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

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

Las estructuras de datos son una manera de organizar y almacenar información de una manera eficiente y accesible en programas informáticos. En Kotlin, las estructuras de datos se utilizan para almacenar y manipular datos de diferentes tipos, como números, cadenas, objetos, entre otros.

b. Ventajas de utilizar estructuras de datos en Kotlin

Kotlin ofrece varias estructuras de datos incorporadas, como listas, conjuntos, mapas y pares, que proporcionan una serie de ventajas para los programadores:

-Eficiencia en el acceso a datos

-Mantenimiento de la integridad de los datos

- Fácil manejo de datos

- Reutilización de código

- Combinación con otras características de Kotlin

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

-Sintaxis: Kotlin y Java tienen una sintaxis diferente para declarar y trabajar con estructuras de datos. Por ejemplo, para crear una lista en Java, se usa la clase ArrayList, mientras que en Kotlin se usa la palabra clave listOf.

-Tipos nulos: Kotlin es un lenguaje más seguro en términos de nulabilidad que Java, por lo que en Kotlin todas las estructuras de datos pueden ser nulas por defecto. En Java, las estructuras de datos no pueden ser nulas y, por lo tanto, requieren una verificación adicional para garantizar que no se produzcan excepciones de puntero nulo.

-Inmutabilidad: en Kotlin, las estructuras de datos son inmutables de forma predeterminada. Esto significa que no se pueden modificar después de la creación, lo que evita errores de concurrencia y asegura la estabilidad del código. En Java, las estructuras de datos son mutables de forma predeterminada, por lo que los programadores deben tener cuidado de evitar la modificación accidental de los datos.

-Funcionalidades adicionales: Kotlin ofrece algunas funcionalidades adicionales para las estructuras de datos, como el filtrado y la transformación de elementos, que no se encuentran en las estructuras de datos en Java.

-Null-safety: Kotlin introduce un sistema de tipo de Null-safety para mejorar la seguridad de null en tiempo de compilación. Las estructuras de datos de Kotlin tienen soporte para la anotación de tipos nullable, lo que asegura que los programadores tengan un control más riguroso sobre los casos donde un dato podría o no estar presente.

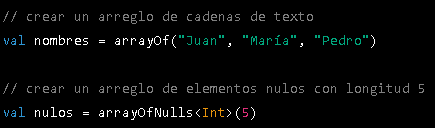
**2. Arreglos en Kotlin**

a. ¿Qué es un arreglo?

Una colección ordenada de elementos del mismo tipo, donde se pueden agregar, eliminar y acceder a los elementos por índice.

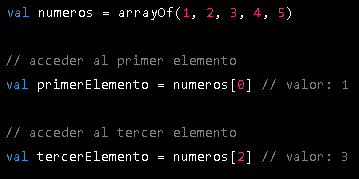
b. Creación de arreglos en Kotlin

Para crear un arreglo en Kotlin, puedes usar la función arrayOf() o la función arrayOfNulls().



c. Accediendo a los elementos de un arreglo

Para acceder a los elementos de un arreglo en Kotlin, puedes usar el operador de índice, que es el símbolo de corchetes []. El índice del primer elemento es 0, el índice del segundo elemento es 1, y así sucesivamente.



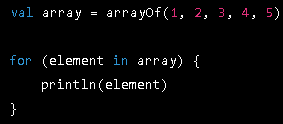
d. Modificando los elementos de un arreglo

Para modificar los elementos de un arreglo en Kotlin, simplemente usa el operador de índice [] para acceder al elemento que deseas modificar y luego asigna un nuevo valor al mismo.

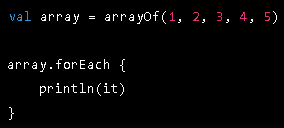


e. Recorriendo un arreglo

Puedes recorrer un arreglo en Kotlin usando un ciclo for o un ciclo while. Aquí te muestro cómo hacerlo con un ciclo for.



Recorrido con un bucle forEach:



f. Funciones útiles para trabajar con arreglos en Kotlin

-size: devuelve la longitud del arreglo

-max y min: devuelven el valor máximo y mínimo en el arreglo

-average: devuelve el promedio de todos los elementos del arreglo

-sorted y sortedDescending: devuelven una copia ordenada del arreglo en orden ascendente o descendente

-filter: devuelve una lista de elementos que cumplen con una condición dada

-map: devuelve una lista de los resultados de aplicar una función dada a cada elemento del arreglo

**3. Listas en Kotlin**

a. ¿Qué es una lista?

Una lista es una colección de elementos ordenados, donde cada elemento se puede acceder mediante un índice. Las listas en Kotlin pueden ser mutables o inmutables, dependiendo si se permite modificar su contenido una vez que se han creado.

b. Creación de listas en Kotlin

- Utilizando la función listOf()



- Utilizando la función arrayListOf()



Utilizando la función mutableListOf() para crear una lista mutable



c. Accediendo a los elementos de una lista

Se puede acceder a los elementos de una lista utilizando el operador de índice []





d. Modificando los elementos de una lista

Se puede modificar los elementos de una lista si esta es mutable. Las listas mutables se pueden crear utilizando la función mutableListOf() o alguna otra implementación de la interfaz MutableList.

Para modificar un elemento específico de la lista, se puede utilizar el operador de índice [].







También se pueden agregar nuevos elementos a la lista utilizando el método add() o addAll(). Por ejemplo, para agregar la palabra "uva" al final de la lista, se puede utilizar el método add()





O se puede agregar varios elementos a la lista a la vez utilizando el método addAll(). Por ejemplo, para agregar los elementos "mango" y "piña" al final de la lista, se puede utilizar el método addAll()





También se pueden eliminar elementos de la lista utilizando los métodos remove() o removeAt(). Por ejemplo, para eliminar el segundo elemento de la lista, que ahora es "pera", se puede utilizar el método removeAt()



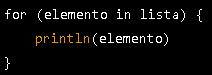


e. Recorriendo una lista

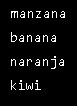
-for:

El bucle for es una forma sencilla y común de recorrer una lista en Kotlin





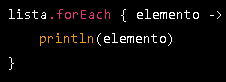
Este bucle imprimirá lo siguiente:

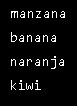


-forEach:

La función forEach() es una forma conveniente de recorrer una lista y realizar una acción en cada elemento



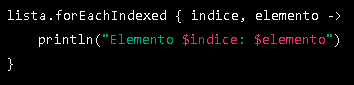




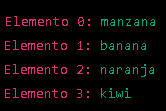
-forEachIndexed:

La función forEachIndexed() es similar a la función forEach(), pero también proporciona el índice de cada elemento





Este código imprime lo siguiente:



f. Funciones útiles para trabajar con listas en Kotlin

- size: Esta función devuelve el número de elementos en la lista.

- get: Esta función devuelve el elemento en el índice especificado. El índice comienza en cero.

- indexOf: Esta función devuelve el índice de la primera aparición del elemento especificado en la lista, o -1 si el elemento no se encuentra en la lista.

- contains: Esta función devuelve true si la lista contiene el elemento especificado, o false en caso contrario.

- filter: Esta función devuelve una nueva lista que contiene solo los elementos que cumplen con un cierto criterio. El criterio se especifica utilizando una función lambda que toma un elemento de la lista como argumento y devuelve un valor booleano.

- map: Esta función devuelve una nueva lista que contiene los resultados de aplicar una función dada a cada elemento de la lista. La función se especifica utilizando una función lambda que toma un elemento de la lista como argumento y devuelve un nuevo valor.

- intersect: en Kotlin es una función de extensión para colecciones, que se utiliza para encontrar los elementos comunes (la intersección) entre dos colecciones

- reduce: Esta función combina los elementos de la lista en un solo valor utilizando una función dada. La función se especifica utilizando una función lambda que toma dos argumentos y devuelve un nuevo valor. La función se aplica a los elementos de la lista de izquierda a derecha.

**4. Conjuntos en Kotlin**

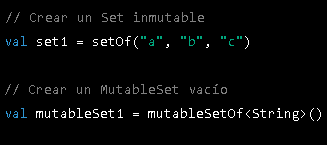
a. ¿Qué es un conjunto?

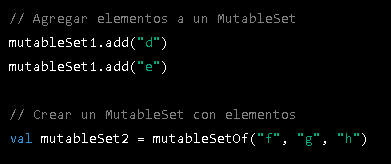
Los conjuntos (sets) son una estructura de datos que almacenan elementos únicos, es decir, no permiten duplicados. Los conjuntos son mutables, lo que significa que se pueden agregar y eliminar elementos después de crearlos.

b. Creación de conjuntos en Kotlin

En Kotlin, se pueden crear conjuntos utilizando la clase Set. Hay dos tipos de conjuntos: Set y MutableSet.

Un Set es inmutable, lo que significa que no se puede agregar, eliminar o modificar elementos una vez que se ha creado. Un MutableSet, por otro lado, es mutable, lo que significa que se pueden agregar, eliminar o modificar elementos después de que se ha creado.





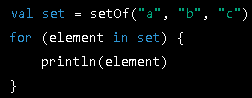
También se puede crear un conjunto utilizando la función setOf() o mutableSetOf(). Estas funciones toman una lista de elementos y crean un conjunto con ellos



c. Accediendo a los elementos de un conjunto

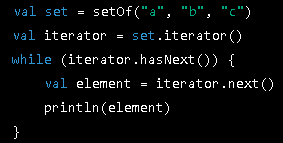
-for:

Se puede acceder a los elementos de un conjunto utilizando un bucle for o un iterador.

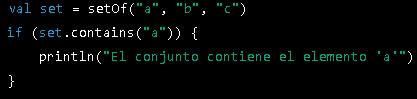
-iterator:

También se puede utilizar un iterador para recorrer los elementos de un conjunto. Para hacer esto, primero se debe obtener un iterador llamando al método iterator() del conjunto, y luego usar el iterador para acceder a los elementos uno por uno



-contains:

También se puede acceder a un elemento específico de un conjunto utilizando el método contains() para verificar si un elemento está presente en el conjunto. Si el elemento está presente, se puede acceder a él directamente mediante su nombre.



d. Modificando los elementos de un conjunto

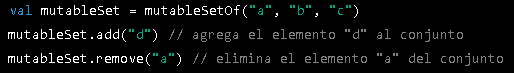
-add(element: E): agrega un elemento al conjunto

-remove(element: E): elimina un elemento del conjunto

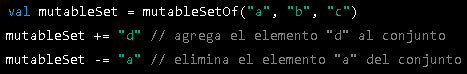
addAll(elements: Collection<E>): agrega varios elementos al conjunto

-removeAll(elements: Collection<E>): elimina varios elementos del conjunto

-retainAll(elements: Collection<E>): elimina todos los elementos del conjunto que no se encuentran en la colección especificada

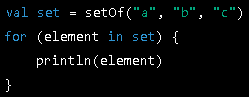


También se puede usar el operador += y -= para agregar y eliminar elementos de un conjunto mutable de manera más concisa:

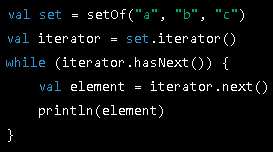


e. Recorriendo un conjunto

Para recorrer todos los elementos de un conjunto, se puede usar un bucle for



También se puede utilizar un iterador para recorrer los elementos de un conjunto. Para hacer esto, primero se debe obtener un iterador llamando al método iterator() del conjunto, y luego usar el iterador para acceder a los elementos uno por uno.



f. Funciones útiles para trabajar con conjuntos en Kotlin

setOf(vararg elements: T): Crea un conjunto inmutable a partir de los elementos especificados.

mutableSetOf(vararg elements: T): Crea un conjunto mutable a partir de los elementos especificados.

intersect(other: Iterable<T>): Set<T>: Devuelve un conjunto que contiene los elementos que están presentes en ambos conjuntos.

union(other: Iterable<T>): Set<T>: Devuelve un conjunto que contiene los elementos de ambos conjuntos.

subtract(other: Iterable<T>): Set<T>: Devuelve un conjunto que contiene los elementos del primer conjunto que no están presentes en el segundo conjunto.

filter(predicate: (T) -> Boolean): Set<T>: Devuelve un conjunto que contiene los elementos que cumplen con la condición especificada por el predicado.

map(transform: (T) -> R): Set<R>: Devuelve un conjunto que contiene los resultados de aplicar la función de transformación a cada elemento del conjunto original.

toSet(): Set<T>: Convierte una lista o cualquier otra colección en un conjunto.

distinct(): Set<T>: Devuelve un conjunto que contiene solo los elementos únicos del conjunto original.

**5. Mapas en Kotlin**

a. ¿Qué es un mapa?

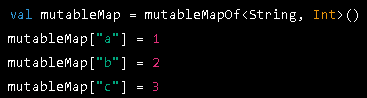
Los mapas son una estructura de datos que almacena pares clave-valor. Cada clave debe ser única y se utiliza para acceder a su respectivo valor. Los mapas son mutables, lo que significa que se pueden agregar, actualizar y eliminar pares clave-valor después de crearlos.

b. Creación de mapas en Kotlin

Para crear un mapa en Kotlin, puedes usar la función mapOf() que devuelve un mapa inmutable. Aquí hay un ejemplo de cómo crear un mapa con algunas parejas clave-valor:



También puedes crear un mapa mutable usando la clase MutableMap:



c. Accediendo a los elementos de un mapa

Para acceder a los valores de un mapa, puedes usar la sintaxis de corchetes y la clave correspondiente:



d. Modificando los elementos de un mapa

Puedes usar el método put para agregar elementos a un mapa mutable. Este método devuelve el valor anterior asociado con la clave, o null si la clave no estaba presente en el mapa:

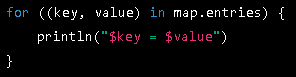


Para eliminar una entrada del mapa, puedes usar el método remove:



e. Recorriendo un mapa

Para iterar sobre las entradas del mapa, puedes usar un bucle for y la función entries:



También puedes usar la función forEach para hacer lo mismo:



f. Funciones útiles para trabajar con mapas en Kotlin

-getOrDefault: Devuelve el valor asociado con la clave especificada, o un valor predeterminado si la clave no está presente en el mapa.

- getOrElse: Devuelve el valor asociado con la clave especificada, o un valor calculado por una lambda si la clave no está presente en el mapa.

- filter: Devuelve un nuevo mapa que contiene solo los pares clave-valor que cumplen con una condición dada.

- map: Devuelve un nuevo mapa que contiene los resultados de aplicar una función a cada valor del mapa original.

- keys y values: Devuelve un conjunto que contiene todas las claves o valores del mapa, respectivamente.

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

- forEach: Itera sobre todas las entradas del mapa y aplica una función a cada entrada.

**6. Pares en Kotlin**

a. ¿Qué es un par?

Un par es una estructura de datos que representa dos valores. Es una clase genérica llamada Pair, que tiene dos propiedades llamadas first y second. Puedes crear pares utilizando la función Pair() o el operador to.

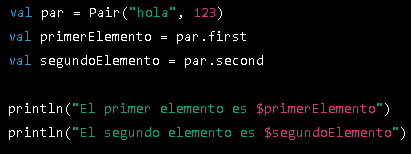
b. Creación de pares en Kotlin

La función Pair() toma dos argumentos y devuelve un objeto de tipo Pair. El primer argumento es el valor para el primer elemento del par, y el segundo argumento es el valor para el segundo elemento del par.

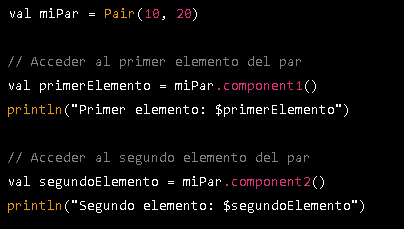


c. Accediendo a los elementos de un par

Se pueden acceder a los elementos individuales del par mediante las propiedades first y second



puedes utilizar la función component1() y component2() para acceder al primer y segundo elemento del par respectivamente.



d. Modificando los elementos de un par

Para actualizar los valores de un par, se pueden usar las funciones copy() o let() en combinación con la función Pair().

La función copy() devuelve un nuevo objeto de tipo Pair con valores actualizados para uno o ambos elementos del par original.



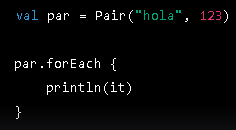
La función let() toma el par original como argumento y devuelve un nuevo par con los valores actualizados especificados dentro del bloque let



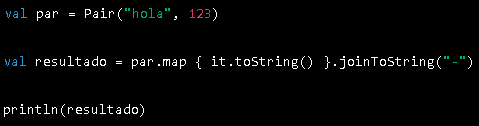
e. Recorriendo un par

Para recorrer un par, se puede utilizar una expresión lambda en una función como forEach, map, filter, entre otras.

para imprimir cada elemento del par en la consola, se puede utilizar la función forEach



Recorre un par con la función map



f. Funciones útiles para trabajar con pares en Kotlin

-first y second: Son propiedades que devuelven el primer y el segundo elemento del par, respectivamente

-toList(): Convierte el par en una lista que contiene los dos elementos.

- swap(): Devuelve un nuevo par con los elementos intercambiados.

- equals(): Compara dos pares para ver si tienen los mismos valores.

- toString(): Devuelve una cadena que representa el par.

**7. Prácticas de estructuras de datos en Kotlin**

a. Ejercicios prácticos para aplicar los conceptos aprendidos

-Listas: Crear una lista de números enteros y determinar si todos los elementos son pares

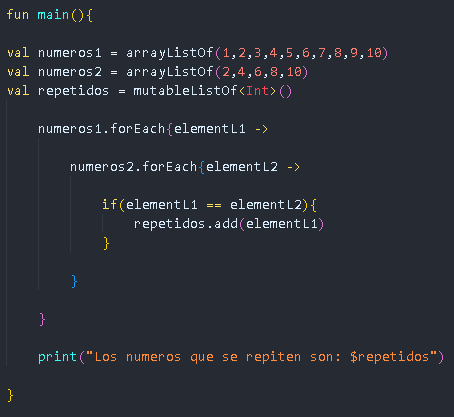
-Conjuntos: Haga un algoritmo que pida el nombre y notas de un estudiante y que cada una se guarden en un conjunto diferente, pregunte si quiere agregar cada vez mas estudiantes, y al final diga si ganan o pierden la materia

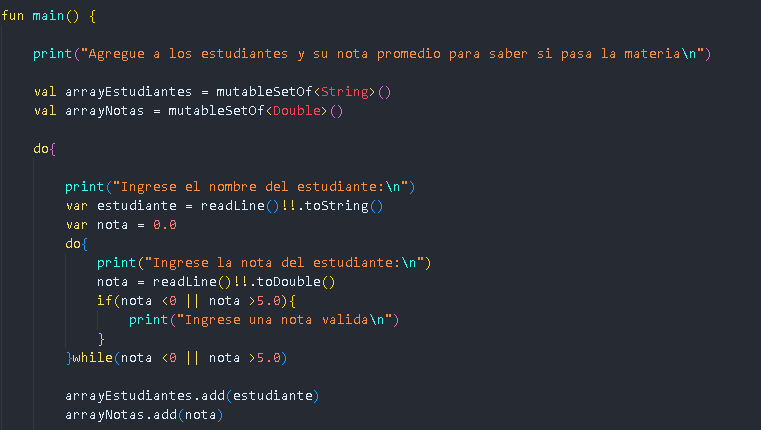
-Mapas y Arreglos: Haga un algoritmo de un cajero automático que pida un usuario y contraseña, y que cada usuario tenga un saldo en banco distinto, luego de que el cajero valide que el usuario existe en la base de datos (datos quemados) el usuario podrá retirar, consignar y consultar el dinero que tiene en el banco

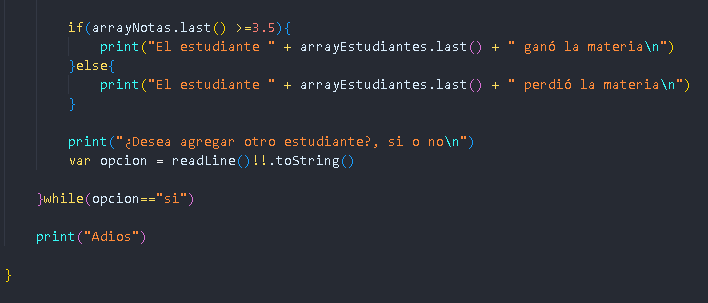
-Pares: Recorrer una lista de números enteros y en una lista guardar los números pares y en otra lista los números impares, sumar el total de los números pares e impares y guardarlos en una variable par

b. Solución a los ejercicios prácticos

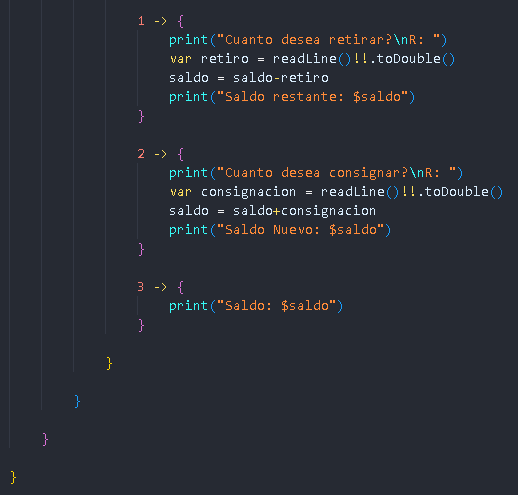
-Listas:



-Conjuntos: 



-Mapas y arreglos: 



-Pares:

