



FUNDAMENTOS EN KOTLIN



¿QUÉ VAMOS A VER?

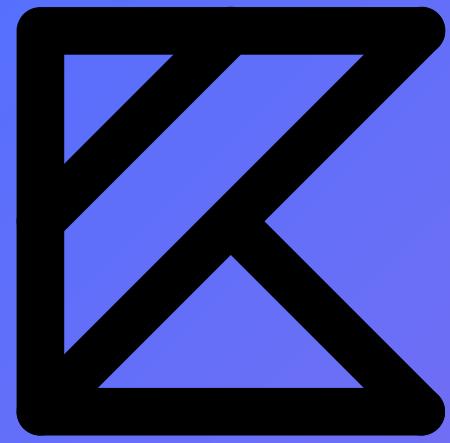


- Bucle While
- Arrays
- Listas
- Bucle For
- Foreach
- Condicional IF
- When
- Funciones
- Clases
- Programación funcional
- Filter
- Map
- Reduce



¡VAMOS A COMENZAR!





BUCLE WHILE



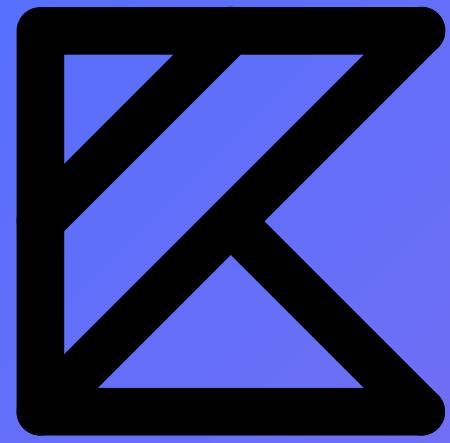
BUCLE WHILE

- **Recordemos:** Para que nuestro código pueda entrar en un ciclo while, la condición debe ser **verdadera**.
- Por lo tanto, estaremos dentro del ciclo mientras la condición no cambie su estado booleano a **falso**.
- La sintaxis del bucle no difiere de lo que hemos utilizado hasta ahora, es más, también podemos agregar un **break** dentro del código, para terminar la iteración.

```
var num = 1
while (num <= 10) {
    println(num)
    num++
}
```

```
while (true) {
    try {
        println("Escribe el primer valor")
        n1 = readln().toDouble()

        println("Escribe el segundo valor")
        n2 = readln().toDouble()
        break
    } catch (e: NumberFormatException) {
        println("Error: escribe un número valido - $e")
    }
}
```



ARRAYS



ARRAYS

- Kotlin nos entrega una facilidad extra al momento de querer crear un array, ya que su sintaxis es bastante simple.
- **arrayOf()**: Nos permite crear un arreglo que dentro puede contener cualquier tipo de dato.
- **intArrayOf()**: Podemos crear un arreglo exclusivo para almacenar datos de tipo Int.
- **.contentToString()**: Nos permite mostrar el contenido del array como si fuera un String, pero con la forma en que podemos visualizar el arreglo.

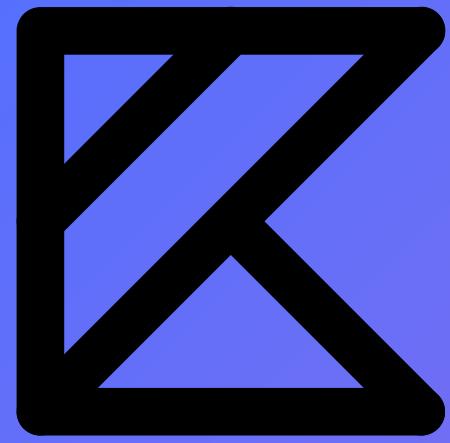




ARRAYS

- Ya sabemos que dentro de un arreglo cada dato va a tener su propio **índice**, comenzando **siempre en cero**.
- Bajo esa premisa, podemos acceder a los datos si sabes el indicador del que queremos visualizar.
- Es más, con su índice también vamos a tener la capacidad de modificar algún dato.
- La función **.size** nos va a permitir obtener la dimensión de un arreglo. Mientras que con **.sort()** lo podremos ordenar de menor a mayor.
- Es más, tendremos incluso la opción de mostrar **solo** los datos que no se encuentran duplicados, aplicando nuestro propio filter.





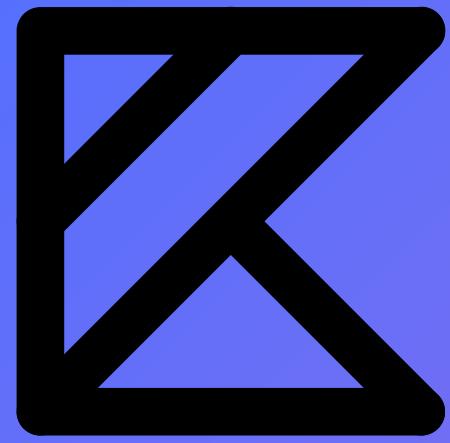
LISTAS



LISTAS

- Si de entrada comparamos las listas con los arrays, podemos decir que su sintaxis es mucho más fácil de manera, tanto para crear como para mostrar los datos que vamos a tener almacenados.
- Tendremos la opción de crear listas **mutables e inmutables**, cosa que básicamente se define con la función que la creamos desde un inicio.
- **listOf()** -> Lista inmutables
- **mutableListOf()** -> Lista mutable
- **Dato curioso:** En Kotlin las listas se utilizan para casos puntuales, pero los arrays están mejor optimizados para trabajar con datos provenientes de alguna BBDD.



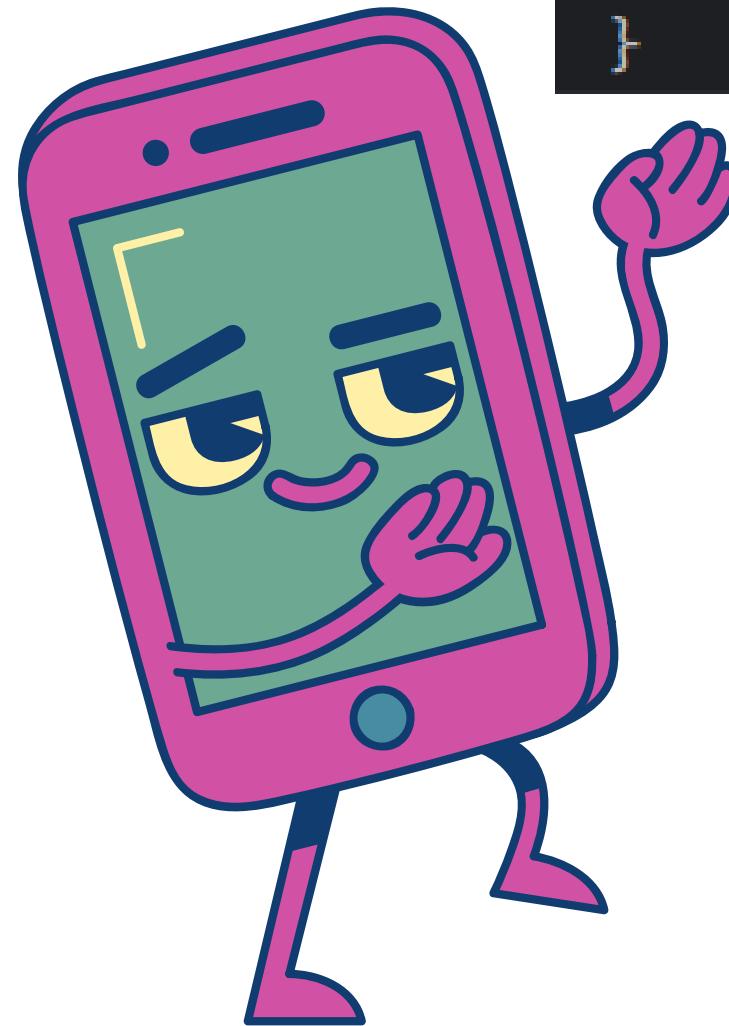


BUCLE FOR

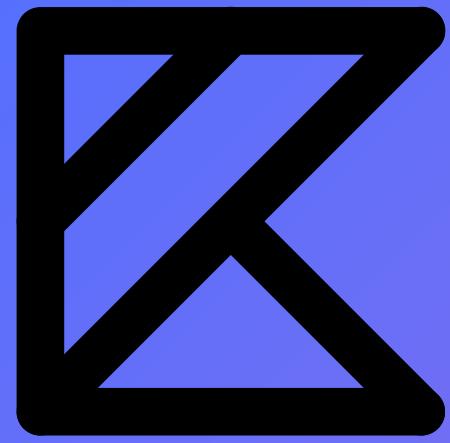


BUCLE FOR

- Con este tipo de ciclo vamos a tener el primer gran choque “cultural” entre dos lenguajes de programación, ya que la sintaxis que ofrece Kotlin presenta ciertas diferencias.
- Para declarar un bucle en for, lo podremos hacer a través de rangos, es decir, ya no vamos a necesitar seguir una estructura fija como en el ciclo for de Java.
- **i in 1..5** -> Esto quiere decir que tendremos cinco iteraciones que comenzarán de 1 hasta el 5 y la variable i va a representar a cada elemento del ciclo iterable.



```
for (i in 1 .. 5) {  
    println(i)  
}
```



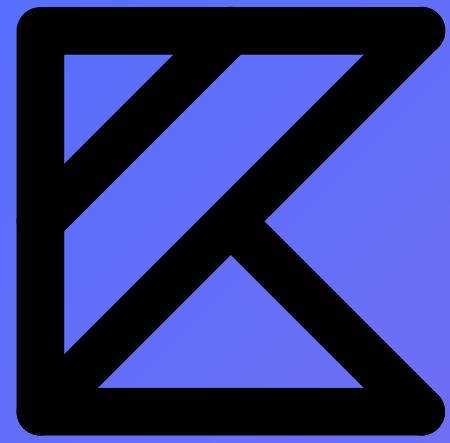
FOREACH



FOREACH

- En esencia, el Foreach intenta lograr exactamente lo que puede hacer un For, con la salvedad de que tiene una sintaxis mucho más simple.
- **it** -> Hace referencia al objeto de la estructura en que lo estemos utilizando. En el caso de usarse junto a un Foreach, hace referencia al elemento que nos encontramos en cada iteración del ciclo.





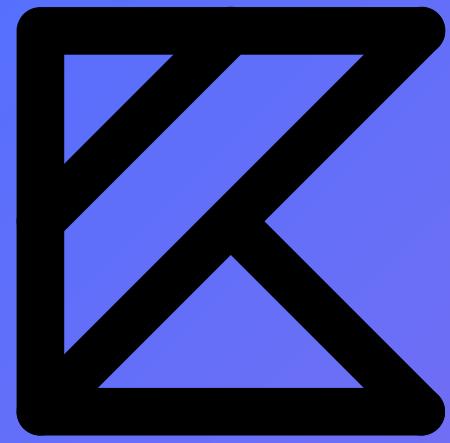
CONDICIONAL IF



CONDICIONAL IF

- Su base de funcionamiento no tiene nada de diferente respecto al **If/else** que conocimos en Java.
- En caso de que la condición de la estructura sea **True**, se ejecutará todo lo que esté en el primer bloque, de lo contrario, si la condición retorna un **False**, se ejecutará todo lo que se encuentre en el bloque luego del else.
- En Kotlin tendremos a nuestra disposición la condición de igualar por medio de 3 símbolos (**==**). Con esto se logra comprobar si el dato que estamos comparando además de ser iguales, pertenecen o no al mismo tipo de dato.



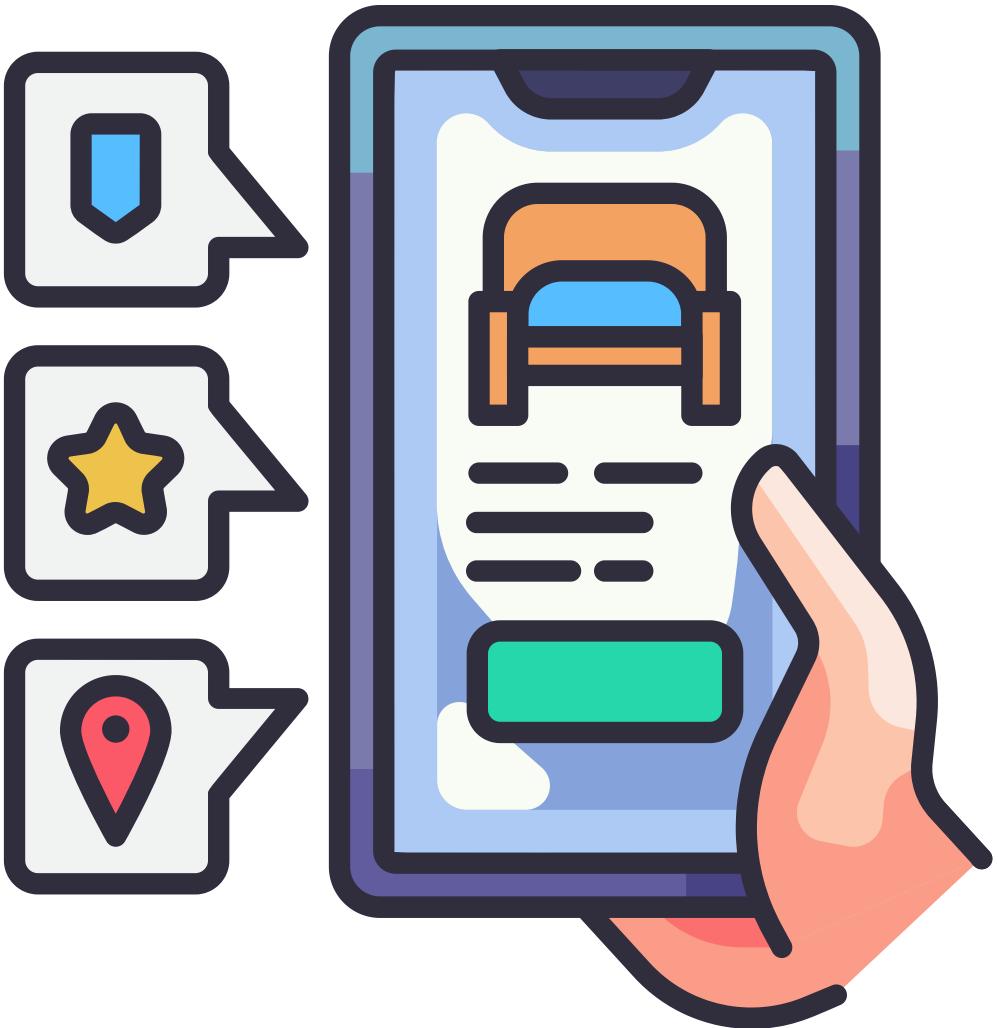


WHEN (SWITCH)

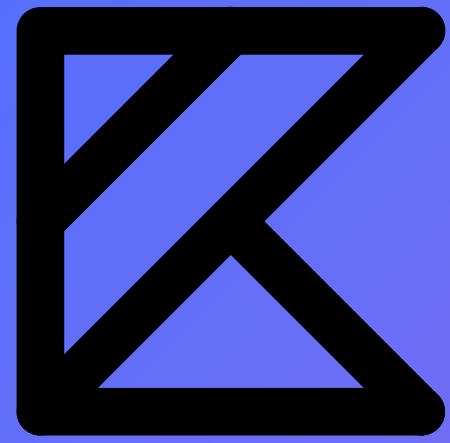


WHEN (SWITCH)

- El gran cambio de sintaxis en este objeto, es el when por switch, lo demás, funciona exactamente igual.
- No podemos negar que su sintaxis generalizada se parece mucho a la que nos ofrecía NetBeans en Java.
- Incluso, existe la oportunidad de ocupar rangos para tener una gama más amplia de selección.



```
val x = 5
when (x) {
    in 1 .. 10 -> println("Está dentro del rango")
    in 10 .. 20 -> println("Está fuera de rango")
}
```

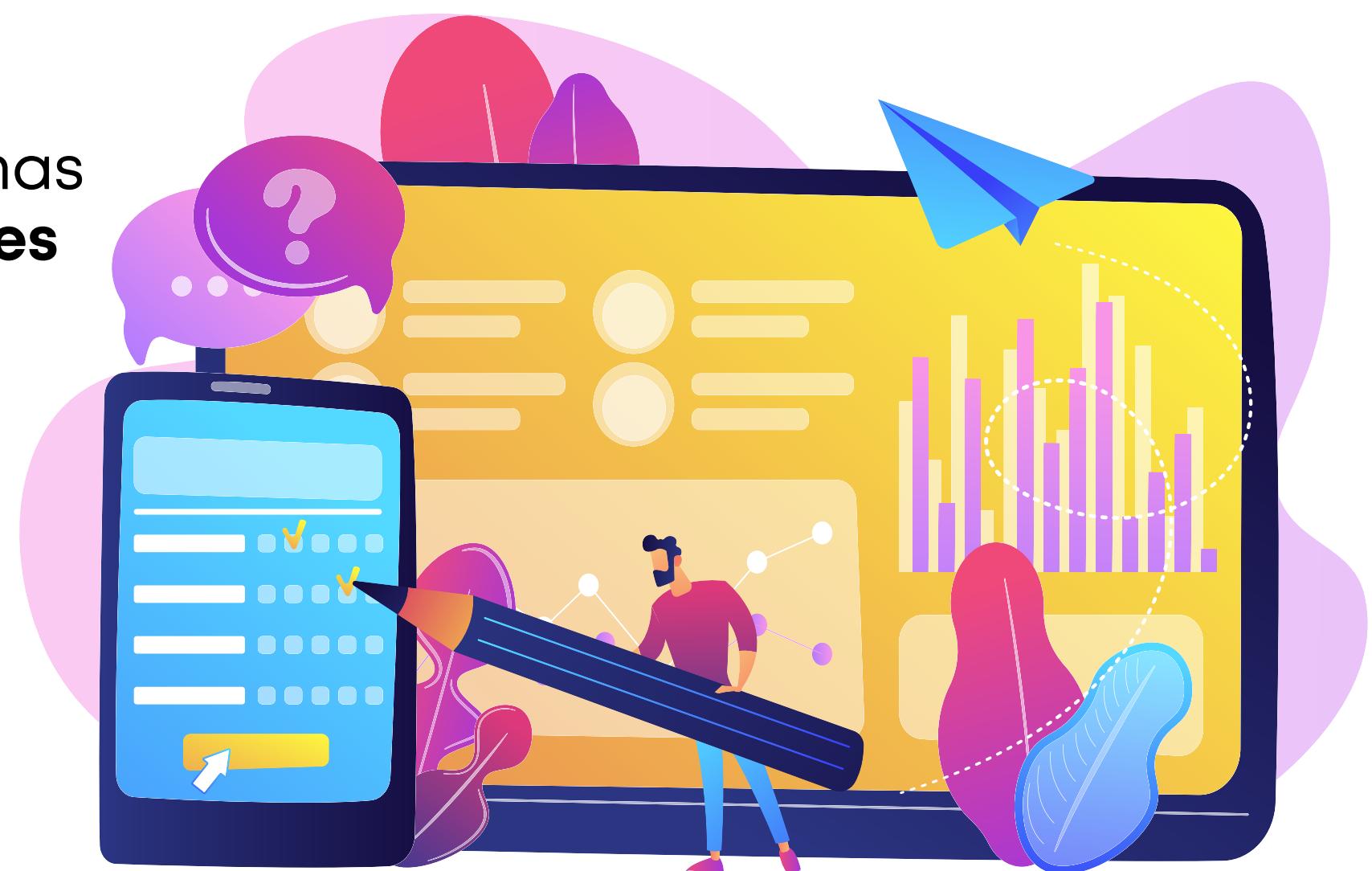


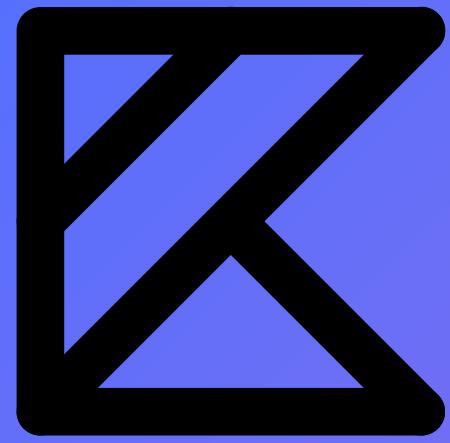
FUNCIONES



FUNCIONES

- También conocidas y nombradas como métodos, las funciones ser el eslabón principal a la hora de comenzar a fragmentar nuestro código y separar las utilidades del main, dejando así su pequeño código funcional.
- Dentro de las funciones vamos a tener diferentes formas de crearlas, entre las mas comunes nos encontraremos las **Funciones Lamda** y **con retorno**.





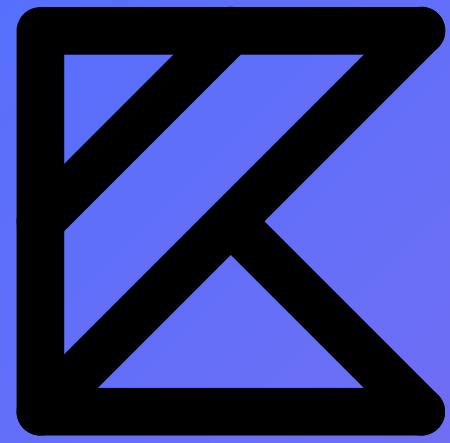
CLASES



CLASES

- Son básicas para la programación orientada a objetos dentro de Kotlin.
- Siempre que alguno de los métodos que usamos en nuestro código, genere un import, eso significa que la funcionalidad se encuentra dentro de otra clase.
- A la hora de crear una clase se recomienda dejar su nombre en plural, ya que va a representar a elementos particulares que llevarán el nombre en singular.
- Tendremos dos tipos de clases que van a depender netamente de sus constructores, primario o secundario.





PROGRAMACIÓN FUNCIONAL



PROGRAMACIÓN FUNCIONAL

- Es un paradigma de la programación, que se basa en el uso de funciones matemáticas.
- Algunas de sus funciones más comunes son: **filter, map, reduce, sorted**.
- Kotlin adopta la programación funcional, y tiene la cualidad de que puede combinarla con la programación orientada a objetos.
- Es importante tener que cuenta que Jetpack Compose está creado y trabaja bajo el paradigma funcional de programación declarativa, ya que todo lo hacemos a través de funciones.

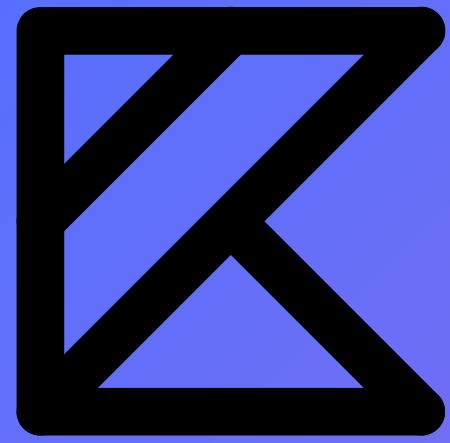




PROGRAMACIÓN FUNCIONAL

- La **programación imperativa** se acerca al paso a paso que programamos comúnmente, ya que es algo que se centra netamente en el proceso del código.
- En cambio, la **programación declarativa** se centra netamente en el resultado, por eso muchos de los procesos solo se cumplen llamando a una función, sin importar lo que realmente ocurrió para llegar a ese resultado.





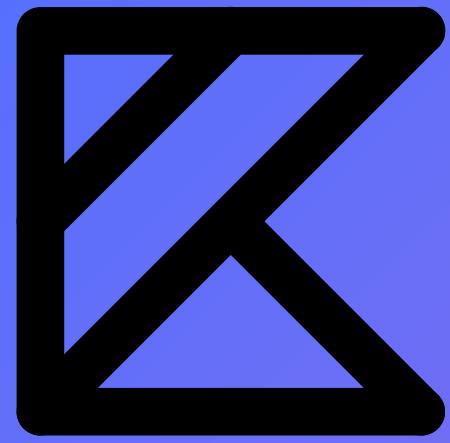
FILTER



FILTER

- Viene a ser una especia de alternativa funcional del For y Foreach para presentar los datos que tengamos en una lista.
- Por otro lado, tiene la cualidad de que podemos pasarle una condición en específico para filtrar los datos que está leyendo.
- Se podría incluso visualizar como una combinación de un if con un for.
- Un punto importante es que las diferentes opciones de la programación funcional, se pueden unir para generar resultados diferentes entre ellas.





MAP

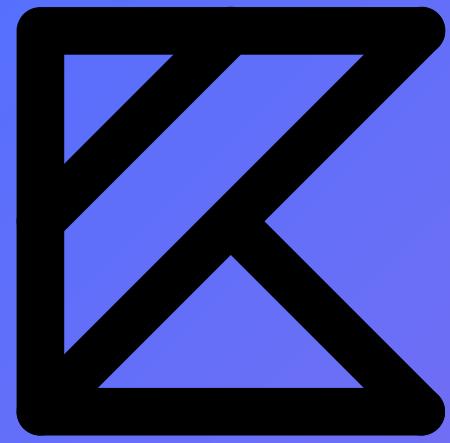


MAP

- Nos sirve para transformar nuestra lista de datos en nuevas series de datos, con otro orden u otro sentido según lo requerido.
- Podemos agregar una condición para manipular los elementos como nosotros queramos, ya no se trata de simplemente “mostrar” los datos, si no que de mostrarlos como nosotros decidimos hacerlo.

```
val listaPersonas = personas.map { it.nombre }
```





REDUCE



REDUCE

- A pesar de trabajar sobre una lista o gran cantidad de datos, reduce se suele condensar en un solo resultado.
- Puede ser incluso una combinación de valores que al final convergen todos en una sola cadena de texto.





**GRACIAS POR
LA ATENCIÓN**