Taller Estructuras de Datos en Kotlin

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?

Las estructuras de datos son formas de organizar y almacenar datos en un programa de computadora. Se utilizan para ayudar a que los programas sean más eficientes y efectivos en el procesamiento de datos.
Las estructuras de datos se utilizan en una amplia variedad de aplicaciones informáticas, desde bases de datos y sistemas operativos hasta aplicaciones de juegos y análisis de datos. Al elegir la estructura de datos adecuada para una tarea específica, es posible lograr un mejor rendimiento, reducir la complejidad del código y hacer que un programa sea más fácil de mantener y actualizar.

b. Ventajas de utilizar estructuras de datos en Kotlin

Kotlin, al igual que otros lenguajes de programación, ofrece soporte para estructuras de datos para ayudar a los desarrolladores a organizar y procesar datos de manera eficiente. Algunas ventajas de utilizar estructuras de datos en Kotlin son:

1. Mayor eficiencia
2. Flexibilidad
3. Mejora de la calidad del código
4. Reducción de errores
5. Aumento de la productividad

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

Kotlin y Java son lenguajes de programación que comparten muchas similitudes, pero también presentan algunas diferencias en cuanto a las estructuras de datos que ofrecen.

Algunas de las diferencias entre las estructuras de datos en Kotlin y Java son las siguientes:

1. Null safety
2. Funciones de extensión
3. Sintaxis más concisa
4. Colecciones inmutables por defecto
5. Rendimiento

2. Arreglos en Kotlin
   a. ¿Qué es un arreglo?

En Kotlin, un arreglo es una estructura de datos que almacena un conjunto de elementos del mismo tipo en una secuencia indexada. Es decir, cada elemento en el arreglo tiene un índice que lo identifica de manera única dentro del arreglo.

b. Creación de arreglos en Kotlin1

```kotlin
fun main(){
//Se declara el arreglo, se le asigna un identificador, despues el "=", la funcion arrayof para colocar entre parentesis los elementos que contendra el arreglo. Una manera de crear un arreglo es la sgte, que se puede usar tanto con numero enteros como con cadenas de texto

val numeros = arrayOf(5, 2, 4, 1, 3)

//Otra forma es con la palabra Array, y dentro de los parentesis colocando el tamaño que va a tener y una funcion con los indices que va a almacenar. Abajo se pueden agregar los elementos que contendra uno por uno

val frutas = Array(4, { i -> "" })
    frutas[0]="Manzana"
    frutas[1]="Pera"
    frutas[2]="Banano"
    frutas[3]="Sandia"
}
```

c. Accediendo a los elementos de un arreglo

```
//Acceder a un elemento especifico del arreglo se usan los corchetes [] con el indice del elemento al que deseamos acceder

val tercerNumero = numeros[2]

println(tercerNumero)
}
```

d. Modificando los elementos de un arreglo

```
//Modificar un elemento primero se debe obtener y se le asigna el nuevo valor

numeros[0]=8
numeros[1]=10

println(numeros[0])
println(numeros[1])

//otra forma de modificar un elemento de un arreglo es con la funcion set poniendo la posicion del indice y lo que se va a poner en esa posicion

numeros.set(1,5)
println(numeros[1])

}
```

e. Recorriendo un arreglo

```
//Recorrer un arreglo se hace con el ciclo for, que toma un contador que va aumentando en cada posicion del arreglo y luego lo imprime

for(i in numeros){
    println(i)
}

}
```

f. Funciones útiles para trabajar con arreglos en Kotlin

1. **`size`**: devuelve el tamaño del arreglo
2. **`get`** y **`set`**: permiten obtener y establecer el valor de un elemento en una posición dada del arreglo
3. **`contains`**: comprueba si un valor dado está presente en el arreglo.
4. **`indexOf`**: devuelve el índice de la primera ocurrencia de un valor dado en el arreglo.
5. **`copyOf`**: crea una copia del arreglo.
6. **`sliceArray`**: devuelve una sublista del arreglo especificada por un rango de índices.
7. **`filter`**: devuelve una lista que contiene los elementos del arreglo que cumplen una condición dada.
8. **`sum`**: devuelve la suma de todos los elementos del arreglo.

3. Listas en Kotlin

a. ¿Qué es una lista?

En Kotlin, una lista es una colección ordenada de elementos que se pueden acceder por su índice. Se define utilizando la interfaz **List** y puede contener elementos del mismo tipo o de diferentes tipos. La lista en Kotlin es inmutable por defecto, lo que significa que no se pueden modificar los elementos de la lista después de su creación. Si se necesitan operaciones de modificación, se puede usar una lista mutable, que se define utilizando la interfaz **MutableList**

b. Creación de listas en Kotlin

```
// Crear una lista inmutable de cadenas. No se puede modificar
val listaDeCadenas = listOf("Manzana", "Naranja", "Plátano")

// Crear una lista mutable de números. Si se puede modificar
val listaDeNumeros = mutableListOf(1, 2, 3, 4, 5)
```

c. Accediendo a los elementos de una lista

```
//Para acceder a un elemento de la lista es con corchetes [] y con el metodo get y el numero del indice
println(listaDeNumeros[0])
println(listaDeNumeros.get(4))
```

d. Modificando los elementos de una lista

```
//Modificar un elemento de la lista, se pone el nombre de la lista con corchetes [] y la posicion que se va cambiar
listaDeNumeros.set(1,7)
listaDeNumeros[0]=12
```

e. Recorriendo una lista

```kotlin
//Recorrer un lista se hace con el ciclo for, que toma un contador que va aumentando en cada posicion del lista y luego lo imprime

for(i in listaDeNumeros){
    println(i)
}
println(listaDeNumeros)
}
```

f. Funciones útiles para trabajar con listas en Kotlin

1. **`size`**: devuelve el tamaño de la lista.
2. **`isEmpty`**: devuelve un booleano indicando si la lista está vacía o no.
3. **`contains`**: devuelve un booleano indicando si la lista contiene un elemento específico.
4. **`indexOf`**: devuelve el índice de la primera ocurrencia de un elemento específico en la lista, o -1 si no se encuentra.
5. **`lastIndexOf`**: devuelve el índice de la última ocurrencia de un elemento específico en la lista, o -1 si no se encuentra.
6. **`subList`**: devuelve una sublista de la lista original, especificando los índices de inicio y fin.
7. **`distinct`**: devuelve una lista que contiene sólo los elementos distintos de la lista original
8. **`sorted`**: devuelve una lista que contiene los elementos de la lista original ordenados en orden ascendente.
9. **`reversed`**: devuelve una lista que contiene los elementos de la lista original en orden inverso.
10. **`filter`**: devuelve una lista que contiene sólo los elementos que cumplen una determinada condición.

4. Conjuntos en Kotlin

a. ¿Qué es un conjunto?

En Kotlin, un conjunto (Set) es una colección de elementos sin orden y sin duplicados. Esto significa que los elementos en un conjunto no tienen una posición específica y no se pueden repetir.

Los conjuntos en Kotlin se pueden crear utilizando la función **`setOf()`** o **`mutableSetOf()`**, dependiendo de si se quiere un conjunto inmutable o mutable respectivamente.

b. Creación de conjuntos en Kotlin

```kotlin
// Conjunto inmutable de números enteros
val conjuntoA = setOf(1, 2, 3, 4, 5)

// Conjunto mutable de números enteros
val conjuntoC = mutableSetOf(1, 2, 3, 4, 5)
```

c. Accediendo a los elementos de un conjunto

```kotlin
//acceder a un elemento de conjunto
println(conjuntoA[0])
println(conjuntoB.get(3))
```

d. Modificando los elementos de un conjunto

```kotlin
//Modificar elementos de conjunto
conjuntoA.set(0,5)
conjuntoB[1]=9
```

e. Recorriendo un conjunto

```kotlin
//Recorrer los elementos de un conjunto

for(i in conjuntoA){
    println(i)
}
```

f. Funciones útiles para trabajar con conjuntos en Kotlin

1. **`size`**: devuelve el número de elementos en el conjunto.
2. **`isEmpty`**: devuelve **`true`** si el conjunto está vacío.
3. **`contains`**: devuelve **`true`** si el conjunto contiene un elemento específico.
4. **`intersect`**: devuelve un conjunto que contiene solo los elementos que están presentes en ambos conjuntos.
5. **`union`**: devuelve un conjunto que contiene todos los elementos de ambos conjuntos.
6. **`minus`**: devuelve un conjunto que contiene todos los elementos del conjunto original excepto los elementos que están presentes en otro conjunto.

5. Mapas en Kotlin

a. ¿Qué es un mapa?

En Kotlin, un mapa (Map) es una colección de pares clave-valor, donde cada clave se asigna a un valor. Los mapas se pueden usar para almacenar y recuperar datos de manera eficiente utilizando una clave en lugar de una posición como en las listas. En Kotlin, se puede crear un mapa utilizando la función **`mapOf()`** o **`mutableMapOf()`**, dependiendo de si se quiere un mapa inmutable o mutable respectivamente

b. Creación de mapas en Kotlin

```kotlin
// Mapa inmutable
val mapaA = mapOf("uno" to 1, "dos" to 2, "tres" to 3)

// Mapa mutable
val mapaB = mutableMapOf("uno" to 1, "dos" to 2, "tres" to 3)
```

c. Accediendo a los elementos de un mapa

```kotlin
//Acceder a los elementos de un mapa
mapaA["uno"]
```

d. Modificando los elementos de un mapa

```kotlin
//Modificar un elemento de un mapa se puede mediante put
mapaA.put("dos",8)
mapaB.remove("dos")
```

e. Recorriendo un mapa

```kotlin
//Recorrer un mapa se hace mediante un for

for(i in mapaA){
    println(i)
}
```

f. Funciones útiles para trabajar con mapas en Kotlin

1. **`size`**: devuelve el número de pares clave-valor en el mapa.
2. **`isEmpty`**: devuelve **`true`** si el mapa está vacío.
3. **`containsKey`**: devuelve **`true`** si el mapa contiene una clave específica.
4. **`containsValue`**: devuelve **`true`** si el mapa contiene un valor específico.
5. **`keys`**: devuelve un conjunto con todas las claves en el mapa.
6. **`values`**: devuelve una lista con todos los valores en el mapa.

6. Pares en Kotlin
    a. ¿Qué es un par?

En Kotlin, un par (Pair) es una estructura de datos que representa dos valores relacionados entre sí. Un par se puede utilizar para agrupar dos valores diferentes en un solo objeto, lo que facilita su manipulación y transmisión.
En Kotlin, se puede crear un par utilizando la función **`Pair()`**

b. Creación de pares en Kotlin

```kotlin
//Para crear pares en kotlin se usa la palabra Pair y se le ponen los dos valores o la palabra reservada to
var pair1 = Pair("Hola","Chao")

var pair2 = "Hola" to "Chao"

//se puede descomponer en dos variables asignandoles nombres entre parentesis
var(usuario, identificacion) = Pair ("Pedro", 1012)
```

c. Accediendo a los elementos de un par

```kotlin
//Acceder a un elemento par se debe usar first para acceder al primer elemento y second para acceder al segundo elemento
println(pair.first)
println(pair.second)
```

d. Modificando los elementos de un par

```kotlin
//Modificar elementos par deben estar descompuestos en distintas variables y solo llamar a la variable y asignarle un nuevo valor
usuario="Samuel"
println(usuario+identificacion)
```

e. Recorriendo un par

```kotlin
//Recorrer un par se realiza con un forEach, poniendo como los parametros el primer y segundo elemento y al final imprimirlos
pair.forEach{(first, second) ->
println("$first $second")}
```

f. Funciones útiles para trabajar con pares en Kotlin

1. **`first`** y **`second`**: Estas son propiedades del objeto par que permiten acceder a los dos valores que están almacenados en el par
2. **`to`**: Esta es una función de extensión que se puede utilizar para crear un par de valores. Esta función es equivalente a crear un objeto Pair utilizando la sintaxis convencional.
3. **`component1`** y **`component2`**: Estas son funciones de desestructuración que permiten extraer los valores de un par en dos variables separadas.
4. **`copy`**: Esta es una función que permite crear una copia del par con uno o ambos valores cambiados.
5. **`equals`** y **`hashCode`**: Estas son funciones que permiten comparar dos pares para ver si contienen los mismos valores
6. **`toString`**: Esta es una función que devuelve una cadena que representa el par. Por defecto, devuelve una cadena de la forma "(valor1, valor2)".

7. Prácticas de estructuras de datos en Kotlin
    a. Ejercicios prácticos para aplicar los conceptos aprendidos
    b. Solución a los ejercicios prácticos

```kotlin
fun main() {
    // Crear el arreglo de números
    val numeros = arrayOf(10, 4, 6, 8, 2, 5, 1, 9, 7, 3)

    // Calcular la suma de los números pares y la cantidad de números impares
    var sumaPares = 0
    var cantidadImpares = 0

    for (numero in numeros) {
        if (numero % 2 == 0) {
            sumaPares += numero
        } else {
            cantidadImpares++
        }
    }

    // Imprimir los resultados
    println("La suma de los números pares es: $sumaPares")
    println("La cantidad de números impares es: $cantidadImpares")
}
```

```kotlin
fun main() {
    // Crear la lista de palabras
    val palabras = listOf("manzana", "naranja", "pera", "banana", "kiwi", "limón", "fresa")

    // Contar las palabras según su longitud
    var largoMayor5 = 0
    var largoMenorIgual5 = 0

    for (palabra in palabras) {
        if (palabra.length > 5) {
            largoMayor5++
        } else {
            largoMenorIgual5++
        }
    }

    // Imprimir los resultados
    println("La cantidad de palabras con longitud mayor a 5 es: $largoMayor5")
    println("La cantidad de palabras con longitud menor o igual a 5 es: $largoMenorIgual5")
}
```

```kotlin
fun main() {
    // Crear el conjunto de frutas y el conjunto de verduras
    val frutas = setOf("manzana", "naranja","papaya", "pera", "banana", "kiwi", "limón", "fresa")
    val verduras = setOf("lechuga", "tomate", "cebolla", "pimiento", "calabacín", "papaya", "patata")

    // Encontrar la intersección entre los conjuntos de frutas y verduras
    val interseccion = frutas.intersect(verduras)

    // Imprimir los elementos de la intersección
    if (interseccion.isEmpty()) {
        println("No hay elementos comunes entre los conjuntos")
    } else {
        println("Los elementos comunes entre los conjuntos son:")
        for (elemento in interseccion) {
            println("- $elemento")
        }
    }
}
```

```kotlin
fun main() {
    // Crear el mapa de libros
    val libros = mapOf(
        "libro1" to mapOf(
            "titulo" to "Cien años de soledad",
            "autor" to "Gabriel García Márquez",
            "año" to 1967,
            "editorial" to "Sudamericana"
        ),
        "libro2" to mapOf(
            "titulo" to "El Principito",
            "autor" to "Antoine de Saint-Exupéry",
            "año" to 1943,
            "editorial" to "Reynal & Hitchcock"
        ),
        "libro3" to mapOf(
            "titulo" to "El Aleph",
            "autor" to "Jorge Luis Borges",
            "año" to 1949,
            "editorial" to "Sur"
        )
    )

    // Obtener la información del libro2
    val libro2 = libros["libro2"]

    // Imprimir la información del libro2
    if (libro2 != null) {
        println("Título: ${libro2["titulo"]}")
        println("Autor: ${libro2["autor"]}")
        println("Año: ${libro2["año"]}")
        println("Editorial: ${libro2["editorial"]}")
    } else {
        println("No se encontró el libro")
    }
}
```

```kotlin
fun main() {
    // Crear la lista de pares de personas
    val personas = listOf(
        Pair("Juan", 30),
        Pair("Pedro", 25),
        Pair("María", 35),
        Pair("Luis", 27)
    )

    // Ordenar la lista de pares de personas por edad
    val personasOrdenadas = personas.sortedBy { it.second }

    // Imprimir la lista de pares de personas ordenada
    personasOrdenadas.forEach { persona ->
        println("${persona.first} - ${persona.second}")
    }
}
```

- 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 practicos de cada estructura y deberá publicar el código fuente en un repositorio en GitHub.