**Taller Estructuras de Datos en Kotlin**

Santiago Quesada Marulanda

SENA centro comercio y turismo

ADSI 2469283

Cristian David Henao Hoyos

20 de febrero del 2023

**Estructuras de Datos en Kotlin**

Las estructuras de datos son formas organizadas de almacenar y gestionar datos en un programa informático. Son utilizadas por los programadores para representar y manipular información de manera eficiente, con el objetivo de realizar operaciones como búsqueda, inserción, eliminación y modificación de datos de forma eficiente y rápida.

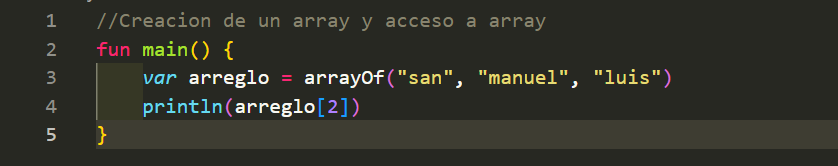
Hay muchos tipos de estructuras de datos, incluyendo arrays, listas enlazadas, pilas, colas, árboles, grafos, entre otros. Cada una de estas estructuras tiene sus propias características y aplicaciones específicas, y la elección de una estructura de datos adecuada dependerá del problema que se esté intentando resolver.

El uso de estructuras de datos eficientes es fundamental para el desarrollo de aplicaciones informáticas complejas, ya que puede marcar una gran diferencia en la velocidad y la eficiencia del programa.

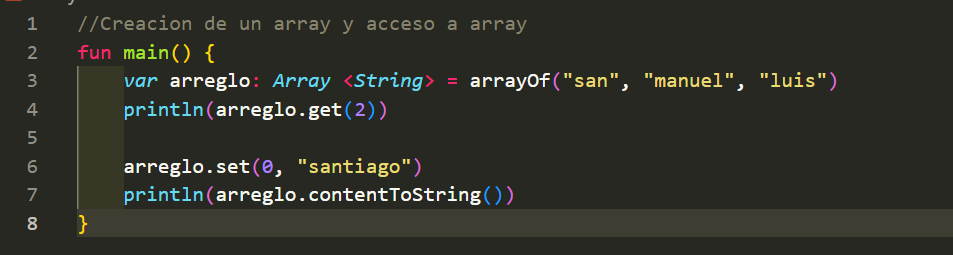
**Arrays**

Un arreglo, también conocido como array en inglés, es una estructura de datos en la programación que permite almacenar un conjunto de valores del mismo tipo en una sola variable.

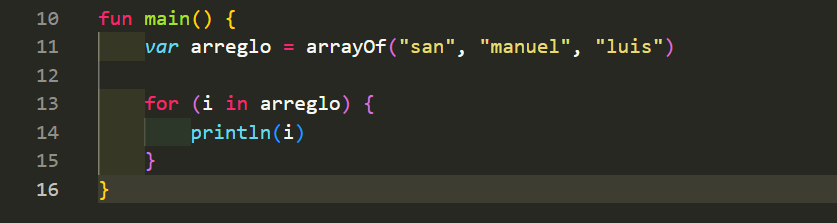
Para la creación de arrays se hace uso del constructor **Array()** como también la función **ArrayOf(),** en estas se debe definir el tipo de dato que contendrá. Ahora para acceder a los elementos de un arreglo se debe definir a que índice se debe ingresar en ese caso:



Para la modificación de un array en particular se hace uso de **.set()** en donde se debe indicar primero el índice que se quiere modificar seguido de su nuevo valor; en cuanto a la obtención de cada elemento se hace uso de la función .**get()** que traerá el valor de los índices.



En caso de recorrer un arreglo se hace uso del **for,** este como anteriormente se había dicho recorre cada uno de los elementos existentes que contiene un arreglo y realiza una acción definida, como en este caso, devolverá todos los nombres que están en el arreglo.



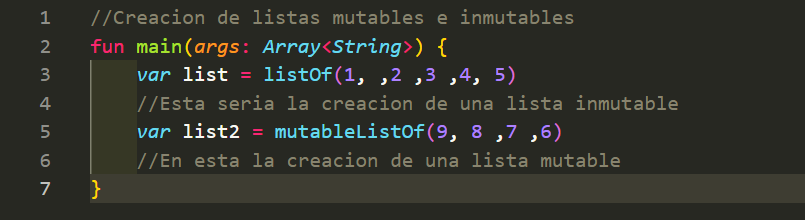
1. size: Retorna el número de elementos en una lista.
2. get: Retorna el elemento en una posición específica de la lista. La lista comienza en la posición 0.
3. indexOf: Retorna el índice de la primera aparición de un elemento en una lista.
4. lastIndexOf: Retorna el índice de la última aparición de un elemento en una lista.
5. contains: Retorna un valor booleano que indica si un elemento está presente en una lista.
6. subList: Retorna una lista que es una vista de una parte de la lista original.
7. filter: Retorna una lista que contiene solo los elementos que cumplen con un criterio específico.
8. map: Retorna una lista que contiene los resultados de aplicar una función a cada elemento de la lista original.
9. reduce: Combina los elementos de la lista usando una función y retorna el resultado final.
10. sorted: Retorna una lista que contiene los elementos de la lista original ordenados de acuerdo a un orden específico.

**Listas**

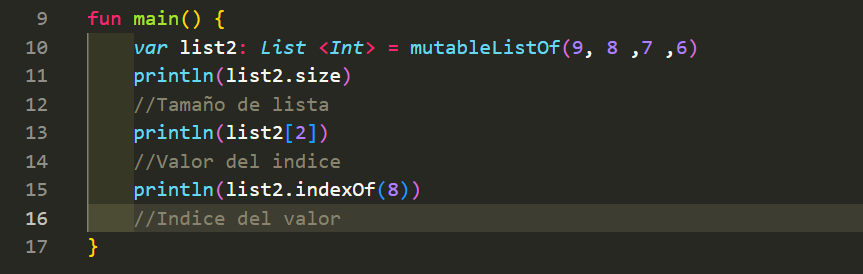
En Kotlin, una lista es una colección de elementos ordenados, que pueden ser de cualquier tipo de dato. Se pueden agregar, eliminar y acceder a los elementos de una lista mediante su índice.

Las listas en Kotlin se pueden declarar utilizando la interfaz List o la clase MutableList, dependiendo si se desea que la lista sea inmutable o mutable.

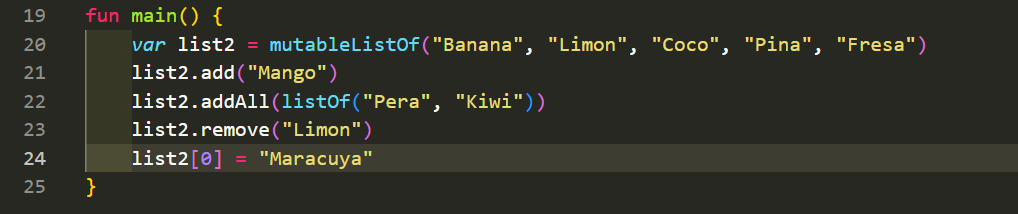
Para la creación de ambas se necesita definir como variable y nombre **listOf()**, si la lista es inmutable solo se define la lista, pero si es inmutable se debe definir que es de tipo inmutable **mutableListOf()**, finalmente accedemos a la lista



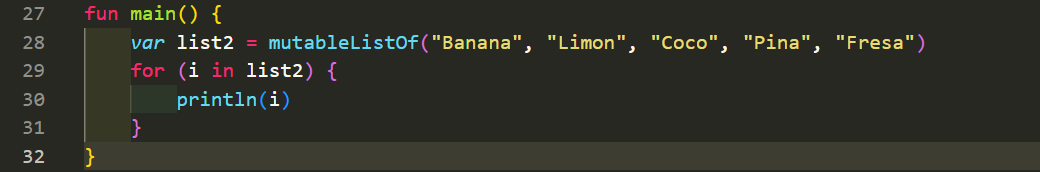
Para acceder a una lista se puede utilizar diferentes funciones como: size para obtener la cantidad de elementos de la lista, lista[index] para obtener el elemento ubicado en index, indexOf(element) para obtener el índice de la primera ocurrencia de elemento.



Para la modificación de listas se cuenta con varios métodos como. add**()** el cual agregara un nuevo elemento al final de la lista, para remover un elemento se usa el método **.remove()**, en caso de que se deba actualizar un elemento se el operador de indexación y para ordenar los elementos de una lista se hace uso de **.sort().**



En caso de recorrer un arreglo se hace uso del for, este como anteriormente se había dicho recorre cada uno de los elementos existentes que contiene un arreglo y realiza una acción definida, como en este caso, devolverá todos los nombres que están en el arreglo.

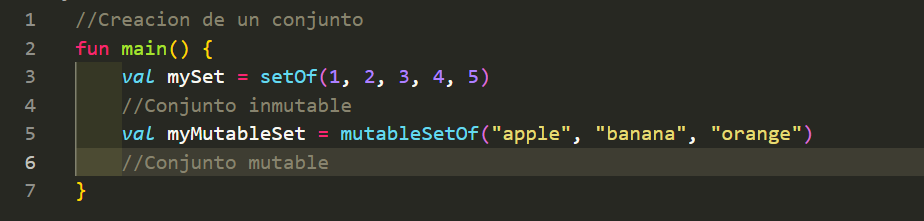


1. size: Retorna el número de elementos en una lista.
2. get: Retorna el elemento en una posición específica de la lista. La lista comienza en la posición 0.
3. indexOf: Retorna el índice de la primera aparición de un elemento en una lista.
4. lastIndexOf: Retorna el índice de la última aparición de un elemento en una lista.
5. contains: Retorna un valor booleano que indica si un elemento está presente en una lista.
6. subList: Retorna una lista que es una vista de una parte de la lista original.
7. filter: Retorna una lista que contiene solo los elementos que cumplen con un criterio específico.
8. map: Retorna una lista que contiene los resultados de aplicar una función a cada elemento de la lista original.
9. reduce: Combina los elementos de la lista usando una función y retorna el resultado final.
10. sorted: Retorna una lista que contiene los elementos de la lista original ordenados de acuerdo a un orden específico.

**Conjunto**

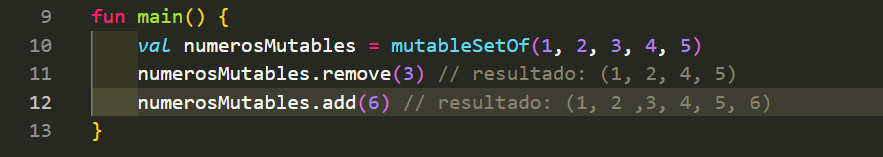
En Kotlin, un conjunto (set en inglés) es una colección de elementos que no permite duplicados y cuyo orden no está definido. En otras palabras, un conjunto es una colección de elementos únicos, sin importar el orden en que se agregaron.

En Kotlin, un conjunto se puede crear utilizando la función setOf() o mutableSetOf(), dependiendo de si se desea un conjunto inmutable o mutable, respectivamente.

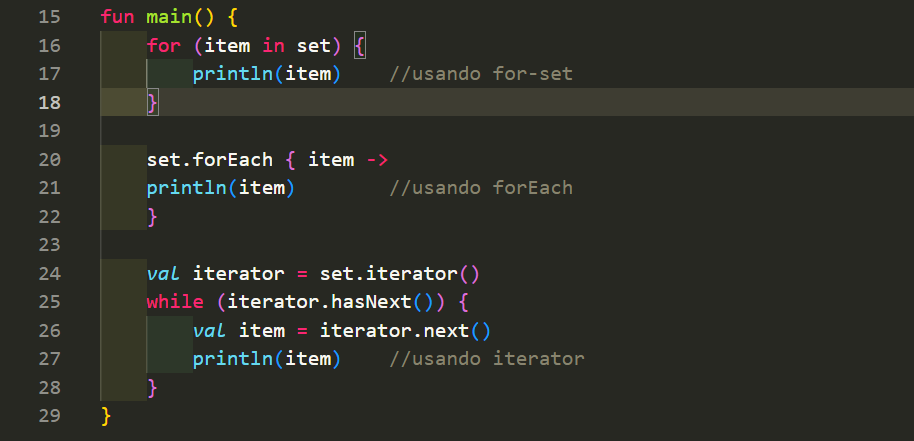


Para acceder a un conjunto primero se debe crear uno. Como por ejemplo de tipo **String** en donde se utiliza la implementación **val conjunto = HashSet<String>().** Se puede utilizar el .add() para poder agregar elementos al conjunto: **conjunto.add(“elemento1”).** Tambien se puede utilizar **.contains()** para verificar si un elemento se encuentra presente en el conjunto **conjunto.contains(“elemento1”).**

Para poder modificar un conjunto, inicialmente debe ser mutable ya que en caso contrario no se podrá modificar. Se puede usar .add() para poder agregar un nuevo elemento dentro del conjunto, así como también se puede usar .remove() para eliminar alguno de los elementos existentes en el conjunto



Existen varias maneras para recorrer un conjunto, como por ejemplo: haciendo uso de **for,** utilizando .**forEach** y usando la función **iterator**

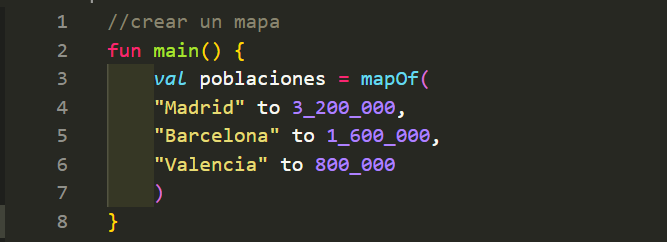


Algunas de las funciones que pueden servir en el uso de los conjuntos pueden ser el **setOf** que permite crear un conjunto inmutable y **mutableSetOf** para los conjuntos mutables, **add()** se usa para agregar un elemento al conjunto asi como también **addAll()** que permite agregar varios elementos, de lo contrario para eliminar un elemento se recurre a la función **remove().**

**Contains** se utiliza para verificar si un elemento se encuentra en el conjunto, .**unión()** sirve para combinar dos conjuntos existentes, **.subtract()** puede ser utilizado para sacar la diferencia entre dos conjuntos

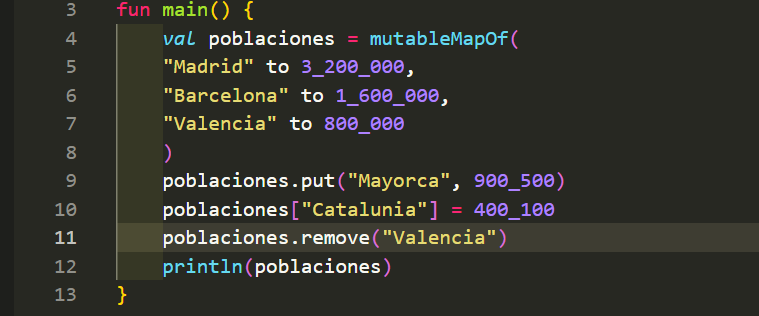
**Mapas**

Los mapas son colecciones de elementos que se pueden almacenar en pares **Clave-Valor**, es decir que asocia un valor a un código o clave que lo identifica. Para a creación de un mapa se implementa la interfaz **Map.**

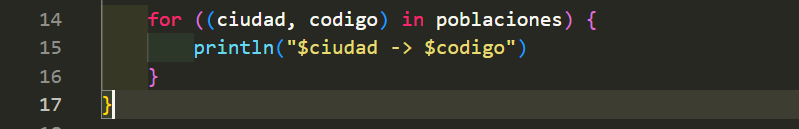


Para acceder a un mapa em Kotlin lo principal es utilizar su clave, creando un **hashMapOf**  que es la forma de instanciar el hashMap. La sintaxis clave to valor permite agregar elementos a los mapas.

La modificación de los elementos en un mapa es sencillo ya que para agregar un elemento se da uso al método **.put()** que lleva el valor del elemento y su clave, para eliminar un elemento se usa el método **.remove()** que solo lleva el valor del elemento.



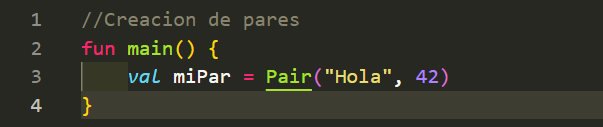
Debido a la naturaleza de los mapas en Kotlin, es posible desestructurar las declaraciones que comprometan a sus entradas en valores individuales.

Un ejemplo claro de esto es recorrer sobre los elementos de un mapa en un bucle for: 

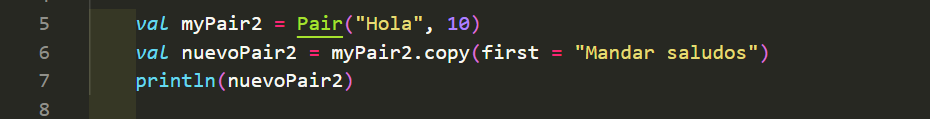
**Pares**

En Kotlin, los "pairs" (parejas) son un tipo de datos que se utilizan para agrupar dos valores en una única estructura. Se puede pensar en un par como una tupla de dos elementos, donde el orden de los elementos importa.

En Kotlin, un par se declara utilizando la función Pair(), que toma dos valores como argumentos y devuelve un objeto de tipo Pair. Por ejemplo:



En Kotlin los pares son inmutables, esto quiere decir que no hay forma directa de realizar modificaciones en los pares. Sin embargo, si es posible realizar una copia de los pares actualizados a partir del par inicial.



Dado que un par en Kotlin es inmutable, no hay forma de recorrerlos como una colección de elementos, pero si se puede acceder a cada uno de los valores individuales utilizando las propiedades **first** y **second.**

1. first y second: son propiedades de Pair que permiten acceder al primer y segundo valor de un par.
2. component1 y component2: son funciones que permiten desestructurar un par en dos variables separadas.
3. toList y toSet: son funciones que convierten un par en una lista o un conjunto.
4. quals y hashCode: son funciones que permiten comparar pares por igualdad. Por defecto, dos pares son iguales si sus primeros y segundos valores son iguales.
5. copy: es una función que permite crear una copia de un par, cambiando uno o ambos de sus valores.
6. copy: es una función que permite crear una copia de un par, cambiando uno o ambos de sus valores.