CORSO KOTLIN

Sommario

INTRODUZIONE

Kotlin è un linguaggio di programmazione ufficialmente supportato per la programmazione android. Questo linguaggio funziona dovunque funzioni JAVA, dal momento che viene eseguito sulla JVM, che è la stessa macchina su cui viene eseguito anche il codice JAVA.

Kotlin è un linguaggio orientato agli oggetti, ma allo stesso tempo utilizza la programmazione funzionale. Nella programmazione funzionale si possono utilizzare funzioni come variabili, posso salvarle all'interno delle variabili, posso ritornarle da altre funzioni e posso passarle come parametri in altre funzioni.

In Kotlin si possono dichiarare degli elementi come immutabili, ovvero che non possono essere modificati.

Tutto ciò consente a Kotlin di essere un linguaggio molto coinciso e permette di scrivere molto meno codice, rispetto a quello che avremmo scritto in JAVA.

Infine Kotlin ha la capacità di poter essere eseguito e di essere richiamato da/su classi JAVA.

VARIABILI

Le variabili sono i contenitori in cui andiamo a mettere i diversi tipi di dati. La keyword var è quella che ci permette di creare una variabile. Tale keyword a sua volta chiede altre informazioni per creare la variabile, ossia il nome, il tipo e il valore. Vediamo qualche esempio:

```
var numeroDiPersone: Int = 2000
var numeroDiPersone2: Byte =20
var numeroDiPersone3: Short = 200
var numeroDiPersone4:  Long = 2383848484339L

var PI: Double = 3.121537327468
var PI2: Float = 3.23629329f

var nome: String = "Ciao"

var carattere: Char = 'a'

var booleano: Boolean = true
```

In Kotlin i tipi int, short, float, char e così via, non sono considerati tipi primitivi come in JAVA, ma sono considerati comunque come oggetti.

Detto ciò, non è obbligatorio inserire il tipo della variabile, ma c'è un modo per farlo intuire automaticamente da Kotlin. Basta togliere i due punti con il tipo a seguire e, comunque, il codice continuerà a funzionare. Questo perché Kotlin riesce a dedurre il tipo della variabile semplicemente dal valore che gli andiamo ad assegnare:

```
var numeroDiPersone = 2000
var numeroDiPersone2 =20
var numeroDiPersone3 = 200
var numeroDiPersone4 = 2383848484339L

var PI = 3.121537327468
var PI2 = 3.23629329f

var nome = "Ciao"

var carattere = 'a'
var booleano = true
```

RIASSEGNAZIONE VARIABILI

Partiamo con una semplice assegnazione di un valore ad una variabile:

```
var moneta = 1000
```

Vogliamo in seguito cambiare quel valore dentro la variabile money. Il modo giusto per farlo è semplicemente riscrivendo solo il nome della variabile, assegnandole il nuovo valore, senza utilizzare la keyword var, perché non abbiamo bisogno di crearne una nuova dato che ce l'abbiamo già:

```
moneta = 800
```

Possiamo però assegnare ad una nuova variabile un'altra variabile, come segue:

```
var accountBanca = moneta
```

Se stampiamo il valore di accountBanca, vediamo a schermo l'ultimo valore aggiunto a moneta:

```
print ("Totale soldi nel suo conto: " + accountBanca) //Totale soldi nel suo conto: 800
```

OPERAZIONI CON VARIABILI

Vediamo un esempio facendo il calcolo della media:

```
fun main() {
    //Operazione di media: (8 + 9 + 4 + 6) / 4
    var voto1 = 8
    var voto2 = 9
    var voto3 = 4
    var voto4 = 6
    var media = (voto1 + voto2 + voto3 + voto4) / 4f //la f serve per avere risultato float, altrimenti arrotonda all'intero
    println("La media dei voti è: " + media) // 6.75
}
```

Nel prossimo esempio, analogo a quello precedente, vediamo come convertire una stringa in un valore numerico:

```
fun main() {
    var voto1 = "8"
    var voto2 = 9
    var voto3 = 4
    var voto4 = 6
    var media = (voto1.toInt() + voto2 + voto3 + voto4) / 4f
    println("La media dei voti è: " + media)
}
```

OPERATORI

Vediamo un esempio:

```
fun main() {
    // operatori classici: + - / *

    // modulo: % ci restituisce il resto di una divisione
    println(2 % 2)// 0
    println(3 % 2)// 1

    var number = 5
    number += 10 // corrisponde a 10 + 5 e questo vale anche per - / *
    println(number) //15

    number++ // incrementa di 1 la variabile
    println(number) //16
    number-- // decrementa di 1 la variabile
    println(number) //15
}
```

C'è da approfondire una casistica che riguarda gli operatori di incremento e decremento. Infatti è possibile scrivere tali operatori anche prima della variabile e questo ne cambia anche il comportamento. Vediamo un esempio con il decremento per capire la differenza:

```
var number2 = 3
println(number2--)// 3
println(number2)//2
```

In questo esempio usiamo il decremento dopo la variabile, ma la stampa a video sulla stessa riga dell'operatore restituisce lo stesso valore con cui abbiamo inizializzato la variabile. Solo sulla seconda stampa a video avremo effettivamente il decremento del valore. Questo perché, nel caso in cui l'operatore è scritto dopo la variabile, il compilatore andrà prima a controllare il contenuto della variabile e, solo sulla riga successiva, andrà effettivamente a decrementarlo. Viceversa succede se andiamo a scrivere l'operatore prima della variabile:

```
var number2 = 3
println(--number2)// 2
```

dove il decremento avviene direttamente sulla stessa riga dove ho l'operatore. Stessa cosa vale per l'incremento.

DIFFERENZA VAL E VAR

Entrambe le keyword vengono utilizzate per dichiarare una variabile. La differenza sta nel fatto che, come abbiamo visto, quando abbiamo una variabile con la keyword var, possiamo inizializzare più volte il valore che contiene, sostituendo quello precedente. Viceversa, una variabile con la keyword val viene considerata "immutabile", cioè una volta inizializzata ad un valore, quel valore non può essere più cambiato. Vediamo un esempio:

```
val voto1 = 10
voto1 = 9 //Val cannot be reassigned

var voto2 = 10
voto2 = 9
```

TRACCIA 1 PER ESERCITAZIONE

Per completare questo esercizio dovrai:

- Creare un algoritmo in grado di calcolare la tua età .
- Per fare questo dovrai poter inserire solo l'anno in cui sei nato.
- Non potrai scrivere a mano l'anno corrente ma dovrai prenderlo tramite delle classi Kotlin o Java (cerca su Internet come fare)

```
import java.text.SimpleDateFormat
import java.time.Year
import java.util.*

fun main(args: Array<String>) {
    print("Il mio anno di nascita è ")
    var annoNascita:Int = readLine()!!.toInt()
    // readLine() è usato per accettare la stringa
    // e ".toInt()" la converte da stringa a intero.
    //val sdf = SimpleDateFormat("dd/M/yyyy hh:mm:ss")
    val sdf = SimpleDateFormat("yyyy")
    val dataCorrente = sdf.format(Date())
    //System.out.println(" La data di oggi è: "+ currentDate)
    var eta:Int= dataCorrente.toInt() - annoNascita
    print("Ho $eta anni")
}
```

STRINGHE

Come sappiamo, le stringhe sono variabili che contengono del testo. In JAVA, per stampare a video un valore numerico con del testo affianco, è necessario concatenare il valore numerico dopo o prima del testo, come nell'esempio:

```
fun main() {
    val moneta = 5.34
    println("Il totale delle monete in mio possesso è " + moneta)
}
```

In Kotlin è possibile farlo in una maniera meno macchinosa grazie all'utilizzo del simbolo $, come nel prossimo esempio:

```
fun main() {
    val moneta = 5.34
    println("Sono in possesso di $moneta monete")
}
```

In questo modo è possibile richiamare una variabile direttamente all'interno della stringa. Il tipo della variabile da inserire nella stringa può essere qualsiasi, anche un carattere o un'altra stringa.

Possiamo anche eseguire delle operazioni all'interno della stringa, come nel seguente esempio:

```
fun main() {
    val moneta = 5.34
    val tasse = 2.20
    println("Sono in possesso di ${moneta - tasse} monete")
}
```

Kotlin mette a nostra disposizione quelle che sono chiamate Raw Strings, che sono stringhe racchiuse tra 3 doppie virgolette e sono utilizzate per indicare il percorso di un file, come nell'esempio seguente:

```
val path = """C:\cartella1\cartella2\file"""
```

Possiamo anche utilizzare le Raw Strings anche per creare un output ordinato, facendo in modo che venga fuori un testo allineato senza spazi o tab indesiderati, come nel seguente esempio:

```
val biografia = """Mi chiamo Stefano
                  |ho 27 anni
                  |faccio l'informatico""".trimMargin()
println(biografia)
```

L'output di tale codice è il seguente:

Mi chiamo Stefano
ho 27 anni
faccio l'informatico

Se al posto del simbolo |, che è utilizzato di default da Kotlin per allineare il testo, volessimo utilizzare un altro simbolo a nostra scelta, dobbiamo fare come nell'esempio seguente:

```
val biografia = """Mi chiamo Stefano
                  -ho 27 anni
                  -faccio l'informatico""".trimMargin(marginPrefix = "-")
println(biografia)
```

L'output di questo codice è uguale a quello precedente.

ARRAY

Un'array è un insieme di variabili. Ogni variabile occupa una posizione all'interno dell'array, contrassegnata da un indice. Si deve però prestare attenzione al fatto che l'indice in prima posizione parte da 0 e non da 1. Per dichiarare un array abbiamo bisogno della keyword val (perchè l'array nasce come struttura immutabile), seguita da un nome e un metodo arrayOf(), che ha come parametri un insieme di valori che saranno inseriti all'interno dell'array. Vediamo qualche esempio:

```
fun main() {
    val arrayInteri = arrayOf(1, 2, 3, 4)
    val arrayStringhe = arrayOf("Marco", "Anna", "Matilde", "Gianfranco")
    val arrayMisto = arrayOf("marco", 2, 5, 4.0, 8.0f, 's', false)
    //ciclo for per leggere ogni elemento dell'array di interi
    for (numero in arrayInteri){
        println(numero)
    }
    println()
    //ciclo for per leggere ogni elemento dell'array di stringhe
    for (nome in arrayStringhe){
        println(nome)
    }
    println()
    //ciclo for per leggere ogni elemento dell'array misto
    for (elemento in arrayMisto){
        println(elemento)
    }
}
```

È possibile tuttavia dichiarare un array, scrivendo per ogni singolo indice che valore inserire. Vediamo un esempio:

```
val arrayInteri2: Array<Int?> = arrayOfNulls(3)
arrayInteri2[0] = 1
arrayInteri2[1] = 2
arrayInteri2[2] = 3
//lettura di un singolo elemento dell'array all'indice indicato
println(arrayInteri2[1])
//ciclo for per leggere ogni elemento dell'array misto
for (elemento2 in arrayInteri2){
    println(elemento2)
}
```

CONDIZIONALI

IF ELSE STATEMENT

Vediamo un esempio di classico costrutto if:

```
fun main() {
    val totaleMonete = 0
    //val totaleMonete = 50
    //val totaleMonete = 10
    if (totaleMonete > 0 && totaleMonete <= 5){
        print("posso spendere pochi soldi")
    }else if (totaleMonete > 5 && totaleMonete <= 20){
        print("posso comprarmi un buon pasto")
    }else if(totaleMonete > 20) {
        print("sono ricco")
    }else if (totaleMonete == 0){
        print("sono povero")
    }
}
```

É possibile in Kotlin creare degli if else statement con un range di valori all'interno della condizione (con estremi compresi nell'intervallo), come nel seguente esempio:

```
fun main() {
    val totaleMonete = 7
    if (totaleMonete in 1..5){
        print("posso spendere pochi soldi")
    }else if (totaleMonete in 6..20){
        print("posso comprarmi un buon pasto")
    }else if(totaleMonete > 20) {
        print("sono ricco")
    }else if (totaleMonete == 0){
            print("sono povero")
    }
}
```

Una funzione simile ai due puntini è until, solo che non comprende nell'intervallo il secondo estremo, come nell'esempio:

```
fun main() {
    val totaleMonete = 4
    if (totaleMonete in 1 until 5){ // totaleMonete >=1 && totaleMonete < 5
        print("posso spendere pochi soldi")
    }else if (totaleMonete in 6..20){
        print("posso comprarmi un buon pasto")
    }else if(totaleMonete > 20) {
        print("sono ricco")
    }else if (totaleMonete == 0){
            print("sono povero")
    }
}
```

WHEN STATEMENT

Il costrutto when viene utilizzato come se fosse uno switch in altri linguaggi. Quello che si fa con when, si può fare anche con un if else, solo che il costrutto when è più pulito dal punto di vista della leggibilità del codice. Vediamo l'esempio precedente con questo costrutto:

```
fun main() {
    val totaleMonete = 4
    when(totaleMonete){
        in 0..5 -> println("posso spendere pochi soldi")
        in 6..20 -> println("posso comprarmi un buon pasto")
        else -> {println("sono ricco")}
    }
}
```

CICLI

Un ciclo è un blocco di codice che viene eseguito più volte, finché una determinata condizione non sarà verificata. I cicli possono essere definiti in vari modi e possono essere di vari tipi.

FOR

Il ciclo for è quello più utilizzato in assoluto. La variabile inserita all'interno della condizione non deve essere dichiarata, ma sarà il compilatore che inserirà direttamente un var dietro le quinte. Vediamo alcuni esempi:

```
fun main() {
    for(x in 1..10){
        print("$x ") //1 2 3 4 5 6 7 8 9 10
    }
}
```

```
fun main() {
    for(x in 1 until 10){
        print("$x ") //1 2 3 4 5 6 7 8 9
    }
}
```

```
fun main() {
    for(x in 15 downTo 1) {
        print("$x ") //15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
    }
}
```

```
for(x in 1..10 step 2){
   print("$x ") //1 3 5 7 9
}
```

Il for è utilizzato anche per poter accedere ad ogni elemento di una lista e di una maplist, ed eseguirvi del codice su di ognuno di esso. Vediamo degli esempi su queste strutture:

```
fun main() {
    val listaNomi= listOf<String>("Marco", "Anna", "Michele", "Alfredo")
    for(nome in listaNomi){
        print("$nome ")//Marco Anna Michele Alfredo
    }
    print("\n")
    val mapListNomi= mapOf(30 to "Marco", 20 to "Anna", 40 to "Michele", 50 to
"Michele")
    for((key, value) in mapListNomi){
        print("$key $value, ")//30 Marco, 20 Anna, 40 Michele, 50 Michele,


    }

}
```

WHILE

É importante sapere che tutto ciò che si può fare con il ciclo for, si può fare anche con il ciclo while. Il ciclo for in alcuni casi può essere più comodo da utilizzare rispetto il ciclo while, però ci sono altri casi in cui il while diventa più importante del for.

Nel caso del ciclo while, la variabile da utilizzare all'interno della condizione ha bisogno di essere dichiarata e inizializzata. Il codice al suo interno verrà eseguito finché la condizione al suo interno risulta essere vera, altrimenti esce fuori. Vediamo un esempio:

```
fun main() {
    var x = 1
    while (x < 11){
        print("$x ")//1 2 3 4 5 6 7 8 9 10
        x++
    }
}
```

Vediamo un altro esempio in cui creiamo un loop infinito, in quanto la condizione sarà sempre vera e il codice verrà sempre rieseguito, quindi non riuscirà mai ad uscire dal ciclo:

```
fun main() {
    while(true){
        println ("sto eseguendo")
    }
}
```

DO WHILE

Il do while è molto simile al while, ma il do while esegue il codice almeno una volta. Mentre il while, prima di eseguire il blocco di codice, va a verificare la condizione almeno una volta, il do while esegue prima il blocco di codice e poi va a verificare se la condizione è vera, oppure no. Anche qui, il codice verrà rieseguito finché la condizione risulterà essere vera. L'utilizzo del do while in programmazione è molto raro, ma si utilizza soprattutto nei casi in cui si deve eseguire un blocco di codice e poi dobbiamo andare a controllarlo. Vediamo un esempio che ricrea la tabellina del 2:

```
fun main() {
    var num = 2
    var i = 1
    do{
        println("2 * $i = ${num * i}")
        i++
    }while (i < 11)
}
```

TRACCIA 3 PER ESERCITAZIONE

La traccia 2 verrà svolta quando si studieranno le MapList.

Per completare questo esercizio dovrai:

- utilizzare un ciclo per poter prendere i nomi dalla lista `names`
- dovrai successivamente sostituire lo spazio tra i nomi con un "_" e sistemare le lettere in modo che siano tutte minuscole. Il risultato dovrebbe essere con quello mostrato all'interno della lista `usernames`

val names = listOf("Marco Rossi", "Alfredo Andrei", "John Mayer", "Justin Biber")

//The output should be like this:
usernames = [marco_rossi, alfredo_andrei, john_mayer, justin_biber]

Per completare questo task dovrai cercare su internet come sostituire i caratteri all'interno delle stringhe con altri e come rendere tutte le lettere minuscole.

```
fun main() {
    val names = listOf("Marco Rossi", "Alfredo Andrei", "John Mayer", "Justin
Biber", "Maria Verdi")
    val namesArray: Array<String?> = arrayOfNulls(names.size)
    var x = 0
    var y = 0
    print("usernames = [")
    for(nome in names){
        val nomeMinuscolo = nome.lowercase().replace(" ","_")
        namesArray[x] = nomeMinuscolo
        x++
    }
    for (nomeMinuscolo in namesArray){
        if(y < namesArray.size-1) {
            print("$nomeMinuscolo, ")
        } else {
            print("$nomeMinuscolo")
        }
        y++
    }
    print("]")
}
```

BREAK E CONTINUE

Qualche volta abbiamo bisogno di bloccare un ciclo, ed è qui che entrano in gioco le keyword break e continue. Entrambe possono essere usate per cicli for e while.

Break serve per terminare un ciclo, mentre continue serve solo per saltare un'iterazione del ciclo. Vediamo un esempio in cui abbiamo un programma in cui andremo a vedere se il peso totale di un quantitativo di prodotti supera il peso massimo che può portare un camion di rifornimenti:

```
fun main() {

    val mapFood = mapOf(
        "banane" to 15,
        "materassi" to 24,
        "mangime per cani" to 42,
        "attrezzi da lavoro" to 120,
        "formaggi" to 5)

    var pesoCamion = 0
    val articoli = mutableListOf<String>()

    for ((tipoArticolo, pesoArticolo) in mapFood){
        println("Il peso totale del camion è $pesoCamion")
        if (pesoCamion >= 100){
            println("ciclo stoppato")
            break
        }
        else if (pesoCamion + pesoArticolo > 100) {
            println("ciclo saltato")
            continue /*salta iterazione,
            quindi non aggiunge l'ultimo articolo sul
            camion se il peso che sta portando il
            camion in quel momento + il peso del prossimo
            articolo supera il peso massimo (100)
            che può portare il camion */
        } else {
            println("aggiungi $pesoArticolo di $tipoArticolo")
            articoli.add(tipoArticolo)
            pesoCamion += pesoArticolo
        }
    }
}
```