FUNCIONES Y LISTAS

Kotlin es un lenguaje orientado a objetos pero introduce características existentes en los lenguajes funcionales que nos permiten crear un código más claro y expresivo.

Una de las características del paradigma de la programación funcional son las funciones de orden superior.

Las funciones de orden superior son funciones que pueden recibir como parámetros otras funciones y/o devolverlas como resultados.

Una función de orden más alto es solo una función que toma otra función (o expresión lambda) como parámetro, devuelve una función, o hace ambos.

```
fun operar(v1: Int, v2: Int, fn: (Int, Int) -> Int) : Int
{
   return fn(v1, v2)
}
fun sumar(x1: Int, x2: Int) = x1 + x2
fun restar(x1: Int, x2: Int) = x1 - x2
fun multiplicar(x1: Int, x2: Int) = x1 * x2
fun dividir(x1: Int, x2: Int) = x1 / x2

fun main(parametro: Array<String>) {
   val resu1 = operar(10, 5, ::sumar)
   println("La suma de 10 y 5 es $resu1")
   val resu2 = operar(5, 2, ::sumar)
   println("La suma de 5 y 2 es $resu2")
   println("La resta de 100 y 40 es ${operar(100, 40, ::restar)}")
   println("El producto entre 5 y 20 es ${operar(5, 20, ::multiplicar)}")
   println("La división entre 10 y 5 es ${operar(10, 5, ::dividir)}")
}
```

La única función de orden superior es la llamada "operar".

```
fun operar(v1: Int, v2: Int, fn: (Int, Int) -> Int) : Int
{
   return fn(v1, v2)
}
```

El tercer parámetro de esta función se llama "fn" y es de tipo función. Cuando un parámetro es de tipo función debemos indicar los parámetros que tiene dicha función (en este caso tiene dos parámetros enteros) y luego del operador -> el tipo de dato que retorna esta función.

```
fn: (Int, Int) -> Int
```

o Unit en caso de que no se devuelva ningún tipo

```
fn: (Int, Int) -> Unit
```

Utilizar / Retornar

```
  return fn(v1, v2)
```

Listas.
Podemos construir una lista donde todos sus elementos son los mismos.

```
val numbersInt = listOf(4,3,2)
```

O una de múltiples tipos (Any)

```
val my list = listOf(4,"lalalala", "ll",2)
```

Estructuras de dos dimensiones con el tipo hashMapsOf

```
val months = hashMapOf("Enero" to 1, "Febrero" to 2)
```

Filtrar es una de las opciones más populares en el procesamiento de colecciones. Las condiciones de filtrado son definidas por predicados (funciones lambda con elementos de condición y que devuelven un resultado booleano).
Las librerías por defecto incluyen funciones de extensión que permiten filtrar colecciones de una sola llamada.

```
val numbers = listOf("one", "two", "three", "four")
val longerThan3 = numbers.filter { it.length > 3 }
println(longerThan3)
```

Esto nos da como resultado [“three”, “four”].

filterIndexed() nos permite trabajar también con los índices de los elementos. filterNot() nos devuelve los elementos que no cumplen con la condición, filterIsInstance() nos da una colección de elementos del tipo indicado, y filterNotNull() nos devuelve objetos no nulos.

Algunos predicados default son any, none y all. Que devuelven true si almenos algún elemento cumple con la condición, si ninguno o si todos, respectivamente.

FUENTES DE INFORMACIÓN

Kotlin Ya: Funciones de orden superior.

https://www.tutorialesprogramacionya.com/kotlinya/detalleconcepto.php?punto=36&codigo=36&inicio=30

Recursividad List, anncode

https://platzi.com/clases/1543-kotlin/19428-recursividad-list/

Filtering Collections - Kotlin

https://kotlinlang.org/docs/reference/collection-filtering.html