Obviamente podem existir **Arrays** de **Objetos que facilita iterações e mapeamentos.** É bem parecido com **Java na questão de encadear operações em cima de uma coleção** *(parece Streams)*:

```
val arrayBigDecimal: Array<BigDecimal> = popularArray( ...valores: "16000", "1300","5670","26800")
```

```
fun popularArray(vararg valores: String): Array<BigDecimal> {
    return Array<BigDecimal>(valores.size) {i -> valores[i].toBigDecimal()}
}
```

```
val aumento = BigDecimal( val: "1.1")
val salariosAumentados = arrayBigDecimal.map { salario ->
    if(salario < "5000".toBigDecimal()){
        salario.plus("500".toBigDecimal())  ^map
    }else{
        salario.multiply(aumento).setScale( newScale: 2,RoundingMode.HALF_UP)  ^map
    }
}.toTypedArray()
```

Operações de ***Redução (somatória total de Big Decimals)*** e ***ordenação*** também são simples. Tem o **take/takeLast** para pegar elementos da lista **de acordo com a quantidade desejada:**

```
val totalSalarios = salariosAumentados.reduce{ acumulador, salario -> acumulador + salario}

salariosAumentados.sorted().takeLast( n: 3)
```

A forma mais comum de se trabalhar com coleção de Objetos não é através da estrutura de **Array** e sim de **Lista.** Em **Kotlin** listas são sempre de ***Objetos*****,** nunca de tipos primitivos, elas são declaradas como ***mutableList (listas mutáveis):***

```
val livro = Livro( titulo: "Moacir Rocha",  autor: "Moacir", LocalDate.of( year: 1984, month: 1, dayOfMonth: 14))
val livro2 = Livro( titulo: "Amor e ódio",  autor: "Moacir", LocalDate.of( year: 2000, month: 12, dayOfMonth: 1))
val livro3 = Livro( titulo: "Clean Code",  autor: "Bob Martin", LocalDate.of( year: 1999, month: 9, dayOfMonth: 12))
val livro4 = Livro( titulo: "Moana",  autor: "Laurael", LocalDate.of( year: 2012, month: 3, dayOfMonth: 1))

val listaDeLivros = mutableListOf(livro, livro2,livro3, livro4)
```

Listas são mais simples de se trabalhar do que arrays, primeiro porque elas possuem valores dinâmico (quando mutáveis) e uma série de outros métodos que facilitam a iteração e alteração de valores como ***add(), remove()*** etc...

Métodos de ***ordenação, filtragem*** e ***mapeamento*** ainda se mantém muito semelhante ao que era nos **Arrays:**

```
val listaOrdenada = listaDeLivros.sortedBy { livro -> livro.titulo }

val listaDeTitulosDoAutor = listaDeLivros.filter { it.autor.contains( other: "Moa") } List<Livro>
    .map { it.titulo } List<String>
    .toList()
```

Em **operações de agrupamento que podem ter valores nullos**, podemos **definir um valor default com o elvis operator*(parece o operador ternário)***. A função de ***agrupamento/group by*** cria um mapa com chave e valor, a **chave nesse caso é a editora** e o **valor os livros que pertencem a ela**:

```
listaDeLivros.groupBy { it.editora ?: "Desconhecida "}
    .forEach { editora, livros ->
    println("Editora: $editora ,  Livros: $livros ")
}
```

```
MainKt
"C:\Program Files\Java\jdk-15\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.3.1
Editora: Desconhecida  ,  Livros: [Titulo: Biografia de Moacir Rocha  - Autor: Moacir, Titulo: Moana - Autor: Laurael]
Editora: Abril ,  Livros: [Titulo: Amor e ódio - Autor: Moacir, Titulo: Clean Code - Autor: Bob Martin]
```

Lista mutável e imutável

As **mutableList** do **Kotlin** sempre manipulam elas mesmas, ou seja, elas apontam para o mesmo endereço de objeto**. Listas imutáveis** *(são declaradas apenas como List)* **retornam outra lista com a alteração feita.**

Listas imutáveis estão mais para *Streams do Java*, elas sempre retornam **listas novas** com as modificações feitas enquanto **listas mutáveis** alteram a **elas próprias**.