Guía Definitiva de colecciones en Kotlin: listas, conjuntos y mapas explicados desde cero

1. Introducción

En las unidades anteriores aprendimos a guardar un dato individual en una variable (como un número o un texto). Pero, ¿qué pasa si estás creando una app y necesitas guardar los nombres de **100 usuarios**? ¿Vas a crear 100 variables distintas? ¡Por supuesto que no!

Para eso existen las **colecciones**. Son estructuras que nos permiten agrupar múltiples datos bajo un mismo nombre para procesarlos más tarde.

En Kotlin, la regla de seguridad que vimos con `val` y `var` se mantiene: **las colecciones pueden ser de solo lectura (inmutables) o mutables (modificables)**. Kotlin nos ofrece tres tipos principales. ¡Vamos a destriparlas!

2. Listas (Lists): El cajón ordenado

Una Lista (`List`) es exactamente lo que imaginas: una colección de elementos ordenados uno detrás de otro.

- **Tienen un orden estricto** (el primero, el segundo, el tercero...).
- **Permiten duplicados** (puedes tener el mismo elemento varias veces).

2.1. Crear listas

Para crear una lista de **solo lectura** usamos `listOf()`. Si queremos una lista a la que podamos añadir o quitar cosas en el futuro, usamos `mutableListOf()`.

```
1.  fun main() {
2.      // Lista de solo lectura (Kotlin deduce que es de tipo String)
3.      val formas = listOf("triángulo", "cuadrado", "círculo")
4.      println(formas) // [triángulo, cuadrado, círculo]
5.
6.      // Lista mutable (Aquí le decimos explícitamente el tipo
    <String>)
7.      val formasMutables: MutableList<String> =
    mutableListOf("triángulo", "cuadrado", "círculo")
8.  }
```

2.2. Profundizando: El índice cero y los errores comunes

Como las listas están ordenadas, cada elemento tiene una posición o «índice». **¡Ojo! En programación, siempre empezamos a contar desde el cero.**

Piensa en los **ascensores en España**: cuando entras al edificio desde la calle, estás en la Planta Baja (índice `0`). Si subes un piso, llegas a la Planta 1 (el índice `1` , que en realidad es el segundo nivel). El índice te dice **cuántos saltos das desde el principio**.

```
fun main() {
    val formas = listOf("triángulo", "cuadrado", "círculo")

    // Acceder por su posición (índice)
    println("El primer elemento es: ${formas[0]}") // triángulo

    // ¡CUIDADO! El terror de los novatos: IndexOutOfBoundsException
    // println(formas[3]) -> ¡El programa explotará porque no hay un 4º elemento!
}
```

Kotlin también nos regala funciones súper útiles para no tener que lidiar siempre con los números:

```
fun main() {
    val formas = listOf("triángulo", "cuadrado", "círculo")

    println("El primer elemento: ${formas.first()}") // triángulo
    println("El último elemento: ${formas.last()}") // círculo
    println("Total de elementos: ${formas.count()}") // 3

    // Comprobar si algo existe con la palabra 'in'
    println("círculo" in formas) // true
}
```

2.3. Modificar listas mutables y el «truco del candado»

Si tu lista es `MutableList`, usas `.add()` para añadir y `.remove()` para borrar.

```
val carrito: MutableList<String> = mutableListOf("Manzanas", "Pan")
carrito.add("Leche") // [Manzanas, Pan, Leche]
carrito.remove("Pan") // [Manzanas, Leche]
```

A veces tienes una lista mutable, pero se la vas a pasar a otra parte de tu código y no quieres que se modifique por accidente. Puedes «disfrazarla» de solo lectura:

```
val inventarioMutable: MutableList<String> = mutableListOf("Espada", "Poción")
val inventarioBloqueado: List<String> = inventarioMutable // ¡Candado puesto! Ahora no se puede modificar.
```

3. Conjuntos (Sets): El VIP de la exclusividad

Un conjunto (`Set`) es parecido a una lista, pero con dos diferencias vitales:

- **No tienen orden** (no puedes pedir el elemento `[0]`).
- **NO permiten elementos duplicados**. Son únicos y exclusivos.

Se crean con `setOf()` y `mutableSetOf()`.

```
fun main() {
    // Fíjate que ponemos "cereza" dos veces
    val frutas = setOf("manzana", "plátano", "cereza", "cereza")

    // Al imprimir, ¡la segunda cereza ha desaparecido mágicamente!
    println(frutas) // [manzana, plátano, cereza]
    println("Hay ${frutas.count()} frutas únicas") // 3
}
```

3.1. Profundizando: ¿Por qué usar Sets? ¡Por la velocidad!

Si quieres saber si el email «maria@email.com» está en una **Lista** de 1 millón de usuarios, el ordenador mirará uno a uno. Es lentísimo.

Si usas un **Set**, Kotlin usa matemáticas internas (una función *Hash*) para saber **exactamente** dónde está guardado. Lo encuentra al instante. ¡Usa Sets cuando trabajes con muchos datos únicos!

```
val millonesDeUsuarios = setOf("user1@mail.com", "user2@mail.com")
if ("maria@mail.com" in millonesDeUsuarios) {
    println("¡El usuario ya existe!") // Comprobación súper rápida
}
```

4. Mapas (Maps): El diccionario de datos

Un Mapa (`Map`) almacena datos en **pares Clave-Valor** (Key-Value).

Piénsalo como la carta de un restaurante: el nombre del plato es la *clave* y su precio es el *valor*.

- **Las claves (`keys`) deben ser únicas**.
- **Los valores (`values`) sí pueden repetirse**.

Se crean con `mapOf()` y `mutableMapOf()`. Usamos la palabrita `to` para unir la pareja.

```
fun main() {
    // Especificamos explícitamente: Claves String, Valores Int
    val menuZumos: MutableMap<String, Int> = mutableMapOf("manzana" to 3, "kiwi" to 4)

    // Leer un valor usando su clave entre corchetes
    println("El zumo de manzana cuesta: ${menuZumos["manzana"]}€") // 3€
}
```

4.1. Profundizando: Nulos y sobreescritura

¿Qué pasa si buscas algo que no existe en el menú? Kotlin te devuelve `null` (nulo, vacío). Así evita que tu aplicación se cuelgue por un error.

```
println(menuZumos["piña"]) // Imprime: null
```

¿Y si añado una clave que ya existe? **Se sobreescribe el valor antiguo**. Es la forma oficial de actualizar datos:

```
menuZumos["manzana"] = 5 // ¡Inflación! Actualizamos el precio a 5€
menuZumos["coco"] = 6    // Como "coco" no existe, lo añade nuevo al mapa
menuZumos.remove("kiwi") // Eliminamos el kiwi

println(menuZumos.keys)   // [manzana, coco]
println(menuZumos.values) // [5, 6]
```

5. Ejercicios

Abre el Kotlin Playground e intenta resolver estos ejercicios para asentar lo aprendido.

5.1. Sumando listas

Tienes una lista de números «verdes» y otra de «rojos». Imprime cuántos números hay **en total** sumando el tamaño de ambas listas.

```
fun main() {
    val numerosVerdes = listOf(1, 4, 23)
    val numerosRojos = listOf(17, 2)
    // Escribe tu código aquí:
}
```

5.2. El protocolo de red

Tienes un Set con protocolos en mayúsculas. Un usuario pide «smtp» en minúsculas. Comprueba si está soportado (debe devolver un `Boolean`).

Pista: Usa `.uppercase()` en la petición para pasarla a mayúsculas antes de buscar en el Set con `in`.

```
fun main() {
    val SOPORTADOS = setOf("HTTP", "HTTPS", "FTP")
    val peticion = "smtp"
    val estaSoportado = // Escribe tu código aquí
    println("Soporte para $peticion: $estaSoportado")
}
```

5.3. Diccionario de números

Crea un **Map** que relacione los números del 1 al 3 con su nombre escrito («uno», «dos», «tres»). Imprime cómo se escribe el número 2 buscándolo en el mapa.

```
fun main() {
    val numeroAPalabra = // Escribe tu código aquí
```

```
    val n = 2
    // Imprime el resultado
}
```

5.4. El inventario del héroe

Crea una **lista mutable** llamada `inventario` con: «Espada», «Escudo», «Poción».

Añade un «Arco». Elimina la «Poción». Finalmente, imprime el inventario y cuántos objetos tiene usando `.count()`.

6. Soluciones a los ejercicios

¡No mires hasta haberlo intentado!

6.1. Sumando listas

```
fun main() {
    val numerosVerdes = listOf(1, 4, 23)
    val numerosRojos = listOf(17, 2)
    val total = numerosVerdes.count() + numerosRojos.count()
    println("Total: $total números")
}
```

6.2. El protocolo de red

```
fun main() {
    val SOPORTADOS = setOf("HTTP", "HTTPS", "FTP")
    val peticion = "smtp"
    val estaSoportado = peticion.uppercase() in SOPORTADOS
    println("Soporte para $peticion: $estaSoportado") // false
}
```

6.3. Diccionario de números

```
fun main() {
    val numeroAPalabra = mapOf(1 to "uno", 2 to "dos", 3 to "tres")
    val n = 2
    println("El número $n se escribe como '${numeroAPalabra[n]}'")
}
```

6.4. El inventario del héroe

```
fun main() {
    val inventario = mutableListOf("Espada", "Escudo", "Poción")
    inventario.add("Arco")
    inventario.remove("Poción")
    println("Inventario: $inventario")
    println("Tienes ${inventario.count()} objetos.")
}
```