Tipos básicos en Kotlin: Inferencia, números, textos y operaciones

1. Introducción

En la unidad anterior vimos cómo hacer nuestro primer «Hola Mundo» y cómo usar variables (`val` y `var`). Pero nos dejamos un detalle importantísimo en el tintero: **¿qué tipo de información estamos guardando exactamente en esas variables?**

En Kotlin, **absolutamente todo tiene un tipo**. Los tipos son fundamentales porque le dicen al compilador (el «cerebro» que lee tu código) qué puedes y qué no puedes

hacer con una variable. Por ejemplo, puedes multiplicar dos números, pero no puedes multiplicar dos palabras, ¿verdad?

Ahora vamos a desgranar los tipos básicos de Kotlin, a aprender a operar con ellos y a descubrir cómo Kotlin nos facilita la vida con su «magia» deductiva.

2. La inferencia de tipos: Kotlin es muy listo

Si recuerdas la unidad anterior, declarábamos variables así:

```
var clientes = 10
```

En ningún momento le dijimos a Kotlin: *«Oye, que sepas que `clientes` es un número entero»*. Sin embargo, Kotlin lo supo al instante. Esta capacidad se llama **Inferencia de Tipos** (*Type Inference*).

Como a la variable `clientes` le asignamos un `10`, Kotlin deduce inmediatamente que su tipo es numérico, concretamente un `Int` (entero). Gracias a esto, el compilador sabe que puedes realizar operaciones matemáticas con esta variable.

Mira este ejemplo de **asignaciones compuestas** (una forma abreviada de hacer matemáticas):

```
fun main() {
    var clientes = 10

    // Vienen 3 clientes más (clientes = clientes + 3)
    clientes += 3  // Ahora hay 13

    // Se van 5 clientes
    clientes -= 5  // Ahora hay 8

    // El negocio explota y multiplicamos los clientes por 2
    clientes *= 2  // Ahora hay 16

    // Dividimos a los clientes en 4 grupos
    clientes /= 4  // Ahora hay 4 en cada grupo

    println(clientes)
    // Resultado: 4
}
```

3. Los tipos básicos de Kotlin: La lista completa

Aunque Kotlin infiere los tipos, a veces querrás (o necesitarás) ser explícito. Para declarar un tipo manualmente, se usan los dos puntos `:` después del nombre de la variable.

A continuación mostramos los tipos fundamentales.

3.1. Números enteros (sin decimales)

- `Int` : Es el estándar para números sin decimales. (Ej: `val año: Int = 2024`)
- `Long` : Se usa para números ridículamente grandes. Si un número es demasiado grande para un `Int` , Kotlin lo convertirá en `Long` automáticamente. También puedes forzarlo añadiendo una `L` mayúscula al final. (Ej: `val estrellas: Long = 9876543210L`)
- `Byte` y `Short` : Se usan en casos muy específicos para ahorrar memoria con números pequeños. Raramente los usarás al empezar.
- *Nota: Kotlin también tiene versiones «Unsigned» (sin signo, es decir, solo positivos) como `UInt` o `ULong` marcados con una `u` (Ej: `val puntos: UInt = 100u`).*

3.2. Números con decimales (coma flotante)

- `Double` : Es el estándar de Kotlin para números con decimales. Tiene una precisión doble, es decir, admite muchísimos decimales. (Ej: `val precio: Double = 19.99`)
- `Float` : Ocupa menos memoria pero es menos preciso. Para indicarle a Kotlin que quieres un `Float` y no un `Double` , **debes** añadir una `f` o `F` al final del número. (Ej: `val temperatura: Float = 24.5f`)

3.3. Booleanos (verdadero o falso)

- `Boolean` : Solo puede tener dos valores: `true` (verdadero) o `false` (falso). Es la base de la lógica en programación (Ej: `val estaEncendido: Boolean = true`).

3.4. Caracteres y cadenas de texto

- `Char` : Representa un **único** carácter (una sola letra, número o símbolo). Se escribe entre **comillas simples** `' '` . (Ej: `val inicial: Char = 'J'`)
- `String` : Representa una cadena de texto (muchos caracteres juntos). Se escribe entre **comillas dobles** `" "` . (Ej: `val mensaje: String = "¡Hola, mundo!"`)

4. Declarar ahora, inicializar después

A veces sabes qué tipo de dato vas a guardar, pero aún no tienes el valor exacto. Kotlin te permite declarar una variable y asignarle su valor más tarde.

Eso sí, en estos casos **es obligatorio especificar el tipo explícitamente**, porque Kotlin no tiene un valor inicial del que deducirlo:

```
1.  fun main() {
2.      // Declaramos la variable especificando el tipo explícitamente,
    pero sin darle valor
3.      val d: Int
4.
5.      // Más adelante en el código, la inicializamos
6.      d = 3
7.
8.      println(d) // Imprime: 3
9.  }
```

5. La seguridad de Kotlin frente a errores

¿Qué pasa si intentas imprimir la variable `d` *antes* de darle un valor? En otros lenguajes tu programa explotaría (el temido *NullPointerException* o imprimiría basura de la memoria). **Kotlin no te deja hacerlo**. El código directamente se pondrá en rojo y no compilará, mostrándote el error: *«Variable 'd' must be initialized»*. ¡Un salvavidas enorme!

6. Ejercicios

Abre tu editor o el Kotlin Playground y pon a prueba lo que acabas de aprender.

6.1. El tipado explícito

Kotlin puede inferir los tipos, pero en este ejercicio queremos que seas tú quien los escriba. Modifica el siguiente código para añadir explícitamente (`: Tipo`) el tipo de dato correcto a cada variable:

```
1.  fun main() {
2.      val a = 1000
3.      val b = "mensaje de registro"
4.      val c = 3.14
5.      val d = 100_000_000_000_000 // Fíjate, los guiones bajos sirven
    para leer mejor los números grandes
6.      val e = false
7.      val f = '\n' // Esto es un carácter especial que representa un
    "salto de línea"
8.  }
```

6.2. El cofre del tesoro

Crea un programa que simule el oro de un jugador en un videojuego usando una variable mutable (`var`) llamada `monedas` :

1. El jugador empieza con 50 monedas.
2. Encuentra un cofre mágico y su oro se multiplica por 3. (Usa asignación compuesta `*=`).
3. Compra una espada que cuesta 80 monedas. (Usa `-=`).
4. Imprime el resultado final: `"Tras la aventura, te quedan X monedas"` .

6.3. Detecta el error

El siguiente código tiene un problema y Kotlin se quejará si intentas ejecutarlo. Corrígelo para que funcione y se imprima la edad correctamente.

```
fun main() {
    val edadUsuario: Int
    println("La edad del usuario es $edadUsuario")
    edadUsuario = 25
}
```

6.4. Cuidado con las divisiones

Si divides dos variables `Int` (ej: `10 / 3`), Kotlin devuelve un número entero (`3`) y se come los decimales.

Crea dos variables llamadas `dividendo` (valor 10) y `divisor` (valor 3). Haz que sean de tipo `Double` explícitamente para que, al imprimirlas usando una plantilla de cadena, el resultado sea con decimales (`3.3333333333333335`).

7. Soluciones a los ejercicios

¡No mires hasta que no te hayas peleado un rato con el código!

7.1. El tipado explícito

```
fun main() {
    val a: Int = 1000
    val b: String = "mensaje de registro"
    val c: Double = 3.14  // Al tener decimales y no tener 'f', es Double
    val d: Long = 100_000_000_000_000 // Es demasiado grande para ser Int
    val e: Boolean = false // Verdadero o falso
    val f: Char = '\n' // Comillas simples indican que es un solo Char
}
```

7.2. El cofre del tesoro

```
fun main() {
    var monedas = 50
    monedas *= 3
```

```
    monedas -= 80
    println("Tras la aventura, te quedan $monedas monedas")
    // Imprimirá 70
}
```

7.3. Detecta el error

```
fun main() {
    val edadUsuario: Int
    edadUsuario = 25 // ¡Había que inicializarla ANTES de leerla!
    println("La edad del usuario es $edadUsuario")
}
```

7.4. Cuidado con las divisiones

```
fun main() {
    // Para forzar que un número entero se trate como decimal,
ponemos .0
    val dividendo: Double = 10.0
    val divisor: Double = 3.0

    println("El resultado exacto es ${dividendo / divisor}")
}
```