Guía de ejercicios - Kotlin para el desarrollo de aplicaciones (I)

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

Tabla de contenidos

¡Comencemos!

Actividad guiada: Scope functions

Supongamos que tenemos un listado de “Ítems” y queremos hacer algunos cambios y filtrar el resultado dependiendo sus características.

```
data class Item(
    val id: Int,
    val name: String,
    val description: String? = null,
    var isAvailable: Boolean,
    var isEnabled: Boolean = true,
)
```

ID	nombre	descripción	disponible	habilitado
100	Item1	N/A	SI	SI
101	Item2	Descripción	NO	SI
102	Item3	N/A	SI	NO

1. Creamos un mapa con los ítems especificados.

```
val products = mapOf(
    1 to Item(id = 100, name = "Item1", isAvailable = true),
    2 to Item(id = 101, name = "Item2", description = "descripción",
isAvailable = false),
    3 to Item(id = 102, name = "Item3", isAvailable = true, isEnabled =
false)
    )
```

2. Necesitamos buscar todos los ítems que están NO disponibles y cambiarlos a disponibles. Para hacer esto podemos hacer uso de **`apply`** y **`forEach`**. Recordemos que apply retorna **`this`**.

```
items.apply {
        this.values.forEach {
            if (!it.isAvailable) {
                it.isAvailable = true
            }
        }
    }
```

3. Imprimimos por pantalla y verificamos que los cambios se hayan aplicado donde corresponda:

```
println(items)

resultado:
{
    1=Item(id=100, name=Item1, description=null, isAvailable=true,
isEnabled=true),
    2=Item(id=101, name=Item2, description=descripción,
isAvailable=true, isEnabled=true),
   3=Item(id=102, name=Item3, description=null, isAvailable=true,
isEnabled=false)
}
```

4. En el resultado se puede ver que ahora todos los ítems están disponibles.

5. Ahora, necesitamos filtrar la lista de items y mostrar solo los que **NO** están habilitados. Para eso usamos la función filter, lo aplicamos sobre la lista de items y guardamos el resultado en una nueva variable

```
val filteredProducts = items.filter {
    !it.value.isEnabled
}
```

6. Luego imprimimos por pantalla y verificamos el resultado:

```
println(filteredProducts)
```

```
Resultado:
{3=Item(id=102, name=Item3, description=null, isAvailable=true,
isEnabled=false)}
```

¡Manos a la obra! - Map, List y Set

Primero, un ejemplo antes de empezar:

Supongamos que tenemos un conjunto de enteros y queremos multiplicar cada uno de los ítems por dos.

```
fun main() {
    val numbers = mapOf(1 to 1, 2 to 2, 3 to 3, 4 to 4, 5 to 5)
    val result = numbers.map {
        it.value * 2
    }
    println(result)
}

resultado: [2, 4, 6, 8, 10]
```

Ahora sí, empecemos. Dada la siguiente data clase:

```
data class Product(
    val id: Int,
    val name: String,
    var available: Boolean
)
```

1. Crea una lista que permita almacenar los siguientes productos, luego imprime por pantalla la clase y compara con el resultado.

ID	nombre	disponible
1	Lápiz	SI
2	Hojas de oficio	NO

3	Corchetera	SI

2. Haciendo uso de scope functions, actualiza la lista anterior y cambia el estado de "**Hojas de oficio**" a **disponible** , luego imprime por pantalla el resultado y compara con el resultado.

3. Cambia cada uno de los nombres de los productos a mayúscula, puedes hacer uso de la función **`uppercase`** de la librería estándar.

4. Necesitamos empezar a guardar la cantidad de existente de cada producto. Modifica la clase Product agregando un nuevo campo tipo **`Int`** , nombrarlo **stock**, con valor por defecto **0**. Compara la clase con la respuesta.

5. Actualiza los productos con los siguientes datos (puede ser de forma manual), luego, haciendo uso de **`map`**, suma la cantidad total de todos los productos e imprime por pantalla

ID	nombre	disponible	cantidad
1	Lápiz	SI	100
2	Hojas de oficio	NO	400
3	Corchetera	SI	30
4	Hojas de carta	SI	400

¡Manos a la obra! - Filtrando y ordenando datos

Utiliza el siguiente snippet para completar los ejercicios de Filtrando y Ordenando:

```
val products = mutableMapOf(
        1 to Product(100, "Lapiz", true, 20),
        2 to Product(101, "Hojas de oficio", false, 150),
        3 to Product(102, "Corchetera", true, 100),
        4 to Product(103, "Hojas de carta", true, 150),
    )
```

1. Aplicando lo aprendido de Filter, muestra una lista de los productos que **NO** están disponibles.

2. Ordena los productos de stock en forma descendiente

3. Haciendo uso del resultado anterior, imprime por pantalla el nombre del producto y el stock correspondiente, recuerda que debe estar ordenado de forma descendiente, puedes guiarte con el siguiente ejemplo:

```
Nombre: Lápiz, Stock: 20
Nombre: Corchetera, Stock: 100
.
.
.
```

Preguntas de proceso

Reflexiona:

- A partir de lo aprendido hasta aquí ¿Qué contenido se me ha hecho más difícil? ¿Y cuál más sencillo?
- ¿Existe algún ejercicio de los resueltos previamente que deba repetir o desarrollar otra vez para poder dominarlo mejor?
- Nombra al menos un uso que pueda darle a lo aprendido hasta ahora.

Preguntas de cierre

- ¿Cuándo puede ser útil utilizar map?
- ¿Cuál es la diferencia entre filter y sorting?
- ¿Cuál es la diferencia entre `let y run` ?
- Respecto a las funciones estándar o scope, cuál es la única que recibe como parámetro "`this`"
- ¿En qué situación se podría ocupar el ordenar una lista de forma aleatoria?

Referencias bibliográficas

https://kotlinlang.org/ documentación oficial de Kotlin