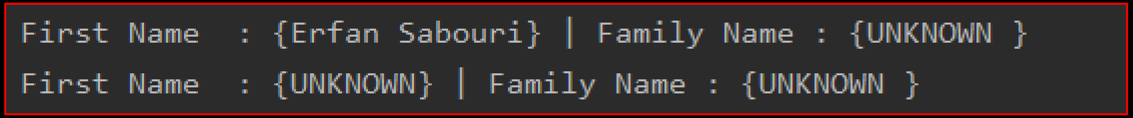
```
fun main(args: Array<String>) {
    foo('x', 2)
}

fun foo(letter: Char = 'a', number: Int = 15) {
    ... .. ...
    ... .. ...
}
```



```
fun insertFamilyInfo ( name : String = "UNKNOWN" , family : String = "UNKNOWN"){
    println("First Name  : {$name} | Family Name : {$family }")

}
```

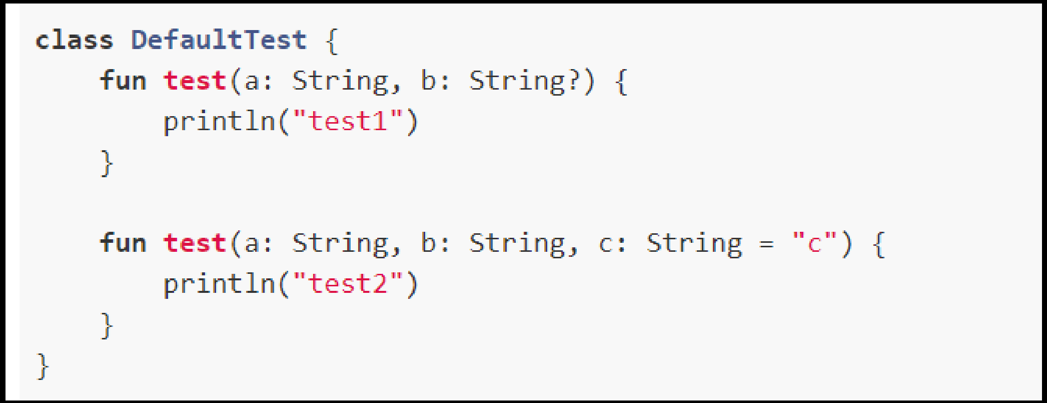


```
// Default values in function parameters
insertFamilyInfo( name: "Erfan Sabouri")
insertFamilyInfo()
```

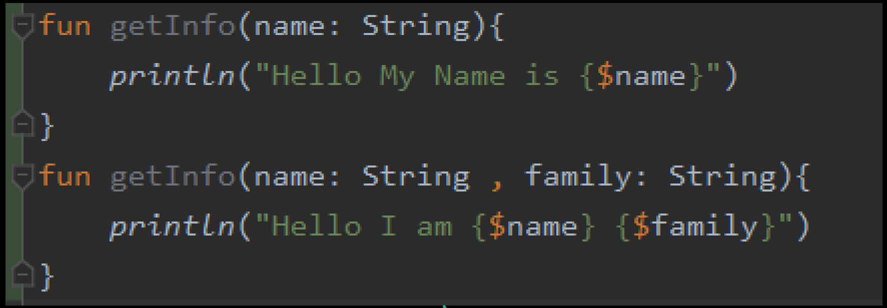


```
First Name  : {Erfan Sabouri} | Family Name : {UNKNOWN }
First Name  : {UNKNOWN} | Family Name : {UNKNOWN }
```

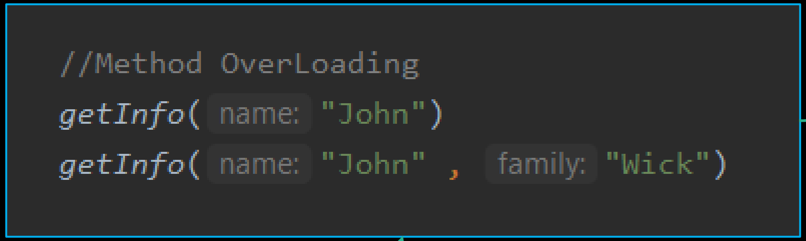
Method Over-Loading: #12



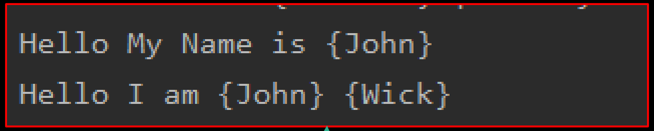
```
class DefaultTest {
    fun test(a: String, b: String?) {
        println("test1")
    }

    fun test(a: String, b: String, c: String = "c") {
        println("test2")
    }
}
```



```
fun getInfo(name: String){
    println("Hello My Name is {$name}")
}
fun getInfo(name: String , family: String){
    println("Hello I am {$name} {$family}")
}
```



```
//Method OverLoading
getInfo( name: "John")
getInfo( name: "John" ,  family: "Wick")
```



```
Hello My Name is {John}
Hello I am {John} {Wick}
```

Argument Passing :

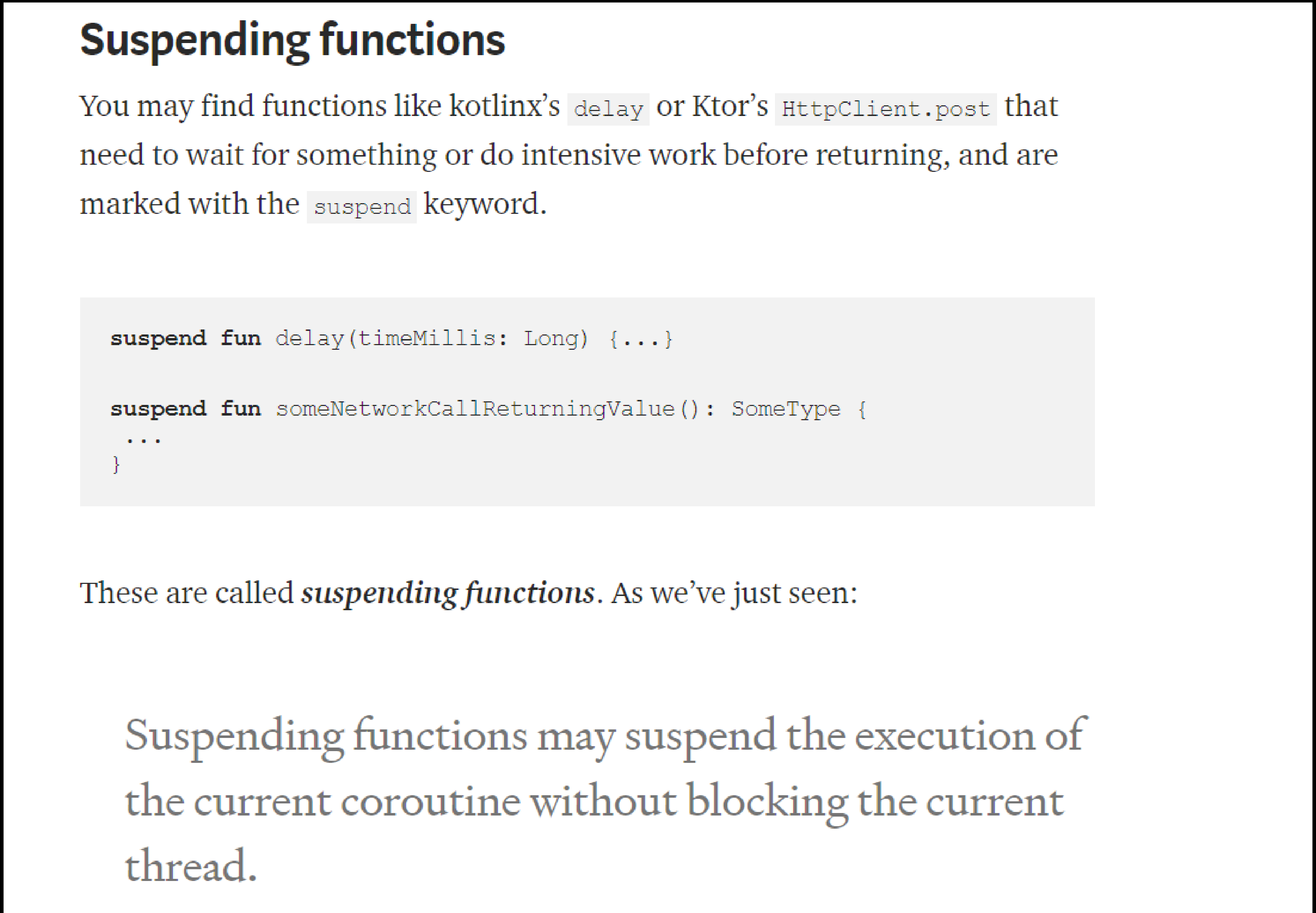
- Call – by – value -> Supports
- Call – by – value – result - > Doesn’t Support
- Call – by – result -> Doesn’t Support
- Call – by – reference -> Doesn’t Support
- Call – by – name -> Doesn’t Support

Static Or Dynamic Scoping ? #1

- Kotlin Only Supports static scoping.

Coroutine :

- The most interesting thing is that a thread can stop executing a coroutine at some specific “suspension points”, and go do some other work. It can resume executing the coroutine later on, or another thread could even take over.



Suspending functions

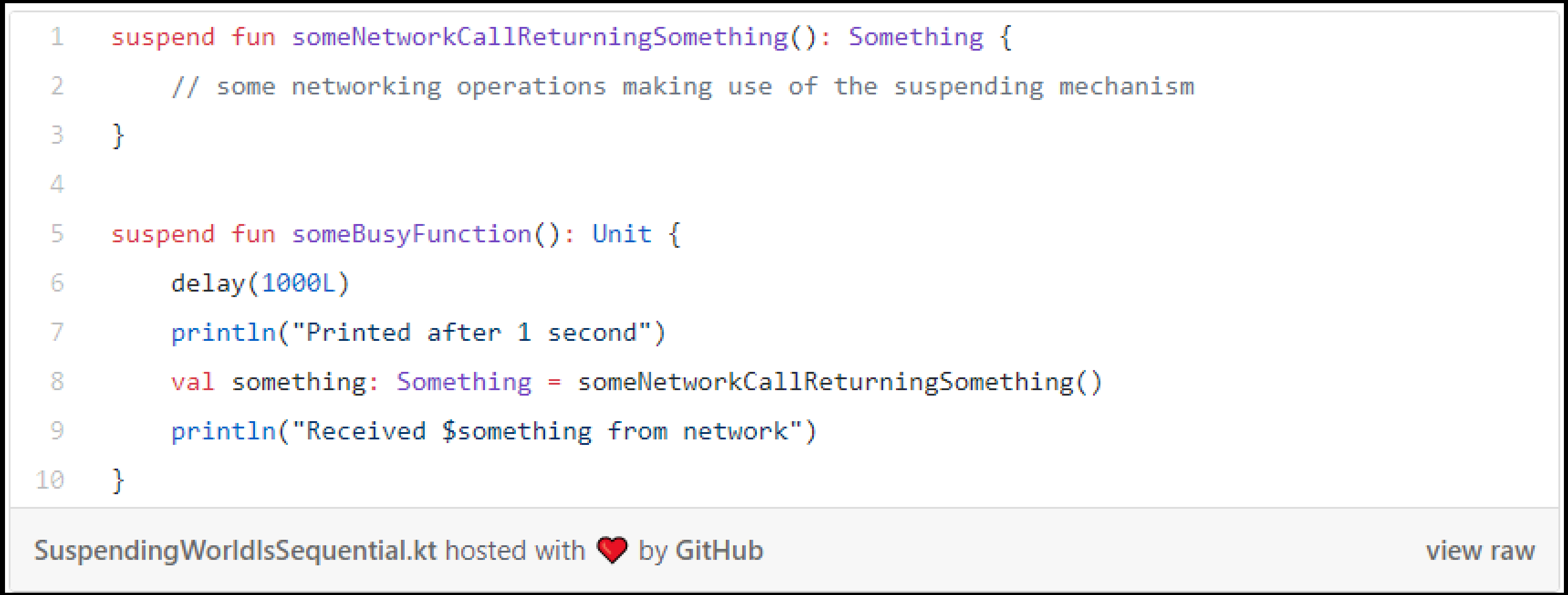
You may find functions like kotlinx's `delay` or Ktor's `HttpClient.post` that need to wait for something or do intensive work before returning, and are marked with the `suspend` keyword.

```
suspend fun delay(timeMillis: Long) {...}

suspend fun someNetworkCallReturningValue(): SomeType {
 ...
}
```

These are called ***suspending functions***. As we've just seen:

> Suspending functions may suspend the execution of the current coroutine without blocking the current thread.

[Reference : https://proandroiddev.com/how-to-make-sense-of-kotlin-coroutines-b666c7151b93](https://proandroiddev.com/how-to-make-sense-of-kotlin-coroutines-b666c7151b93)

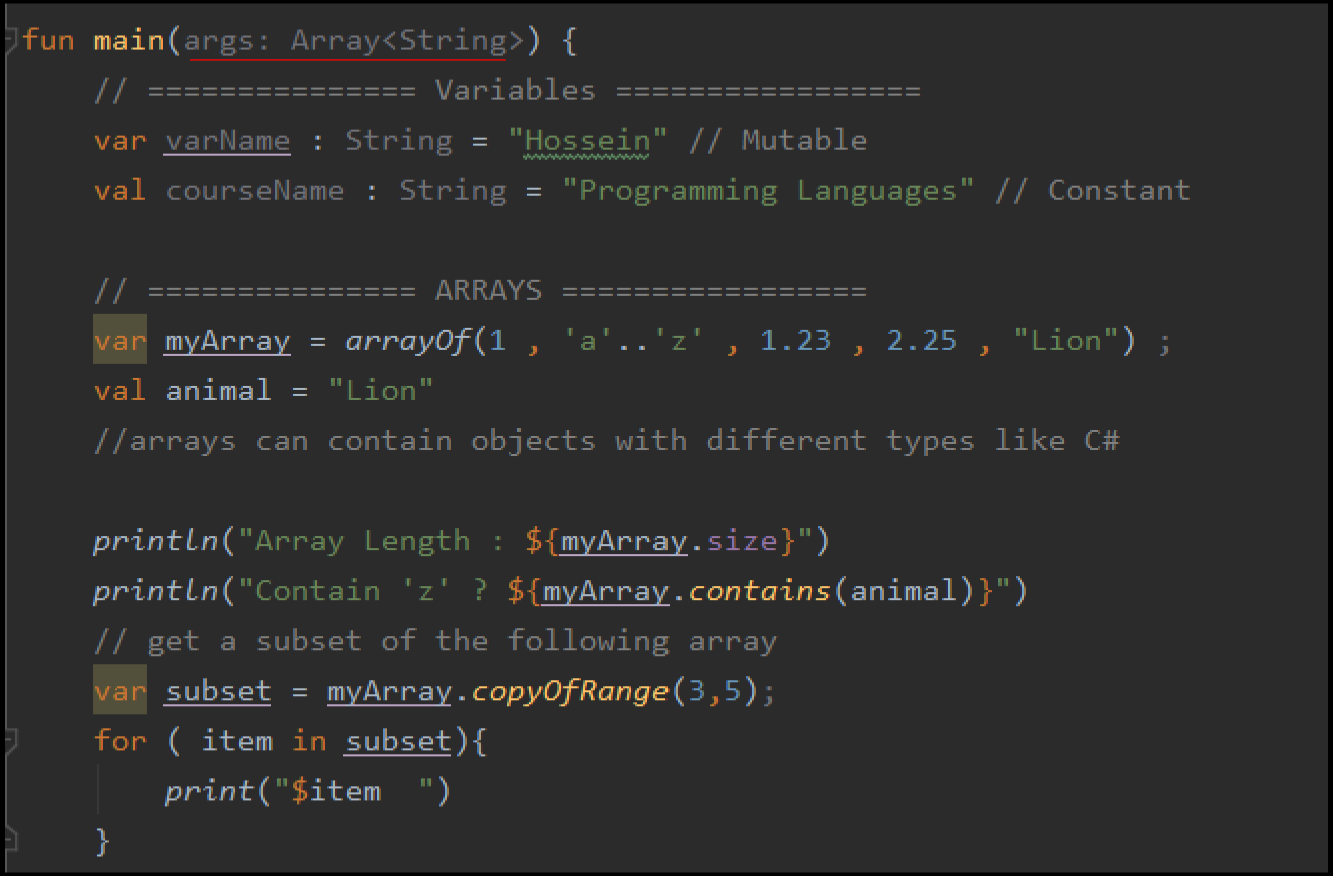


```kotlin
suspend fun someNetworkCallReturningSomething(): Something {
    // some networking operations making use of the suspending mechanism
}

suspend fun someBusyFunction(): Unit {
    delay(1000L)
    println("Printed after 1 second")
    val something: Something = someNetworkCallReturningSomething()
    println("Received $something from network")
}
```

SuspendingWorldIsSequential.kt hosted with ❤ by GitHub

view raw

Positional Parameters :



```kotlin
fun main(args: Array<String>) {
    // =============== Variables =================
    var varName : String = "Hossein" // Mutable
    val courseName : String = "Programming Languages" // Constant

    // =============== ARRAYS =================
    var myArray = arrayOf(1 , 'a'..'z' , 1.23 , 2.25 , "Lion") ;
    val animal = "Lion"
    //arrays can contain objects with different types like C#

    println("Array Length : ${myArray.size}")
    println("Contain 'z' ? ${myArray.contains(animal)}")
    // get a subset of the following array
    var subset = myArray.copyOfRange(3,5);
    for ( item in subset){
        print("$item  ")
    }
```

Unions -> Does not support Unions

References :

- YouTube/Derek Banas
- www.TutorialsPoints.com
- www.proandroiddev.com
- www.programiz.com
- www.kotlinlang.org
- www.baeldung.com

Thanks for watching !

- You can upload the source code and presentation at :
  - https://github.com/hosseindehghanipour1998

Any Questions or Suggestions :
Hossein.dehghanipour1998@gmail.com