



# Taller Estructuras de Datos en Kotlin

**Nombre de la formación:** ADSI                      **Ficha:** 2469285                      **Nombre Aprendiz:** Maite Salome Martínez Restrepo

El objetivo de este taller es que los aprendices sean capaces de comprender y utilizar las principales estructuras de datos en Kotlin, incluyendo arreglos, listas, conjuntos, mapas y pares.

El aprendiz deberá realizar un informe donde se evidencien los siguientes puntos:

1. **Introducción a las estructuras de datos en Kotlin**

a. **¿Qué son las estructuras de datos y para qué se utilizan?**

Son conjunto de datos integrados, como también se podría definir como una colección de datos, son tipos de colección, son una forma de representar información, que permiten almacenar varios valores, generalmente del mismo tipo de datos, de manera organizada, se utilizan para almacenar y manipular datos de manera eficiente, es decir, estos proporcionan una forma de organizar y gestionar los datos de manera que hace que su acceso, inserción, eliminación y búsqueda sea más rápida y eficiente, entonces son útiles para implementar algoritmos y diseñar la solución correcta para un determinado problema.

b. **Ventajas de utilizar estructuras de datos en Kotlin**

La ventajas de utilizarlos está en que, una colección puede ser una lista ordenada, una agrupación de valores únicos o una asignación de valores de un tipo de datos a valores de otro tipos, es decir la capacidad de usar colecciones de forma eficaz tiene la ventaja de permitir implementar funciones comunes, como desplazar listas y solucionar una variedad de problemas que involucran cantidades arbitrarias de datos, no solamente eso, también que con las estructuras de datos, el tiempo de acceso a los datos es muy pequeño porque accedemos a la memoria principal y se tarda lo mismo en acceder a cualquier dato de la estructura, a parte que nos ayudan a resolver un problema de manera mas sencilla gracias a que las reglas que las rigen nunca cambian y son dinámicas.

c. **Diferencias entre las estructuras de datos en Kotlin y Java**

KOTLIN	JAVA
Permiten almacenar varios valores, generalmente del mismo tipo de datos, de manera organizada.	Existen muchas formas de organizar los datos en memoria. Para estructurar los datos en la memoria, se emplean una serie de algoritmos que se conocen como datos abstractos. Estos tipos de datos abstractos son el conjunto de reglas que hacen que los algoritmos operen y funcionen en la estructura de datos en Java.
Pueden ser una lista ordenada, una agrupación de valores únicos o una asignación de valores de un tipo de dato a valores de otro tipo.	Los algoritmos en Java permiten recuperar y mostrar información rápidamente. Se trata de una parte fundamental dentro de la estructura de datos. Un array es una colección de elementos de memoria en el que los datos se almacenan de forma secuencial.
Son inmutables, lo que hace que sean mas comunes y se recomiendan para mejorar la seguridad.	No tiene una construcción específica para estructuras de datos inmutables y deben implementar manualmente.
Proporciona una estructura de datos llamada “rango”, es decir una secuencia de valores que están en un rango específico que se utilizan a menudo en iteraciones de bucles.	No proporciona la estructura de datos llamada “rango”, es decir, no existe.
Permite agregar funciones de extensión a estructuras de datos existentes, lo que permite extender la funcionalidad de las estructuras de datos sin tener que cambiar su código fuente.	No permite agregar funciones de extensión a estructuras de datos existentes, ya que no cuenta con esta capacidad.
Todos los tipos de datos son por defecto no nulos, lo que significa que no se pueden asignar valores nulos a menos que se especifique lo contrario. Se hace mediante el operador “?”, que indica que un valor puede ser nulo.	Los tipos de datos pueden ser nulos por defecto.

## 2. Arreglos en Kotlin

### a. ¿Qué es un arreglo?

Es una estructura de datos que se utiliza para almacenar una colección ordenada de elementos del mismo tipo. Los arreglos digamos que se definen utilizando la clase **Array** tiene una serie de funciones y propiedades para trabajar con los elementos del arreglo, algo curioso es que, los arreglos en Kotlin son de longitud fija, eso quiere decir que una vez que se crea un arreglo, no se pueden agregar o eliminar elementos.

### b. Creación de arreglos en Kotlin

```
main.kt
1 fun main(){
2     //Creación de arreglos en Kotlin
3
4     //Con String
5     val Empresarios = arrayOf("Karla Cepeda","Mariana Muñoz","Juliana Ramos","Mercedes Pinilla","Carlos Guerra")
6
7
8
9     //Con números
10    val numeros = arrayOf(1, 2, 3, 4, 5, 6, 7, 8)
11
12 }
```

### c. Accediendo a los elementos de un arreglo

Para acceder a los elementos de un arreglo, se utiliza la notación de corchetes y se especifica el índice del elemento que se desea acceder.

```
main.kt > ...
1 fun main(){
2     //Creación de arreglos en Kotlin
3
4     //Con String
5     val Empresarios = arrayOf("Karla Cepeda","Mariana Muñoz","Juliana Ramos","Mercedes Pinilla","Carlos Guerra")
6
7     //Acceder a los elementos de un arreglo
8     val elemento = Empresarios[3] //Devuelve el nombre Mercedes Pinilla
9     if(elemento == "Mercedes Pinilla"){
10        println("El nombre del empresario es: $elemento")
11    }else{
12        println("El empresario no es Mercedes Pinilla")
13    }
14
15
16     //Con números
17     val numeros = arrayOf(1, 2, 3, 4, 5, 6, 7, 8)
18
19     //Acceder a los elementos de un arreglo
20     val elementos = numeros[0] //Devuelve un 1
21     if(elementos == 1){
22        println("El número es: $elementos")
23    }else{
24        println("El número no es 1")
25    }
26 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El nombre del empresario es: Mercedes Pinilla
El número es: 1
> []
```

### d. Modificando los elementos de un arreglo

Los arreglos mutables son similares a los arreglos normales, pero se pueden modificar después de su creación. Para crear un arreglo mutable, se utiliza la clase `mutableList`, que tiene una serie de funciones y propiedades para agregar, eliminar y actualizar elementos. Los arreglos mutables en Kotlin se pueden modificar utilizando varias funciones, como `add`, `remove`, `set`, entre otras.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de arreglos mutables en Kotlin
3
4     //Con String
5     val Empresarios = mutableListOf("Karla Cepeda","Mariana Muñoz","Juliana
    Ramos","Mercedes Pinilla","Carlos Guerra")
6
7     //Modificar arreglos
8     Empresarios.add("Camila Escobar") //Agrega un nuevo elemento
9
10    //Con números
11    val numeros = mutableListOf(1, 2, 3, 4, 5, 6, 7, 8)
12
13    //Modificar arreglos
14    numeros.add(10) //Agrega un nuevo elemento
15    numeros.remove(2) //Elimina el elemento
16    numeros[1] = 42 //Modifica el elemento
17    val numeroAdd = numeros.plus(12) //Agrega un nuevo elemento
18    numeros.set(6, 5) //Cambia el valor del elemento
19    println(numeros)
20    println(numeroAdd)
21 }
~~~

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[1, 42, 4, 5, 6, 7, 5, 10]
[1, 42, 4, 5, 6, 7, 8, 10, 12]
> []
```

### Ejemplo con la función slice:

Se encarga de crear un nuevo arreglo que contiene sólo los elementos del arreglo original que se especifican.

```
main.kt > ...
1 fun main(){
2     //Creación de arreglos mutables en Kotlin
3
4     //Con String
5     val Empresarios = mutableListOf("Karla Cepeda","Mariana Muñoz","Juliana
    Ramos","Mercedes Pinilla","Carlos Guerra")
6
7     //Modificar arreglos
8     Empresarios.add("Camila Escobar") //Agrega un nuevo elemento
9
10    //Con números
11    val numeros = mutableListOf(1, 2, 3, 4, 5, 6, 7, 8)
12
13    //Modificar arreglos
14    val numeroDelete = numeros.slice(0 until 1) + numeros.slice(2 until numeros.size)
    //elimina elementos de un arreglo utilizando la función slice
15    println(numeroDelete)
16 }

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[1, 3, 4, 5, 6, 7, 8]
> []
```

### e. Recorriendo un arreglo

Se puede recorrer cada elemento del arreglo utilizando el bucle for, el cual se encarga de recorrer los elementos del arreglo uno por uno.

### Ejemplo:

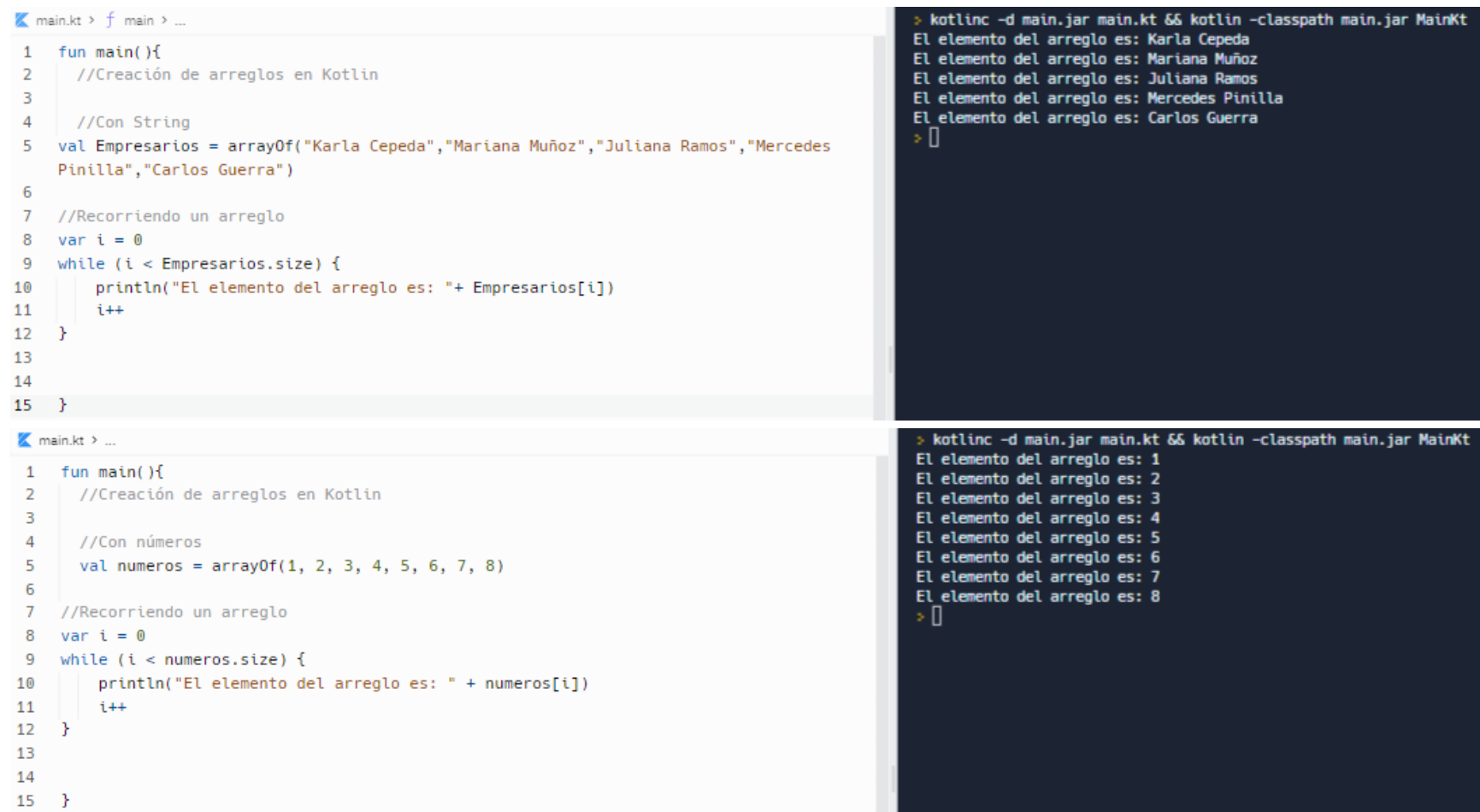
```
main.kt > f main > ...
1 fun main(){
2     //Creación de arreglos en Kotlin
3
4     //Con String
5     val Empresarios = arrayOf("Karla Cepeda","Mariana Muñoz","Juliana Ramos","Mercedes Pinilla","Carlos
    Guerra")
6
7     //Recorriendo un arreglo
8     for (Empresario in Empresarios) {
9         println("El nombre del empresario ingresado es: $Empresario")
10    }
11
12    //Con números
13    val numeros = arrayOf(1, 2, 3, 4, 5, 6, 7, 8)
14
15    //Recorriendo un arreglo
16    for (numero in numeros) {
17        println("El número ingresado es: $numero")
18    }
19 }
20 }

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El nombre del empresario ingresado es: Karla Cepeda
El nombre del empresario ingresado es: Mariana Muñoz
El nombre del empresario ingresado es: Juliana Ramos
El nombre del empresario ingresado es: Mercedes Pinilla
El nombre del empresario ingresado es: Carlos Guerra
El número ingresado es: 1
El número ingresado es: 2
El número ingresado es: 3
El número ingresado es: 4
El número ingresado es: 5
El número ingresado es: 6
El número ingresado es: 7
El número ingresado es: 8
> []
```

También se puede con la función forEach y bucle while

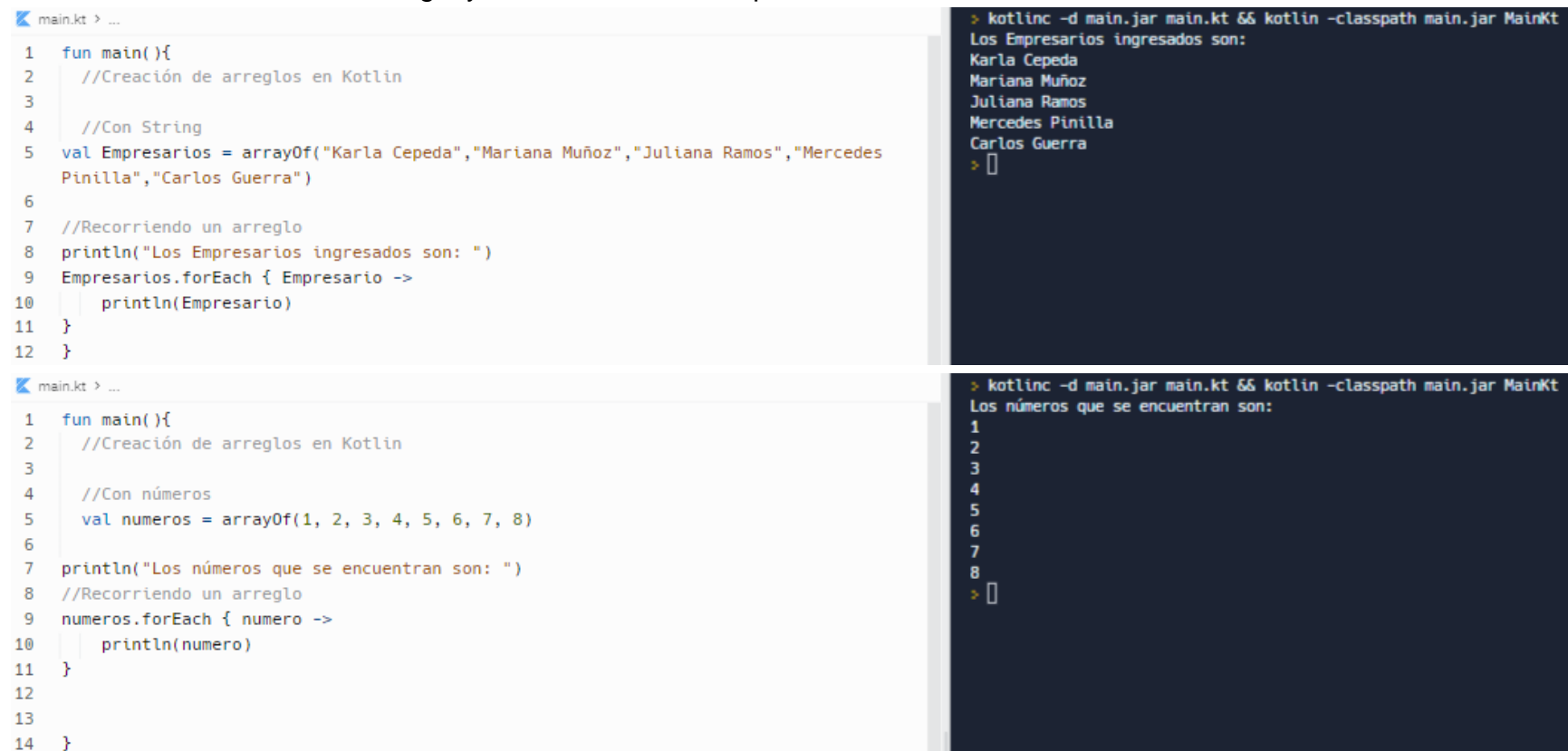
### Ejemplo con bucle while:

En este caso, se necesita un índice de control para recorrer los elementos del arreglo uno por uno.



### Ejemplo con el `forEach`:

Recorre cada elemento del arreglo y llama a una función para cada uno.

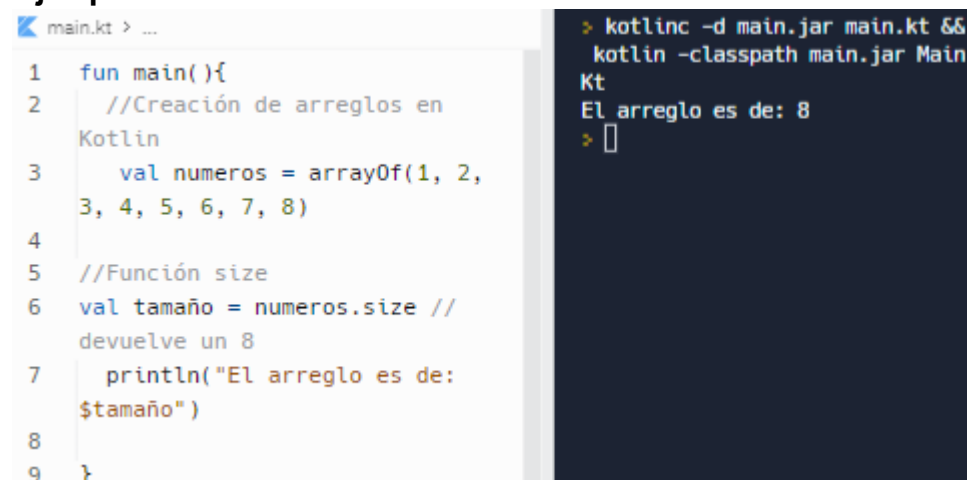


### f. Funciones útiles para trabajar con arreglos en Kotlin

Kotlin proporciona una amplia variedad de funciones y métodos útiles para trabajar con arreglos, tenemos algunos como, **por ejemplo**:

- **size**: Devuelve el número de elementos en el arreglo.

#### Ejemplo:



- **get(index)**: Devuelve el valor del elemento en el índice especificado.

#### Ejemplo:

```
main.kt > f main > ...  
1 fun main(){  
2     //Creación de arreglos en  
   Kotlin  
3     val numeros = arrayOf(1, 2,  
   3, 4, 5, 6, 7, 8)  
4  
5     //Función get(index)  
6     val posicionNum = numeros.get(4)  
   //Devuelve un 5  
7     println("El número es:  
   $posicionNum")  
8 }
```

```
> kotlinc -d main.jar main.kt &&  
kotlin -classpath main.jar Main  
Kt  
El número es: 5  
> []
```

- **forEach:** Ejecuta una función para cada elemento del arreglo.

#### Ejemplo:

```
main.kt  
1 fun main(){  
2     //Creación de arreglos en  
   Kotlin  
3     val numeros = intArrayOf(1,  
   2, 3, 4, 5, 6, 7, 8)  
4  
5     println("Los números del  
   arreglo son: ")  
6     //Función forEach  
7     numeros.forEach { numero ->  
   println(numero)  
8  
9  
10 }  
11  
12 }
```

```
> kotlinc -d main.jar main.kt &&  
kotlin -classpath main.jar Main  
Kt  
Los números del arreglo son:  
1  
2  
3  
4  
5  
6  
7  
8  
> []
```

### 3. Listas en Kotlin

#### a. ¿Qué es una lista?

Es una colección de elementos con un orden específico, es como decir, es una colección genérica de elemento que se caracteriza por almacenarlos de forma ordenada, donde pueden existir duplicados e internamente se comienzan a contar (índice) desde 0, donde se almacenan en un orden secuencial y se puede acceder por su índice y son una estructura de datos que se utiliza para almacenar y manipular datos de manera eficiente.

#### b. Creación de listas en Kotlin

Existen dos tipos de listas en Kotlin: las inmutables y las mutables.

Las listas inmutables se crean con la función **listOf** que crea una lista de solo lectura, la cual no se puede modificar después de su creación. Algo curioso de esto es que los elementos de la lista se pasan como argumentos, separados por comas o se pueden pasar como una colección, es decir, una forma de enumeración en el código que Kotlin proporciona como un espacio en donde se almacenan elementos.

#### Ejemplo:

```
1 fun main() {  
2     //Creación de listas inmutables  
3     //Con numeros  
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10)  
5  
6     //Con String  
7     val nombres = listOf("Martha", "Paula",  
   "Nicolás", "Andrés", "Valentina")  
8  
}
```

Las listas mutables se crean con la función **mutableList** que crea una lista que si se puede modificar después de su creación y los elementos de la lista se pueden agregar o eliminar utilizando métodos.

#### Ejemplo:



```


1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás", "Andrés",
5                                   "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
6
7     //Con números
8     val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
9
10 }

```

### c. Accediendo a los elementos de una lista

Para acceder a los elementos de la lista, se puede utilizar la sintaxis de corchetes y pasar el índice del elemento deseado.

#### Ejemplo en listas inmutables:



```

1 fun main() {
2     //Creación de listas inmutables
3     //Con numeros
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
5
6     //Accediendo a los elementos de una lista
7     val NumeroLista = numeros[3]
8     println("En el puesto 3 de la lista tenemos el número: $NumeroLista")
9
10    //Con String
11    val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés", "Valentina")
12
13    //Accediendo a los elementos de una lista
14    val NombreLista = nombres[2]
15    println("En el puesto 2 de la lista tenemos a: $NombreLista")
16
17 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
En el puesto 3 de la lista tenemos el número: 4
En el puesto 2 de la lista tenemos a: Nicolás
> 

```

#### Ejemplo en listas mutables:



```

1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás", "Andrés", "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
5
6     //Accediendo a los elementos de una lista
7     val NombreLista = nombres[7]
8     println("En el puesto 7 de la lista tenemos a: $NombreLista")
9
10    //Con números
11    val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
12
13    //Accediendo a los elementos de una lista
14    val NumeroLista = numeros[5]
15    println("En el puesto 5 de la lista tenemos el número: $NumeroLista")
16 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
En el puesto 7 de la lista tenemos a: Luffy
En el puesto 5 de la lista tenemos el número: 6
> 

```

### d. Modificando los elementos de una lista

Los elementos de la lista se pueden agregar o eliminar utilizando los métodos de la lista, teniendo en cuenta lo dicho, para modificar los elementos de una lista en Kotlin, la lista debe ser mutable, ya que estas listas son las que se pueden modificar agregando, eliminando o actualizando elementos.



```

1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás", "Andrés", "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
5
6     //Modificar los elementos de una lista
7     nombres.add("Zoro") //Agrega un nuevo elemento
8     nombres.removeAt(7) //Elimina la posición indicada de la lista (recordando que en la lista la posición inicial es 0)
9     nombres[2] = "Valeria" // Actualiza el valor del elemento que se encontraba en esa posición de la lista
10    println(nombres)
11
12    //Con números
13    val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
14
15    //Modificar los elementos de una lista
16    numeros.add(11) //Agrega un nuevo elemento a la lista
17    numeros.removeAt(0) //Elimina la posición indicada de la lista (recordando que en la lista la posición inicial es 0)
18    numeros[0] = 1 // Actualiza el valor del elemento que se encontraba en esa posición de la lista
19    println(numeros)
20
21 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[Martha, Paula, Valeria, Andrés, Valentina, Rocío, Carmela, Nami, Ussop, Zoro]
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11]
> 

```

Al intentar aplicar estos métodos a una lista inmutable, pasará lo siguiente:

```
main.kt x +
main.kt > f main > ...
1 fun main() {
2     //Creación de listas inmutables
3     //Con numeros
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
5
6     //Modificar los elementos de una lista
7     numeros.add(2) //Agrega un nuevo elemento
8     numeros.removeAt(7) //Elimina la posición indicada de la lista (recordando que en la
9     //lista la posición inicial es 0)
10    numeros[2] = 5 // Actualiza el valor del elemento que se encontraba en esa posición de la
11    //lista
12    println(numeros)
13 }
14
15
```

```
>_ Console x Shell x +
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
main.kt:7:9: error: unresolved reference: add
numeros.add(2) //Agrega un nuevo elemento
      ^
main.kt:8:9: error: unresolved reference: removeAt
numeros.removeAt(7) //Elimina la posición indicada de la lista (recordando que en la lista la posición inicial es 0)
      ^
main.kt:9:1: error: unresolved reference. None of the following candidates is applicable because of receiver type mismatch :
public inline operator fun kotlin.text.StringBuilder /* = java.lang.StringBuilder */.set(index: Int, value: Char): Unit defined in kotlin.text
numeros[2] = 5 // Actualiza el valor del elemento que se encontraba en esa posición de la lista
^
main.kt:9:8: error: no set method providing array access
numeros[2] = 5 // Actualiza el valor del elemento que se encontraba en esa posición de la lista
      ^
exit status 1
> []
```

Como pueden ver, saldrá un error, ya que las listas inmutables son solo de lectura que no se puede modificar después de su creación.

También se puede modificar los elementos utilizando un bucle for y la sintaxis de corchetes [] dentro del cuerpo del bucle, **ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de listas mutables
3     val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
4
5     //Modificar los elementos de una lista
6
7     //Utilizando un bucle for
8     for (i in numeros.indices) {
9         numeros[i] = numeros[i] * 4 // Multiplica cada elemento por 4
10    }
11    println(numeros)
12 }
13 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
MainKt
[4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
> []
```

Como pueden ver el bucle altera el valor de los elementos que habíamos colocado, siendo reemplazados por unos nuevos, que van en 4 en 4, pero si se fijan, guarda el numero de posiciones que teníamos al crear la lista.

## e. Recorriendo una lista

Con el bucle for se puede recorrer todos los elementos de una lista.

### Ejemplo en listas inmutables:

```
main.kt > ...
1 fun main() {
2     //Creación de listas inmutables
3     //Con numeros
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10,"\\n")
5
6     //Recorrer una lista
7     for (numero in numeros) {
8         println(numero)
9     }
10
11    //Con String
12    val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés", "Valentina")
13
14    //Recorrer una lista
15    for (nombre in nombres) {
16        println(nombre)
17    }
18 }
19 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
1
2
3
4
5
6
7
8
9
10

Martha
Paula
Nicolás
Andrés
Valentina
> []
```

### Ejemplo en listas mutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás", "Andrés",
5     "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop\\n")
6
7     //Recorrer una lista
8     for (nombre in nombres) {
9         println(nombre)
10    }
11    //Con números
12    val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
13
14    //Recorrer una lista
15    for (numero in numeros) {
16        println(numero)
17    }
18 }
19 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
Martha
Paula
Nicolás
Andrés
Valentina
Rocío
Carmela
Luffy
Nami
Ussop

1
2
3
4
5
6
7
8
9
10
> []
```

## f. Funciones útiles para trabajar con listas en Kotlin

Kotlin da la oportunidad de manejar diversas funciones que son muy buenas para trabajar con listas, en ellas tenemos las siguientes:

- **size:** Devuelve el tamaño de una lista.

**Ejemplo con listas inmutables:**

main.kt > f main > ...

```
1 fun main(){
2     //Creación de listas inmutables
3     //Con números
4
5     //Función size
6     val numeros = listOf(1,2,3,4, 5,6,7,8,9,10)
7     val tamañoLista = numeros.size // Devuelve el tamaño de la lista,
    según el número de elementos que se encuentren
8     println("El tamaño de esta lista es de: $tamañoLista")
9
10    //Con String
11    val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
    "Valentina")
12
13    //Función size
14    val tamañoDeLista = nombres.size // Devuelve el tamaño de la lista,
    según el número de elementos que se encuentren
15    println("El tamaño de esta lista es de: $tamañoDeLista")
16 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El tamaño de esta lista es de: 10
El tamaño de esta lista es de: 5
> []
```

**Ejemplo con listas mutables:**

main.kt > ...

```
1 fun main(){
2     //Creación de listas mutables
3     //Con números
4
5     //Función size
6     val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
7     val tamañoLista = numeros.size // Devuelve el tamaño de la lista,
    según el número de elementos que se encuentren
8     println("El tamaño de esta lista es de: $tamañoLista")
9
10    //Con String
11    val nombres = mutableListOf("Martha", "Paula", "Nicolás", "Andrés",
    "Valentina","Rocío","Carmela","Luffy","Nami","Ussop")
12
13    //Función size
14    val tamañoDeLista = nombres.size // Devuelve el tamaño de la lista,
    según el número de elementos que se encuentren
15    println("El tamaño de esta lista es de: $tamañoDeLista")
16 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El tamaño de esta lista es de: 10
El tamaño de esta lista es de: 10
> []
```

- **contains:** Devuelve un tipo de dato booleano como lo es true si la lista contiene el elemento especificado.

**Ejemplo con listas inmutables:**

main.kt > ...

```
1 fun main(){
2     //Creación de listas inmutables
3     //Con numeros
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
5
6     //Función contains
7     val elementos = numeros.contains(5) // Devuelve true
8     println("Esta lista contiene un 5? $elementos")
9
10    //Con String
11    val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
    "Valentina")
12
13    //Función contains
14    val elemento = nombres.contains("Karen") // Devuelve false
15    println("En esta lista se encuentra la Señora Karen? $elemento")
16 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
Esta lista contiene un 5? true
En esta lista se encuentra la Señora Karen? false
> []
```

**Ejemplo con listas mutables:**



```
main.kt > ...
1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás",
5     "Andrés", "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
6
7     //Función contains
8     val elementos = nombres.contains("Luffy") // Devuelve true
9     println("En esta lista se encuentra el Señor Luffy? $elementos")
10
11     //Con números
12     val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
13
14     //Función contains
15     val elemento = numeros.contains(23) // Devuelve false
16     println("En esta lista se encuentra el numero 23? $elemento")
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
En esta lista se encuentra el Señor Luffy? true
En esta lista se encuentra el numero 23? false
```

- **indexOf:** Devuelve el índice de la primera ocurrencia (o palabra) del elemento especificado en la lista. Si el elemento no está en ella, devuelve -1.

### Ejemplo con listas mutables:

```
main.kt > ...
1 fun main() {
2     //Creación de listas mutables
3     //Con numeros
4     val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
5
6     //Función indexOf
7     val elementos = numeros.indexOf(12) // Devuelve -1 (porque no
8     //está)
9     println(elementos)
10
11     //Con String
12     val nombres = mutableListOf("Martha", "Paula", "Nicolás",
13     "Andrés", "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
14
15     //Función indexOf
16     val elemento = nombres.indexOf("Luffy") // Devuelve un 7
17     println(elemento)
18 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
-1
7
```

Devuelve un 7 ya que, es la posición donde se encuentra el elemento en la lista.

### Ejemplo con listas inmutables:

```
main.kt > ...
1 fun main() {
2     //Creación de listas inmutables
3     //Con numeros
4     val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
5
6     //Función indexOf
7     val elementos = numeros.indexOf(3) // Devuelve 2
8     println(elementos)
9
10
11     //Con String
12     val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
13     "Valentina")
14
15     //Función indexOf
16     val elemento = nombres.indexOf("Marina") // Devuelve un -1
17     // (porque no está)
18     println(elemento)
19 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
2
-1
```

- **lastIndexOf:** Devuelve el índice de la última ocurrencia (o palabra) del elemento especificado en la lista. Si el elemento no está en la lista, devuelve -1.

### Ejemplo con listas mutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás",
5     "Andrés", "Valentina","Rocío","Carmela","Luffy","Nami","Ussop")
6
7     //Función lastIndexOf
8     val elementos = nombres.lastIndexOf("Nami") // Devuelve un 8
9     println(elementos)
10
11    //Con números
12    val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
13
14    //Función lastIndexOf
15    val elemento = numeros.lastIndexOf(45) // Devuelve un -1
16    println(elemento)
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
8
-1
[]
```

Devuelve un -1 ya que el 45 no se encuentra en la lista.

### Ejemplo con listas inmutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas inmutables
3     //Con String
4     val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
5     "Valentina")
6
7     //Función lastIndexOf
8     val elementos = nombres.lastIndexOf("Pepe") // Devuelve un -1
9     (porque no se encuentra en la lista)
10    println(elementos)
11
12    //Con números
13    val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
14
15    //Función lastIndexOf
16    val elemento = numeros.lastIndexOf(10) // Devuelve un 9
17    println(elemento)
18 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
-1
9
[]
```

Devuelve un 9, ya que antes del 10, viene ese elemento, según la lista.

- **subList:** Devuelve una sublista de la lista original que incluye elementos desde el índice de inicio especificado hasta el índice final especificado.

### Ejemplo con listas mutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás",
5     "Andrés", "Valentina","Rocío","Carmela","Luffy","Nami","Ussop")
6
7     //Función subList
8     val elementos = nombres.subList(1, 7) // Devuelve una lista que
9     incluye [1,2,3,4,5,6]
10    println(elementos)
11
12    //Con números
13    val numeros = mutableListOf(1,2,3,4,5,6,7,8,9,10)
14
15    //Función subList
16    val elemento = numeros.subList(5,8) // Devuelve una lista que
17    incluye [5,6,7]
18    println(elemento)
19 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
[Paula, Nicolás, Andrés, Valentina, Rocío, Carmela]
[6, 7, 8]
[]
```

### Ejemplo con listas inmutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas inmutables
3     //Con String
4     val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
5 "Valentina")
6     //Función subList
7 val elementos = nombres.subList(0, 3) // Devuelve una lista que
8 incluye [0,1,2]
9 println(elementos)
10
11 //Con números
12 val numeros = listOf(1,2,3,4,5,6,7,8,9,10)
13 //Función subList
14 val elemento = numeros.subList(1,6) // Devuelve una lista que
15 incluye [1,2,3,4,5]
16 println(elemento)
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[Martha, Paula, Nicolás]
[2, 3, 4, 5, 6]
```

- **sorted:** Devuelve una lista de manera ordenada de los elementos de la lista original. El orden se determina por el orden natural de los elementos, a menos que se especifique un comparador personalizado.

### Ejemplo con listas mutables:

```
main.kt > f main > ...
1 fun main(){
2     //Creación de listas mutables
3     //Con String
4     val nombres = mutableListOf("Martha", "Paula", "Nicolás",
5 "Andrés", "Valentina", "Rocío", "Carmela", "Luffy", "Nami", "Ussop")
6     //Función sorted
7 val elementos = nombres.sorted() // Devuelve una lista de
8 manera ordenada (en forma alfabetica)
9 println(elementos)
10
11 //Con numeros
12 val numeros = mutableListOf(5,7,1,9,10,56,20,560,6,16)
13 //Función sorted
14 val elemento = numeros.sorted() // Devuelve una lista de
15 manera ordenada (menor a mayor)
16 println(elemento)
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[Andrés, Carmela, Luffy, Martha, Nami, Nicolás, Paula, Rocío,
Ussop, Valentina]
[1, 5, 6, 7, 9, 10, 16, 20, 56, 560]
```

### Ejemplo con listas inmutables:

```
main.kt > ...
1 fun main(){
2     //Creación de listas inmutables
3     //Con String
4     val nombres = listOf("Martha", "Paula", "Nicolás", "Andrés",
5 "Valentina")
6     //Función sorted
7 val elementos = nombres.sorted() // Devuelve una lista de manera
8 ordenada (en forma alfabetica)
9 println(elementos)
10
11 //Con números
12 val numeros = listOf(6,3,16,7,9,1,5,10)
13 //Función sorted
14 val elemento = numeros.sorted() // Devuelve una lista de manera
15 ordenada (menor a mayor)
16 println(elemento)
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[Andrés, Martha, Nicolás, Paula, Valentina]
[1, 3, 5, 6, 7, 9, 10, 16]
```

## 4. Conjuntos en Kotlin

### a. ¿Qué es un conjunto?

En Kotlin es una colección no ordenada de elementos que no permite elementos duplicados, es decir, los conjuntos se definen utilizando la clase set y pueden ser mutables o inmutables, pero si algo se sabe es que no tienen un orden específico, por lo

que lo elemento pueden aparecer en un orden diferente al que se agregaron y si se intenta agregar un elemento que ya está presente en el conjunto, no se agregará nuevamente, de allí que no permite elementos duplicados.

#### b. Creación de conjuntos en Kotlin

Para crear un conjunto mutable se utiliza la función `mutableSetOf()`.

**Ejemplo:**

```
1 fun main(){
2     //Conjuntos mutables
3
4     //Con string
5     val nombres =
        mutableSetOf("María",
            "Becky", "Carolina",
            "David", "Karolina")
6
7     //Con números
8     val numeros =
        mutableSetOf(1, 2, 3, 4, 5)
9
10 }
```

Para crear un conjunto inmutable se utiliza la función `setOf()`.

**Ejemplo:**

```
1 fun main(){
2     //Conjuntos inmutables
3
4     //Con String
5     val nombres =
        setOf("Aram", "Lola",
            "Carmen", "Lina", "Sara")
6
7     //Con números
8     val numeros = setOf(1, 2, 3,
        4, 5)
9
10 }
```

#### c. Accediendo a los elementos de un conjunto

Para acceder a los elementos de un conjunto utilizamos el método, o bueno, en Kotlin función llamada `contains()` para verificar si un elemento está presente en el conjunto.

**Ejemplo en conjuntos mutables:**

```
main.kt > f main > ...
1 fun main(){
2     //Conjuntos mutables
3     val numeros = mutableSetOf(1, 2, 3, 4, 5)
4
5     //Acceder a los elementos de un conjunto
6     if (numeros.contains(6)) {
7         println("El conjunto contiene el número 6")
8     } else {
9         println("El conjunto no contiene el número 6")
10    }
11 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
El conjunto no contiene el número 6
> []
```

**Ejemplo en conjuntos inmutables:**

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3
4     val nombres = setOf("Aram", "Lola", "Carmen", "Lina", "Sara")
5
6     //Acceder a los elementos de un conjunto
7
8     if (nombres.contains("Lina")) {
9         println("El conjunto contiene a la Señora Lina")
10    } else {
11        println("El conjunto no contiene a la Señora Lina")
12    }
13 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El conjunto contiene a la Señora Lina
> []
```

#### d. Modificando los elementos de un conjunto

Los conjuntos son colecciones inmutables, lo que significa que no se pueden modificar directamente los elementos que contienen. Si deseamos cambiar un elemento en un conjunto, debemos crear un nuevo conjunto con los elementos actualizados.

#### Ejemplo de conjuntos inmutables:

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3
4     val numeros = setOf(1, 2, 3, 4, 5)
5
6     //Modificar los elementos de un conjunto
7
8     // Crear un nuevo conjunto con los elementos multiplicados por dos
9     val numero = numeros.map { it * 2 }.toSet()
10
11     println(numero)
12
13 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[2, 4, 6, 8, 10]
> []
```

#### Ejemplo con conjuntos mutables:

Estos conjuntos son mutables y permiten agregar y eliminar elemento utilizando los métodos `add()` y `remove()`, se puede modificar el valor de un elemento en un conjunto mutable directamente, ya que los elementos en un conjunto como este, no tienen un índice, algo que hay que saber también es que, para modificar un elemento en un conjunto mutable, debemos eliminar el elemento existente y agregar el elemento actualizado.

```
main.kt > ...
1 fun main(){
2     //Conjuntos mutables
3
4     //Con string
5     val nombres = mutableSetOf("María", "Becky", "Carolina", "David", "Karolina")
6
7     //Modificar elementos de un conjunto
8     nombres.add("Enrique") //agregar elementos utilizando el método add()
9     nombres.remove("Becky") //Elimina un elemento
10
11     println(nombres)
12 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[María, Carolina, David, Karolina, Enrique]
> []
```

#### e. Recorriendo un conjunto

Para recorrer un conjunto utilizamos un ciclo `for` para ver todos los elementos del conjunto, es decir, para iterar sobre cada elemento del conjunto.

#### Ejemplo con conjuntos inmutables:



```
main.kt > f main > numeros
1 fun main(){
2     //Conjuntos inmutables
3
4     val numeros = setOf(1, 2, 3, 4, 5)
5
6     //Recorrer un conjunto
7     for (numero in numeros) {
8         println(numero)
9     }
10 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
1
2
3
4
5
> []
```

### Ejemplo con conjuntos mutables:

```
main.kt > ...
1 fun main(){
2     //Conjuntos mutables
3
4     val nombres = mutableSetOf("María", "Becky", "Carolina", "David",
5                               "Karolina")
6
7     //Recorrer un conjunto
8     for (nombre in nombres) {
9         println(nombre)
10    }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
María
Becky
Carolina
David
Karolina
> []
```

### f. Funciones útiles para trabajar con conjuntos en Kotlin

Kotlin nos brinda diversas funciones que son útiles para trabajar con conjuntos, las cuales son:

- **union:** Devuelve un conjunto que contiene todos los elementos de los dos conjuntos dados.

### Ejemplo con conjuntos inmutables:

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3
4     //Función union()
5     val CampoA = setOf(1, 2,
6                       3, 4, 5)
7     val CampoB = setOf(6, 7,
8                       8, 9, 10)
9     val unir =
10    CampoA.union(CampoB) //
    Devuelve la fusión entre
    ambas
    [1,2,3,4,5,6,7,8,9,10]
    println(unir)
}
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
> []
```

### Ejemplo con conjuntos mutables:

```
main.kt > ...
1 fun main(){
2     //Conjuntos mutables
3
4     //Función union()
5     val PersonalD = mutableSetOf("María", "Becky", "Carolina",
6                                  "David", "Karolina")
7     val PersonalF = mutableSetOf("Aram", "Lola", "Carmen", "Lina",
8                                  "Sara")
9     val unir = PersonalD.union(PersonalF) // Devuelve la fusión
    entre ambas
    ["María","Becky","Carolina","David","Karolina","Aram", "Lola",
    "Carmen", "Lina", "Sara"]
    println(unir)
10 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[María, Becky, Carolina, David, Karolina, Aram, Lola, Carmen, Lina, Sara]
> []
```

- **intersect:** Devuelve un conjunto que contiene solo los elementos que están presentes en ambos conjuntos dados.

### Ejemplo:

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3
4     //Función intersect()
5     val CampoA = setOf(1, 2, 3, 8, 5)
6     val CampoB = setOf(6, 7, 8, 9, 10)
7     val elementos = CampoA.intersect(CampoB) // Devuelve un 8 (ya que se encuentra
8     println(elementos)
9
10    //Conjuntos mutables
11
12    //Función intersect()
13    val PersonalD = mutableSetOf("María", "Becky", "Carolina", "David", "Karolina")
14    val PersonalF = mutableSetOf("Aram", "Lola", "Carmen", "Lina", "David")
15    val elemento = PersonalD.intersect(PersonalF) // Devuelve el nombre David (ya
16    println(elemento)
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[8]
[David]
> []
```

Cuando los conjuntos no comparten elementos en común la una con la otra, sucede esto:

```
main.kt > f main > ...
1 fun main(){
2
3     //Conjuntos mutables
4
5     //Función intersect()
6     val PersonalD = mutableSetOf("María", "Becky", "Carolina", "David", "Karolina")
7     val PersonalF = mutableSetOf("Aram", "Lola", "Carmen", "Lina", "Asaf")
8     val elemento = PersonalD.intersect(PersonalF) // No devuelve nada porque no
9     println(elemento)
10 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[]
> []
```

- **subtract:** Devuelve un conjunto que contiene los elementos del primer conjunto que no están presentes en el segundo conjunto.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3
4     //Función subtract()
5     val CampoA = setOf(1, 2, 3, 8, 5)
6     val CampoB = setOf(6, 7, 8, 9, 10)
7     val elementos = CampoA.subtract(CampoB) // Devuelve los primeros elementos
8     println(elementos)
9
10    //Conjuntos mutables
11
12    //Función subtract()
13    val PersonalD = mutableSetOf("María", "Becky", "Carolina", "David",
14    "Karolina")
15    val PersonalF = mutableSetOf("Aram", "Lola", "Carmen", "Lina", "David")
16    val elemento = PersonalD.subtract(PersonalF) // Devuelve los primeros
17    println(elemento)
18 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[1, 2, 3, 5]
[María, Becky, Carolina, Karolina]
> []
```

- **toMutableSet:** Convierte un conjunto inmutable en un conjunto mutable.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Conjuntos inmutables
3     val CampoA= setOf(1, 2, 3, 4, 5)
4     //Función toMutableSet()
5     val CampoB = CampoA.toMutableSet() //Convierte un conjunto
6     CampoB.add(6) //Agrega un elemento al conjunto
7
8     println(CampoB)
9 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[1, 2, 3, 4, 5, 6]
> []
```

- **contains:** Verifica si un conjunto contiene un elemento determinado.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2 //Conjuntos inmutables
3
4 //Función contains()
5 val CampoA = setOf(1, 2, 3, 4, 5)
6
7 if (CampoA.contains(2)) {
8     println("El conjunto contiene el número 2")
9 } else {
10     println("El conjunto no contiene el número 2")
11 }
12
13 //Conjuntos mutables
14
15 //Función contains()
16 val PersonalD = mutableSetOf("María", "Becky", "Carolina", "David", "Karolina")
17
18 if (PersonalD.contains("Mario")) {
19     println("El conjunto contiene al Señor Mario")
20 } else {
21     println("El conjunto no contiene al Señor Mario")
22 }
23 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El conjunto contiene el número 2
El conjunto no contiene al Señor Mario
> []
```

- **isEmpty:** Verifica si un conjunto está vacío.

### Ejemplo:

```
main.kt > ...
1 fun main(){
2 //Conjuntos inmutables
3
4 //Función isEmpty()
5 val CampoA = setOf(1, 2, 3, 8, 5)
6 if (CampoA.isEmpty()){
7     println("El conjunto está vacío")
8 } else {
9     println("El conjunto no está vacío")
10 }
11 //Conjuntos mutables
12
13 //Función isEmpty()
14 val PersonalF = mutableSetOf<String>()
15 if (PersonalF.isEmpty()){
16     println("El conjunto está vacío")
17 } else {
18     println("El conjunto no está vacío")
19 }
20 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El conjunto no está vacío
El conjunto está vacío
> []
```

## 5. Mapas en Kotlin

### a. ¿Qué es un mapa?

Es una estructura de datos, que se utiliza para almacenar pares, es decir clave-valor. Cada elemento en un mapa consta de una clave única y un valor asociado a esa clave. En los mapas nos encontramos con algunos ejemplos de tipos de datos comunes que se pueden utilizar como clave y valor son Int, String, Double, entre otros. Se puede inicializar utilizando la función `mapOf()`, que toma una lista de pares clave-valor y crea un mapa inmutable a partir de ella, algo de tener en cuenta es que los mapas pueden ser mutables e inmutables, cuando queremos un mapa mutable se coloca la clase `mutableMapOf()` y como anteriormente hemos dicho, estos mapas se pueden modificar, agregar, eliminar o actualizar pares clave-valor o en este caso elementos del mismo, un dato a tener en cuenta es que, las claves deben ser únicas, de lo contrario se producirá un error en el momento de compilación.

### b. Creación de mapas en Kotlin

#### Ejemplo de mapa inmutable:

```
main.kt > ...  
1 fun main(){  
2     //Creación de mapa inmutable  
3     val mapaC = mapOf(  
4         "Identificación" to 1092445018,  
5         "Nombre" to "Martha Muñoz",  
6         "Telefono" to 3214567473  
7     )  
8     println(mapaC)  
9  
10 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt  
{Identificación=1092445018, Nombre=Martha Muñoz, Telefono=3214567473}  
> []
```

Ejemplo de mapa mutable:

```
main.kt > f main > ...  
1 fun main(){  
2     //Creación de mapa mutable  
3     val mapaD = mutableMapOf(  
4         "Nombre" to "Karina Sanchez",  
5         "Formación" to "ADSI",  
6         "Numero de ficha" to 2469285  
7     )  
8     println(mapaD)  
9 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt  
{Nombre=Karina Sanchez, Formación=ADSI, Numero de ficha=2469285}  
> []
```

### c. Accediendo a los elementos de un mapa

Ejemplo:

```
main.kt > ...  
1 fun main(){  
2     //Creación de mapa inmutable  
3     val mapaC = mapOf(  
4         "Identificación" to 1092445018,  
5         "Nombre" to "Martha Muñoz",  
6         "Telefono" to 3214567473  
7     )  
8     // Acceder al valor asociado a una clave en un mapa  
9     val valorC = mapaC["Nombre"]  
10    println("El nombre del aprendiz es: $valorC") // Imprime el  
    nombre de la persona  
11  
12    //Creación de mapa mutable  
13    val mapaD = mutableMapOf(  
14        "Nombre" to "Karina Sanchez",  
15        "Formación" to "ADSI",  
16        "Número de ficha" to 2469285  
17    )  
18    // Acceder al valor asociado a una clave en un mapa  
19    val valorD = mapaD["Número de ficha"]  
20    println("El número de la ficha del aprendiz es: $valorD") //  
    Imprime el numero de la ficha  
21 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt  
El nombre del aprendiz es: Martha Muñoz  
El número de la ficha del aprendiz es: 2469285  
> []
```

### d. Modificando los elementos de un mapa

Ejemplo con mapa mutable:

```
main.kt > ...  
1 fun main(){  
2  
3     //Creación de mapa mutable  
4     val mapaD = mutableMapOf(  
5         "Nombre" to "Karina Sanchez",  
6         "Formación" to "ADSI",  
7         "Número de ficha" to 2469285,  
8         "Telefono" to 3240529529  
9     )  
10  
11    //Modificar elemento de un mapa  
12    mapaD["Nombre"] = "Karen Muñoz" //Modifica el valor del elemento del mapa  
13    mapaD["Trimeste"] = 5 //Agrega un nuevo elemento al mapa  
14    mapaD.remove("Telefono") //Elimina elementos del mapa  
15  
16    println(mapaD)  
17 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath mai  
n.jar MainKt  
{Nombre=Karen Muñoz, Formación=ADSI, Número de ficha=2  
469285, Trimeste=5}  
> []
```

Ejemplo con mapa inmutable:

Se sabe que por definición son mapas inmutables, es decir que no se puede modificar sus elementos, pero si necesitamos modificar los elemento de un mapa con estas características, debemos de crear un nuevo mapa con los elementos actualizados, es como decir si tenemos un mapa inmutable y queremos actualizarlo, pese a que no se pueda, hay una manera de hacerlo y es creando un mapa nuevo que tenga los mismos elementos que el mapa original, pero claramente con el valor actualizado, para ellos utilizamos la función plus() que se utiliza para crear un nuevo mapa que contiene las entradas del mapa original mas las entradas adicionales especificadas. Pero si se necesita realizar muchas operaciones de actualización en un mapa, es posible que sea más eficiente utilizar un mapa mutable en su lugar.

```
main.kt > ...
1 fun main(){
2     //Creación de mapa inmutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     //Modificar elementos de un mapa
9     val MapaE = mapaC.plus("Formación" to "ADSO")
10    println(MapaE)
11
12 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
{Identificación=1092445018, Nombre=Martha Muñoz, Telefono=3214567473, Formación=ADSO}
> []
```

e. Recorriendo un mapa

Los elementos de los mapas se pueden recorrer utilizando un bucle for, la cual itera sobre los pares clave-valor del mismo, como también se puede con la función forEach (), que se utiliza para iterar sobre las entradas del mapa  
Ejemplo con bucle for utilizando el operador in :

```
main.kt > f main > ...
1 fun main(){
2     //Creación de mapa inmutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     // Recorrer todos los pares clave-valor en un mapa
9     for ((clave, valor) in mapaC) {
10        println("La clave $clave tiene el valor de: $valor\n")
11    }
12
13    //Creación de mapa mutable
14    val mapaD = mutableMapOf(
15        "Nombre" to "Karina Sanchez",
16        "Formación" to "ADSI",
17        "Número de ficha" to 2469285
18    )
19    // Recorrer todos los pares clave-valor en un mapa
20    for ((clave, valor) in mapaD) {
21        println("La clave $clave tiene el valor de: $valor\n")
22    }
23 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
La clave Identificación tiene el valor de: 1092445018

La clave Nombre tiene el valor de: Martha Muñoz

La clave Telefono tiene el valor de: 3214567473

La clave Nombre tiene el valor de: Karina Sanchez

La clave Formación tiene el valor de: ADSI

La clave Número de ficha tiene el valor de: 2469285
> []
```

Ejemplo con función forEach:

```
main.kt > f main > ...
1 fun main(){
2     //Creación de mapa inmutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     // Recorrer todos los pares clave-valor en un mapa
9     mapaC.forEach { (clave, valor) ->
10        println("La clave $clave tiene el valor de: $valor\n")
11    }
12
13    //Creación de mapa mutable
14    val mapaD = mutableMapOf(
15        "Nombre" to "Karina Sanchez",
16        "Formación" to "ADSI",
17        "Número de ficha" to 2469285
18    )
19    // Recorrer todos los pares clave-valor en un mapa
20    mapaD.forEach { (clave, valor) ->
21        println("La clave $clave tiene el valor de: $valor\n")
22    }
23 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
La clave Identificación tiene el valor de: 1092445018

La clave Nombre tiene el valor de: Martha Muñoz

La clave Telefono tiene el valor de: 3214567473

La clave Nombre tiene el valor de: Karina Sanchez

La clave Formación tiene el valor de: ADSI

La clave Número de ficha tiene el valor de: 2469285
> []
```

f. Funciones útiles para trabajar con mapas en Kotlin

Kotlin nos brinda la posibilidad de varias funciones que son útiles para trabajar con mapas como, por ejemplo:

- **getOrDefault:** Permite obtener el valor asociado a una clave en un mapa, o un valor predeterminado si la clave no está presente en el mapa.

Ejemplo:



```
main.kt > ...

1 fun main(){
2     //Creación de mapa inmutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473,
7
8         "Identificación" to 1032463663,
9         "Nombre" to "Karen Cruz",
10        "Telefono" to 3245635652,
11
12        "Identificación" to 6424535324,
13        "Nombre" to "Marina Oliveros",
14        "Telefono" to 3214535252
15    )
16    //Función getOrDefault
17    val Mapear = mapaC.getOrDefault("Nombre",
18        3214535252) //Devuelve el nombre de la persona
19    //Devuelve el nombre de la persona
20    println(Mapear)
21
22    //Creación de mapa mutable
23    val mapaD = mutableMapOf(
24        "Nombre" to "Karina Sanchez",
25        "Formación" to "ADSI",
26        "Número de ficha" to 2469285,
27
28        "Nombre" to "Cristian Luna",
29        "Formación" to "ADSI",
30        "Número de ficha" to 2469285,
31
32        "Nombre" to "Natalia Buendía",
33        "Formación" to "ADSO",
34        "Número de ficha" to 2445674
35    )
36    //Función getOrDefault
37    val Buscar = mapaD.getOrDefault("Formación",
38        2445674) //Devuelve el nombre de la formación
39    println(Buscar)
40 }
```

```
> kotlinc -d main.jar main.kt && kotlin -class
path main.jar MainKt
Marina Oliveros
ADSO
> []
```

- **getOrDefault:** Es similar a la anterior, pero en lugar de devolver un valor predeterminado, devuelve un valor calculado a partir de una función lambda.

**Ejemplo:**

```
main.kt > ...

1 fun main(){
2     //Creación de mapa immutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     //Función getOrElse
9     val Mapear = mapaC.getOrElse("Nombre") {0}
10    // Obtener el valor correspondiente a una clave
    existente en el mapa
11    println("El nombre del aprendiz es: $Mapear")
12
13    //Creación de mapa mutable
14    val mapaD = mutableMapOf(
15        "Nombre" to "Karina Sanchez",
16        "Formación" to "ADSI",
17        "Número de ficha" to 2469285
18    )
19    //Función getOrElse
20    val Buscar = mapaD.getOrElse("Formación") {0}
21    // Obtener el valor correspondiente a una clave
    existente en el mapa
22
23    println("El nombre de la formación es: $Buscar")
24 }
```

```
> kotlinc -d main.jar main.kt && kotlin -class
path main.jar MainKt
El nombre del aprendiz es: Martha Muñoz
El nombre de la formación es: ADSI
> []
```

- **filter:** Devuelve un nuevo mapa que contiene sólo las entradas que cumplen con un determinado criterio.

Ejemplo:

```
main.kt > ...

1 fun main(){
2     //Creación de mapa immutable
3     val mapaC = mapOf(
4         "Identificación" to 8,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     //Función filter
9
10    // Filtra los elementos del mapa que tienen un
    valor igual a 8
11    val Mapear = mapaC.filter { it.value == 8 }
12    // Obtener el valor correspondiente a una clave
    existente en el mapa
13    println("La identificación es: $Mapear")
14
15    //Creación de mapa mutable
16    val mapaD = mutableMapOf(
17        "Nombre" to "Karina Sanchez",
18        "Formación" to "ADSI",
19        "Número de ficha" to 5
20    )
21    //Función filter
22
23    // Filtra los elementos del mapa que tienen un
    valor igual a 5
24    val Buscar = mapaD.filter { it.value == 5 }
25    // Obtener el valor correspondiente a una clave
    existente en el mapa
26
27    println("El número de la ficha es: $Buscar")
28 }
```

```
> kotlinc -d main.jar main.kt && kotlin -class
path main.jar MainKt
La identificación es: {Identificación=8}
El número de la ficha es: {Número de ficha=5}
> []
```

- **map:** Devuelve un nuevo mapa que contiene las entradas transformadas por una función lambda.

Ejemplo:

```

main.kt > ...
8 //Función map
9 val Mapear = mapaF.map { (clave,
    valor) ->
10     val NewClave = "nuevo_${clave}"
11     val NewValor = valor * 1.1
12     NewClave to NewValor
13 }
14
15 println("Se modificaron los precios
    de los productos a continuación los
    precios actuales: $Mapear\n") //
    Devuelve el valor del nuevo mapa que
    será el precio del producto del mapa
    original multiplicado por 1.1 (un
    aumento del 10%).
16
17     //Creación de mapa mutable
18 val mapaD = mutableMapOf(
19     "producto1" to 600.0,
20     "producto2" to 200.0,
21     "producto3" to 900.0
22 )
23 //Función map
24 val Buscar = mapaD.map { (clave,
    valor) ->
25     val nuevaClave = "nuevo_${clave}"
26     val nuevoValor = valor * 1.1
27     nuevaClave to nuevoValor
28 }
29
30 println("Se modificaron los precios
    de los productos a continuación los
    precios actuales: $Buscar") //
    Devuelve el valor del nuevo mapa que
    será el precio del producto del mapa
    original multiplicado por 1.1 (un
    aumento del 10%).
31 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar
MainKt
Se modificaron los precios de los productos a continuación
los precios actuales: [(nuevo_producto1, 330.0), (nuevo_pro
ducto2, 550.0), (nuevo_producto3, 660.0)]

Se modificaron los precios de los productos a continuación
los precios actuales: [(nuevo_producto1, 660.0), (nuevo_pro
ducto2, 220.000000000000003), (nuevo_producto3, 990.00000000
000001)]
> []

```

- **toMutableMap:** Convierte un mapa inmutable en un mapa mutable.

**Ejemplo:**

```

main.kt > ...
1 fun main(){
2     //Creación de mapa inmutable
3 val Producto = mapOf(
4     "producto1" to "Manzana",
5     "producto2" to "Coco",
6     "producto3" to "Suavitel"
7 )
8 //Función toMutableMap
9 val Productos = Producto.toMutableMap() //Convierte el mapa
    inmutable a mutable
10 println("Se ha modicado el mapa inmutable a mutable, quedaría
    de la siguiente manera: $Productos")
11 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.
jar MainKt
Se ha modicado el mapa inmutable a mutable, quedaría de
la siguiente manera: {producto1=Manzana, producto2=Coco,
producto3=Suavitel}
> []

```

- **putAll:** Agrega todas las entradas de un mapa a otro mapa existente.

**Ejemplo:**

```
main.kt > ...

1 fun main(){
2     //Creación de mapa inmutable
3     val mapaC = mapOf(
4         "Identificación" to 1092445018,
5         "Nombre" to "Martha Muñoz",
6         "Telefono" to 3214567473
7     )
8     //Función putAll
9     //Creación de mapa mutable
10    val mapaD = mutableMapOf(
11        "Jornada" to "Mixta",
12        "Formación" to "ADSI",
13        "Número de ficha" to 2469285
14    )
15    mapaD.putAll(mapaC) //Agrega todos los
    pares clave-valor del mapaC al mapaD
16    println("Los siguientes datos corresponden
    a un aprendiz: $mapaD")
17
18 }
```

```
> kotlinc -d main.jar main.kt && kotlin
-classpath main.jar MainKt
Los siguientes datos corresponden a un a
prendiz: {Jornada=Mixta, Formación=ADSI,
Número de ficha=2469285, Identificación
=1092445018, Nombre=Martha Muñoz, Telefo
no=3214567473}
> []
```

## 6. Pares en Kotlin

### a. ¿Qué es un par?

Es una estructura de datos que se utiliza para representar un conjunto ordenado de dos elementos. Digamos que este par de define utilizando la clase Pair que tiene dos propiedades: first y second que contienen los dos elementos. Son útiles en mucho caso, cuando se necesita devolver dos valores de una función. En lugar de crear una clase personalizada para representar los dos valores, se puede utilizar un par.

### b. Creación de pares en Kotlin

Se pueden crear pares utilizando la función Pair. Esta función toma dos argumentos y devuelve un par que contiene estos valores, como también se puede utilizar el operador to.

Ejemplo:

```
main.kt > ...

1 fun main(){
2     //Creación de pares
3
4     //Operador to
5     val Aprendiz1 = "María" to 17
6
7     println("El nombre del aprendiz es:
    $Aprendiz1")
8
9     //Funcion Pair
10    val Aprendiz2 = Pair("Martha", 18)
11
12    println("El nombre del aprendiz es:
    $Aprendiz2")
13
14 }
```

```
> kotlinc -d main.jar main.kt && ko
tlin -classpath main.jar MainKt
El nombre del aprendiz es: (María,
17)
El nombre del aprendiz es: (Martha,
18)
> []
```

### c. Accediendo a los elementos de un par

Para acceder a los valores de un par, se pueden utilizar las propiedades first y second, que devuelven el primer y el segundo valor, respectivamente.

Ejemplo:

```

main.kt > ...
4 //Operador to
5 val Aprendiz1 = "María" to 17
6
7
8 //Acceder a elemento de un par
9 val nombreAprendiz1 =
  Aprendiz1.first
10 val edadAprendiz1 = Aprendiz1.second
11 println("El aprendiz
  $nombreAprendiz1 ha cumplido este
  año $edadAprendiz1")
12
13 //Funcion Pair
14 val Aprendiz2 = Pair("Martha", 18)
15 //Acceder a elemento de un par
16 val nombreAprendiz2 =
  Aprendiz2.first
17 val edadAprendiz2 = Aprendiz2.second
18 println("El aprendiz
  $nombreAprendiz2 ha cumplido este
  año $edadAprendiz2")
19 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El aprendiz María ha cumplido este
año 17
El aprendiz Martha ha cumplido este
año 18
> 

```

#### d. Modificando los elementos de un par

La función copy devuelve una copia del par original con los valores especificados.

#### Ejemplo:

```

main.kt > f main > ...
1 fun main(){
2 //Creación de pares
3
4 //Funcion Pair
5 val Aprendiz = Pair("Martha", 18)
6 val ModifAprendiz = Aprendiz.copy(second =
  19) // Actualizar un elemento del par
7
8 println("El aprendiz $Aprendiz ha hecho un
  cambio, el cambio es el siguiente
  $ModifAprendiz")
9 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El aprendiz (Martha, 18) ha hecho un cambio
, el cambio es el siguiente (Martha, 19)
> 

```

También se puede utilizar la función component1() y component2() para descomponer el par en sus valores individuales, modificar los valores y crear un nuevo par.

#### Ejemplo:

```

main.kt > ...
1 fun main(){
2 //Creación de pares
3 //Funcion Pair
4 val Aprendiz = Pair("Martha", 18)
5
6 val nombre = Aprendiz.component1()
7 val edad = Aprendiz.component2()
8
9 val ModifAprendiz = Pair(nombre, edad + 5)
10 println("El aprendiz $Aprendiz ha hecho un
  cambio, el cambio es el siguiente
  $ModifAprendiz") //modifica los valores y
  crear un nuevo par
11 }

```

```

> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El aprendiz (Martha, 18) ha hecho un cambio
, el cambio es el siguiente (Martha, 23)
> 

```

Los pares en Kotlin son objetos inmutables y no se pueden modificar una vez creados. No obstante, si se desea eliminar un par de una colección, se puede utilizar el método remove o removelf de la colección que contiene el par.

#### Ejemplo:



```
main.kt > ...
1 fun main(){
2     //Creación de pares
3
4     //Funcion Pair
5     val Aprendices = mutableListOf (
6         Pair("Martha", 15),
7         Pair("Valentina", 16),
8         Pair("Ana", 18)
9     )
10    Aprendices.removeAt(2) //Elimina el par
    según la posición asignada
11    println("Los aprendices inscriptos son: ")
12    println(Aprendices)
13 }
```

```
> kotlinc -d main.jar main.kt && kotlin -cl
asspath main.jar MainKt
Los aprendices inscriptos son:
[(Martha, 15), (Valentina, 16)]
> []
```

### Ejemplo:

```
main.kt > ...
1 fun main(){
2     val lista = listOf(
3         Pair("Juan", 25),
4         Pair("Pedro", 30),
5         Pair("María", 27)
6     )
7
8     val NewPareja =
        lista.toMutableList()
9     NewPareja.removeIf { it.first ==
        "Juan" }
10    println("La pareja del año es:
        $NewPareja") //Elimina el par
        según la condición asignada
11 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpat
h main.jar MainKt
La pareja del año es: [(Pedro, 30), (María, 27)]
> []
```

### e. Recorriendo un par

Para recorrer un par se utiliza el ciclo for, en este caso una lista de pares, así que se descompone cada par en sus valores.

### Ejemplo:

```
main.kt > ...
1 fun main(){
2     val Aprendiz = listOf(
3         Pair("Martha", 25),
4         Pair("Lina", 16),
5         Pair("Andrea", 18)
6     )
7
8
9     for (Estudiante in Aprendiz) {
10         val nombre = Estudiante.first
11         val edad = Estudiante.second
12         println("$nombre tiene $edad años")
13     }
14 }
```

```
> kotlinc -d main.jar main.kt && kotlin -cl
asspath main.jar MainKt
Martha tiene 25 años
Lina tiene 16 años
Andrea tiene 18 años
> []
```

### f. Funciones útiles para trabajar con pares en Kotlin

Las funciones útiles para trabajar con pares que nos brinda Kotlin son:

- **to**: Crea un nuevo par a partir de dos valores.

### Ejemplo:

```
main.kt > ...
1 fun main(){
2     //Creación de pares
3     //Función to
4     val Aprendiz = "Manuela Cordoba" to 18
5     println(Aprendiz)
6 }
```

```
> kotlinc -d main.jar main.kt && kotl
in -classpath main.jar MainKt
(Manuela Cordoba, 18)
> []
```

- **toList:** Convierte un par en una lista que contiene sus elementos.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de pares
3     //Función toList
4     val Aprendiz = Pair("Juan Duque",20)
5     val Adsi = Aprendiz.toList()
6     println(Adsi)
7
8 }
```

```
➤ kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
[Juan Duque, 20]
➤
```

- **component1 y component2:** Devuelven el primer y segundo elemento del par, respectivamente, se utilizan principalmente en la desestructuración de pares.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de pares
3     //Función component1 y component2
4     val Aprendiz = Pair("Juan Duque",20)
5     val nombre = Aprendiz.component1()
6     val edad = Aprendiz.component2()
7     println("En la ficha 2469285 se encuentra registrado el aprendiz $nombre y su edad es $edad")
8
9 }
```

```
➤ kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
En la ficha 2469285 se encuentra registrado el aprendiz Juan Duque y su edad es 20
➤
```

- **equals:** Compara dos pares para determinar si son iguales. Dos pares son iguales si sus primeros elementos son iguales y sus segundos elementos son iguales.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de pares
3
4     //Función equals
5     val Aprendiz1 = Pair("Karen", 25)
6     val Aprendiz2 = Pair("Karen", 25)
7     val Aprendiz3 = Pair("Pablo", 30)
8
9     if(Aprendiz1.equals(Aprendiz2) || Aprendiz1.equals(Aprendiz3)) {
10         println("Hay campos repetidos")
11     }else{
12         println("No hay campos repetidos")
13     }
14 }
```

```
➤ kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
Hay campos repetidos
➤
```

- **hashCode:** Devuelve un valor de hash para el par. Este valor se utiliza para identificar el par en una tabla hash o en otras estructuras de datos que utilizan hashing.

**Ejemplo:**

```
main.kt > ...
1 fun main(){
2     //Creación de pares
3
4     //Función hashCode
5     val Aprendiz = Pair("Karen Masmela", 18)
6     val Identificador = Aprendiz.hashCode()
7     println("El identificador del aprendiz es: $Identificador") // Devuelve un valor de hash para el par
8 }
```

```
➤ kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
El identificador del aprendiz es: 1266023713
➤
```

- **toString:** Devuelve una representación en cadena de caracteres del par. La representación incluye los elementos del par separados por una coma y encerrados entre paréntesis.

**Ejemplo:**

```
main.kt
1 fun main(){
2     //Creación de pares
3
4     //Función toString
5     val Aprendiz = Pair("Pablo", 30)
6     val Aprendizices = Aprendiz.toString()
7     println("Los datos del aprendiz sancionado son: $Aprendices")
8 }
```

```
> kotlinc -d main.jar main.kt && kotlin -classpath main.jar MainKt
Los datos del aprendiz sancionado son: (Pablo, 30)
> 
```

## 7. Prácticas de estructuras de datos en Kotlin

- a. Ejercicios prácticos para aplicar los conceptos aprendidos
- b. Solución a los ejercicios prácticos

Link de los ejercicios:

<https://github.com/staruniverse2050/Estructuras-de-Control.git>

Recursos adicionales: Taller Estructuras de Datos en Kotlin

Documentación oficial de Kotlin: <https://kotlinlang.org/docs/reference/>

## Entrega.

Se deberá realizar la entrega de un informe con la solución de los puntos anteriores, el aprendiz acompañará la investigación con ejemplos prácticos de cada estructura y deberá publicar el código fuente en un repositorio en GitHub.