ESQUEMA DE USO DE UNA FUNCIÓN DE ORDEN SUPERIOR Y UNA FUNCIÓN LAMBDA

// Definición de la función de orden superior.

fun fnOrdenSuperior (**param1**:Tipo1 , **param2**:Tipo2, ***fnLambda: (pm1: Tipo1) → Tipo2***){

```
    //Hago cosas con param1 y param2
     var b:Tipo2= //Proceso algo y lo recoge b

    //Aquí se invoca a la función Lambda que acepta un parámetro de entrada y devuelve otro

    var a:Tipo2 = fnLambda(b)

    //Puedo seguir haciendo cosas. En este caso no se devuelve nada. Si param1 o param2
    // hubiesen sido pasadas como referencias, entonces los cambios producidos dentro de la
    // función habrían afectado al dato en el contexto original allí donde esté

}
```

// **Uso de fnOrdenSuperior al mismo tiempo que se inyecta el cuerpo de la fnLambda()**

fun main(){

```
    //se hacen cosas

    //Se hace uso de fnOrdenSuperior. Se  introducen los parámetros que necesita y además
    //se inyecta el cuerpo de la funcion lambda que se ejecutará cuando fnOrdenSuperior la
    //invoque

    fnOrdenSuperior(x,y)
    { a->
        // Cuerpo de la función lambda. Hace cosas y devuelve algo

         val z = // algún procesamiento con 'a' y devuelve algo (z), sin Return

        z // El último valor se devuelve automáticamente
    }

    // Se siguen haciendo cosas
}
```

En resumen, una función de orden superior puede entenderse como una función cuyo comportamiento se completa en dos fases: 1) cuando se define la función y 2) cuando se llama y se le inyecta la función lambda, que se ejecutará dentro de la función de orden superior. Esto tiene como ventaja que la función lambda puede trabajar con parámetros de los dos contextos.