Control de flujo en Kotlin: Condicionales (if, when) y bucles (for, while)

1. Introducción

Hasta ahora, todos los programas que hemos escrito se ejecutaban en línea recta: el ordenador leía la línea 1, luego la 2, luego la 3... y terminaba. Pero en el mundo real, las aplicaciones necesitan **tomar decisiones** (si el usuario tiene saldo, haz la compra; si no, muestra un error) y **repetir tareas** (mostrar los 50 mensajes de un chat uno por uno).

Para esto sirve el **Control de Flujo**. En Kotlin, contamos con herramientas modernizadas y súper potentes para dirigir el tráfico de nuestro código. ¡Vamos a

descubrirlas!

2. Tomando decisiones: el clásico `if / else`

La forma más básica de tomar una decisión en programación es usar la estructura `if` (si ocurre esto...) y `else` (si no, haz esto otro...).

La condición a evaluar siempre va entre paréntesis `()`, y el bloque de código que se ejecutará va entre llaves `{}`.

```
fun main() {
    val edad = 18

    if (edad >= 18) {
        println("Puedes entrar a la discoteca.")
    } else {
        println("Lo siento, vuelve a casa.")
    }
}
```

2.1. El súper-poder de Kotlin: el `if` como expresión

Si vienes de lenguajes como Java o JavaScript, conocerás el famoso *operador ternario* (`condicion ? valor1 : valor2`) para asignar variables en una sola línea. **En Kotlin, el operador ternario no existe porque no hace falta.**

En Kotlin, un `if` puede devolver un valor directamente. Si tu `if` y tu `else` solo tienen una línea, puedes quitar las llaves y hacerlo así de elegante:

```
fun main() {
    val a = 10
    val b = 20

    // El resultado del if se guarda directamente en la variable 'mayor'
    val mayor = if (a > b) a else b

    println("El número mayor es $mayor") // Imprime: 20
}
```

3. El condicional `when` : el `switch` con esteroides

Cuando tienes que evaluar muchísimas opciones distintas, usar decenas de `if / else if / else` encadenados hace que el código sea ilegible.

Otros lenguajes usan la palabra `switch` . Kotlin usa `when` (cuando), y es una de las herramientas más queridas por los desarrolladores.

3.1. `when` como instrucción

Colocamos la variable que queremos evaluar entre paréntesis. Luego usamos una «flechita» `->` para indicar qué hacer en cada caso. El `else` actúa como la opción por defecto si no se cumple ninguna de las anteriores.

```
fun main() {
    val boton = "X"

    when (boton) {
        "A" -> println("Saltar")
        "B" -> println("Atacar")
        "X" -> println("Abrir Inventario")
        "Y" -> println("Magia")
        else -> println("Botón no reconocido")
    }
}
```

Nota: Kotlin evalúa de arriba a abajo. En cuanto encuentra una coincidencia, ejecuta esa línea y ***sale*** *del `when` automáticamente. (¡Adiós a la pesadilla de olvidar poner los `break` de otros lenguajes!)*

3.2. `when` como expresión (devolviendo un valor)

Al igual que el `if` , podemos usar `when` para asignar un valor directamente a una variable:

```
val estadoSemaforo = "Rojo"

val accion = when (estadoSemaforo) {
    "Verde" -> "Acelerar"
```

```
        "Ambar" -> "Frenar poco a poco"
        "Rojo" -> "Detenerse"
        else -> "Llamar al mecánico" // Al devolver valor, el 'else' es
    OBLIGATORIO
    }
    println("Debes: $accion")
```

4. Rangos: preparando el terreno para los bucles

Antes de aprender a repetir tareas, necesitamos saber cómo crear **Rangos** (intervalos de valores) en Kotlin. Es facilísimo:

- `..` **(Punto punto):** Crea un rango que **incluye** el último número. `1..4` equivale a `1, 2, 3, 4`.
- `..<` **(Punto punto menor):** Crea un rango que **excluye** el último número. `1..<4` equivale a `1, 2, 3`.
- `downTo` : Cuenta hacia atrás. `4 downTo 1` equivale a `4, 3, 2, 1`.
- `step` : Cambia el tamaño del salto. `1..5 step 2` equivale a `1, 3, 5`.

(¡También funciona con letras del abecedario! Ej: `'a'..'d'`)

5. Bucles: repitiendo tareas sin cansarse

5.1. El bucle `for` (para cada...)

Se usa cuando sabes exactamente cuántas veces quieres repetir algo, o cuando quieres recorrer una Colección (como las Listas que vimos en el artículo anterior).

```
fun main() {
    // Repetir un código un número exacto de veces usando un rango
    for (numero in 1..5) {
        print(numero) // Imprime: 12345
    }

    println() // Salto de línea

    // Recorrer una lista
    val pasteles = listOf("Zanahoria", "Queso", "Chocolate")
```

```
        for (pastel in pasteles) {
            println("¡Qué rico, un pastel de $pastel!")
        }
    }
```

5.2. Los bucles `while` y `do-while` (mientras que...)

Se usan cuando **no sabes** cuántas veces se va a repetir algo, pero sabes que debe repetirse «mientras» se cumpla una condición.

- `while` : Primero comprueba la condición. Si es falsa desde el principio, **nunca** se ejecuta.
- `do-while` : Primero ejecuta el código una vez, y *luego* comprueba la condición. Te asegura que el bloque de código se va a ejecutar **como mínimo una vez**.

```
fun main() {
    var porcionesComidas = 0

    // Bucle while normal
    while (porcionesComidas < 3) {
        println("Me como una porción")
        porcionesComidas++ // Esto suma 1 a la variable (es lo mismo que porcionesComidas = porcionesComidas + 1)
    }
}
```

6. Ejercicios

Abre tu Kotlin Playground y vamos a machacar lo aprendido.

6.1. Los dados

Crea un minijuego donde ganas si al lanzar dos dados sacas el mismo número. Si son iguales, imprime `"¡Has ganado :)"`. Si no, `"Has perdido :("`.

(Nota: Para generar números aleatorios usaremos una librería nativa de Kotlin llamada `Random`).

```kotlin
import kotlin.random.Random

fun main() {
    // Genera un número aleatorio entre 0 y 5.
    val dado1 = Random.nextInt(6)
    val dado2 = Random.nextInt(6)

    println("Dado 1: $dado1 | Dado 2: $dado2")
    // Escribe tu código (if/else) a partir de aquí:

}
```

6.2. Botones de consola

Usando un `when` que actúe como **expresión** (asignando su resultado o metiéndolo directo en un `println`), haz que el programa imprima la acción correspondiente al botón pulsado:

- A -> «Sí»
- B -> «No»
- X -> «Menú»
- Y -> «Nada»
- Cualquier otro -> «No existe ese botón»

```kotlin
fun main() {
    val boton = "A"

    // Escribe tu when aquí dentro del println
    println(
        // ...
    )
}
```

6.3. Comiendo Pizza

Tienes un código muy feo que cuenta porciones de pizza repitiendo líneas a mano. Conviértelo en un bucle `while` que cuente automáticamente hasta llegar a las 8 porciones.

```kotlin
fun main() {
```

```
    var porciones = 0
    // ¡Borra este desastre y usa un bucle while!
    porciones++
    println("Solo hay $porciones porción/es de pizza :(")
    porciones++
    println("Solo hay $porciones porción/es de pizza :(")
    // ... así hasta 7 ...

    // Al salir del bucle debe imprimir esto (y la variable debe valer 8):
    println("¡Tenemos $porciones porciones! ¡Una pizza entera! :D")
}
```

6.4. El clásico reto FizzBuzz

Este es uno de los ejercicios más famosos en las entrevistas de programación junior.

Escribe un programa que imprima los números del 1 al 100, pero:

- Si el número es divisible por 3, imprime la palabra `"fizz"` en lugar del número.
- Si el número es divisible por 5, imprime `"buzz"`.
- Si el número es divisible por 3 **Y** por 5 (es decir, divisible por 15), imprime `"fizzbuzz"`.

Pista: Usa un bucle `for` del 1 al 100. Dentro, usa un `when` sin argumento para evaluar condiciones usando el operador Módulo `%` (que te da el resto de una división. Si `numero % 3 == 0`, es que es divisible por 3).

```
fun main() {
    // Escribe tu código aquí (for + when)
}
```

7. Soluciones a los ejercicios

No hagas trampas. Échale un vistazo a las soluciones únicamente cuando te hayas peleado un buen rato con el código.

7.1. Los dados

Raw Copy Extern

```
import kotlin.random.Random

fun main() {
    val dado1 = Random.nextInt(6)
    val dado2 = Random.nextInt(6)
    println("Dado 1: $dado1 | Dado 2: $dado2")

    // Comprobamos la igualdad con ==
    if (dado1 == dado2) {
        println("¡Has ganado :)")
    } else {
        println("Has perdido :(")
    }
}
```

7.2. Botones de consola

```
fun main() {
    val boton = "A"

    println(
        when (boton) {
            "A" -> "Sí"
            "B" -> "No"
            "X" -> "Menú"
            "Y" -> "Nada"
            else -> "No existe ese botón"
        }
    )
}
```

7.3. La pizza

```
fun main() {
    var porciones = 0

    while (porciones < 7) {
        porciones++
        println("Solo hay $porciones porción/es de pizza :(")
    }

    porciones++ // Sumamos la octava porción al salir
    println("¡Tenemos $porciones porciones! ¡Una pizza entera! :D")
```

```
}
```

7.4. El clásico reto FizzBuzz

```
fun main() {
    for (numero in 1..100) {
        println(
            // Al usar when SIN variable, podemos evaluar condiciones booleanas libres
            // ¡El orden importa! Hay que comprobar el 15 primero.
            when {
                numero % 15 == 0 -> "fizzbuzz"
                numero % 3 == 0 -> "fizz"
                numero % 5 == 0 -> "buzz"
                else -> numero // Si no cumple ninguna, imprime el número normal
            }
        )
    }
}
```