Bucles y matrices en Kotlin

Los bucles pueden ejecutar un bloque de código siempre que se alcance una condición específica.

Los bucles son útiles porque ahorran tiempo, reducen errores y hacen que el código sea más legible.

Kotlin bucle While

El bucle while recorre un bloque de código siempre que una condición especificada sea true:

Sintaxis

```
while (condition) {
  // code block to be executed
}
```

En el siguiente ejemplo, el código del bucle se ejecutará una y otra vez, siempre que la variable del contador (i) sea menor que 5:

Ejemplo

```
var i = 0
while (i < 5) {
  println(i)
  i++
}
```

Nota: No olvide aumentar la variable utilizada en la condición; de lo contrario, el ciclo nunca terminará.

El bucle Do...While

El bucle do..while es una variante del bucle while. Este bucle ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, luego repetirá el bucle mientras la condición sea verdadera.

Sintaxis

```
do {
  // code block to be executed
} while (condition);
```

El siguiente ejemplo utiliza un bucle do/while. El bucle siempre se ejecutará al menos una vez, incluso si la condición es falsa, porque el bloque de código se ejecuta antes de que se pruebe la condición:

Ejemplo

```
var i = 0
do {
  println(i)
  i++
```

```
} while (i < 5)
```

¡No olvides aumentar la variable utilizada en la condición, de lo contrario el ciclo nunca terminará!

Matrices Kotlin

Las matrices se utilizan para almacenar múltiples valores en una sola variable, en lugar de crear variables separadas para cada valor.

Para crear una matriz, use la función arrayOf() y coloque los valores en una lista separada por comas dentro de ella:

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
```

Acceder a los elementos de una matriz

Puede acceder a un elemento de matriz consultando el **número de índice**, entre corchetes.

En este ejemplo accedemos al valor del primer elemento en cars:

Ejemplo

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
println(cars[0])
// Outputs Volvo
```

Nota: Al igual que con las cadenas, los índices de matriz comienzan con 0: [0] es el primer elemento. [1] es el segundo elemento, etc.

Cambiar un elemento de matriz

Para cambiar el valor de un elemento específico, consulte el número de índice:

Ejemplo

```
cars[0] = "Opel"
```

Ejemplo

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
cars[0] = "Opel"
println(cars[0])
// Now outputs Opel instead of Volvo
```

Longitud/tamaño de la matriz

Para saber cuántos elementos tiene una matriz, use la propiedad **size**:

Ejemplo

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
println(cars.size)
// Outputs 4
```

Comprobar si existe un elemento

Puede utilizar el operador **in** para comprobar si un elemento existe en una matriz:

Ejemplo

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
if ("Volvo" in cars) {
  println("It exists!")
} else {
  println("It does not exist.")
}
```

Recorrer una matriz

A menudo, cuando trabaja con matrices, necesita recorrer todos los elementos.

Puede recorrer los elementos de la matriz con el bucle for, sobre el cual aprenderá aún más en el próximamente.

El siguiente ejemplo genera todos los elementos de la matriz cars:

Ejemplo

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
for (x in cars) {
  println(x)
}
```

Kotlin bucle for

A menudo, cuando trabaja con matrices, necesita recorrer todos los elementos.

Para recorrer los elementos de la matriz, utilice el bucle for junto con el operador in:

Ejemplo
Genere todos los elementos en la matriz de autos:

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
for (x in cars) {
  println(x)
}
```

Puede recorrer todo tipo de matrices. En el ejemplo anterior, utilizamos una matriz de cadenas.

Bucles y matrices en Kotlin

En el siguiente ejemplo, recorremos una matriz de números enteros:

Ejemplo
```
val nums = arrayOf(1, 5, 10, 15, 20)
for (x in nums) {
  println(x)
}
```

Tradicional bucle for

A diferencia de Java y otros lenguajes de programación, en Kotlin no existe un bucle for tradicional.

En Kotlin, el bucle for se usa para recorrer matrices, rangos y otras cosas que contienen una cantidad contable de valores.

Kotlin Ranges

Con el bucle for también puedes crear rangos de valores con "..":

Ejemplo
Imprime todo el alfabeto:

```
for (chars in 'a'..'x') {
  println(chars)
}
```

También puedes crear rangos de números:

Ejemplo
```
for (nums in 5..15) {
  println(nums)
}
```

Nota: El primer y último valor están incluidos en el rango.

Comprobar si existe un valor

También puede utilizar el operador **in** para comprobar si existe un valor en un rango:

Ejemplo
```
val nums = arrayOf(2, 4, 6, 8)
if (2 in nums) {
  println("It exists!")
} else {
  println("It does not exist.")
}
```

Ejemplo
```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
if ("Volvo" in cars) {
```

```
  println("It exists!")
} else {
  println("It does not exist.")
}
```

Variantes del for

Si necesitamos que la variable del for no reciba todos los valores comprendidos en el rango sino que avance de 2 en 2 podemos utilizar la siguiente sintaxis:

```
for(i in 1..10 step 2) println(i)
```

Se imprimen los valores 1, 3, 5, 7, 9

Si necesitamos que la variable tome el valor 10, luego el 9 y así sucesivamente, es decir en forma inversa debemos utilizar la siguiente sintaxis:

```
for(i in 10 downTo 1) println(i)
```

Se imprimen los valores 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

También podemos utilizar el step con el downTo:

```
for(i in 10 downTo 1 step 2) println(i)
```

Se imprimen los valores 10, 8, 6, 4, 2

Podemos utilizar la siguiente variante de for que tendrá sentido cuando recorramos vectores:

```
for(x in 0 until 10) println(x)
```

Se imprimen los valores de 0 a 9 (No incluye el valor 10)

Ejercicios

E801: Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:
a) De cada triángulo la medida de su base, su altura y su superficie (la superficie se calcula multiplicando la base por la altura y dividiendo por dos).
b) La cantidad de triángulos cuya superficie es mayor a 12.

```
fun main() {
    var cantidad = 0
    print("Cuantos triángulos procesará:")
    val n = readln().toInt()
    for(i in 1..n) {
        print("Ingrese el valor de la base:")
        val base = readln().toInt()
        print("Ingrese el valor de la altura:")
        val altura = readln().toInt()
        val superficie = base * altura / 2
        println("La superficie es de $superficie")
        if (superficie > 12)
            cantidad++
    }
    print("La cantidad de triángulos con superficie superior a 12 son: $cantidad")
}
```

E802: Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.

```
fun main() {
    var suma = 0
    for(i in 1..10) {
        print("Ingrese un valor entero:")
        val valor = readln().toInt()
        if (i > 5)
            suma += valor
    }
    print("La suma de los últimos 5 valores es: $suma");
}
```

E803: Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50)

E804: Confeccionar un programa que permita ingresar un valor del 1 al 10 y nos muestre la tabla de multiplicar del mismo (los primeros 12 términos)
Ejemplo: Si ingresó 3 deberá aparecer en pantalla los valores 3, 6, 9, hasta el 36.

E805: Realizar un programa que lea los lados de n triángulos, e informar:

a) De cada uno de ellos, qué tipo de triángulo es: equilátero (tres lados iguales), isósceles (dos lados iguales), o escaleno (ningún lado igual)
b) Cantidad de triángulos de cada tipo.

E806: Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano.
Informar cuántos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que se ingrese la cantidad de puntos a procesar.

E807: Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:
a) La cantidad de valores ingresados negativos.
b) La cantidad de valores ingresados positivos.
c) La cantidad de múltiplos de 15.
d) El valor acumulado de los números ingresados que son pares.

E808: Realiza un programa que convierta un número decimal en su representaci6n binaria. Hay que tener en cuenta que desconocemos cuantas cifras tiene el número que introduce el usuario.
Por simplicidad, iremos mostrando el número binario con un digito por línea.

E809: Escribe un programa que incremente la hora de un reloj. Se pedirán por teclado Ia hora, minutos y segundos, así como cuantos segundos se desea incrementar Ia hora introducida. La aplicación mostrará la nueva hora. Por ejemplo, si las 13:59:51 se incrementan en 10 segundos, resultan las 14:0:1.

E810: Realiza un programa que nos pida un número n, y nos diga cuantos números hay entre 1 y n que sean primos. Un número primo es aquel que solo es divisible por 1 y por el mismo. Veamos un ejemplo para n = 8:

```
1 ->  primo
2 ->  primo
3 ->  primo
4 ->  no primo
5 ->  primo
6 ->  no primo
7 ->  primo
8 ->  no primo
```

E811: Diseña una aplicación que dibuje el triángulo de Pascal, para n filas. Numerando las filas y elementos desde 0, la fórmula para obtener el m-ésimo elemento de la n-ésima file es:

$$E(n, m) = \frac{n!}{m!(n - m)!}$$

Donde n! es el factorial de n.

Un ejemplo de triángulo de Pascal con 5 filas (n = 4) es:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

E812: Solicita al usuario un número n y dibuja un triángulo de base y altura n, de la forma (para n igual a 4):