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.

Nombre: Jackson Alexis Londoño Bonilla.

Ficha: 2469283.

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?

   R/= Las estructuras de datos son aquellas que nos proporcionan, organizar la información de forma óptima y tener una solución efectiva para un problema en determinado, por otra parte, se utilizan para trabajar almacenando información para así después acceder, modificar y manipular la misma.

   b. Ventajas de utilizar estructuras de datos en Kotlin

   R/= - **Fácil de usar:** Kotlin nos proporciona una sintaxis sencilla al la hora de trabajar con estructuras de datos , lo que facilita su manipulación de datos, es decir se pueden manipular listas, mapas, conjuntos y matrices con pocas líneas de código.

   - **Flexibilidad:** Kotlin nos brinda una amplia variedad de estructuras de datos,

   las cuales se adaptan a diferentes tipos de datos y situaciones, es decir las listas nos son útiles para almacenar datos en un orden especifico, mientras que los

mapas son útiles para almacenar datos en una estructura clave-valor.

- **Eficiencia:** Las estructuras de datos están diseñadas para ofrecer una alta eficiencia en tiempo de ejecución y en el uso de memoria. Esto quiere decir que se pueden manejar grandes cantidades de datos de manera eficiente y rápida.

- **Funciones incorporadas:** Kotlin nos da una variedad de funciones incorporadas para trabajar con estructuras de datos. Por ejemplo, podemos utilizar la función “filter” para filtrar elementos de una lista o la función “map” para transformar cada elemento de una lista en un nuevo valor.

- **Seguridad de tipo:** Kotlin es un lenguaje de programación de tipos seguros, lo cual significa que los errores de tipo se detectan en tiempo de compilación en lugar de tiempo de ejecución. Las estructuras de datos de este lenguaje aprovechan esta característica, lo cual ayuda a evitar errores en el tiempo de ejecución y mejorar la calidad del código.

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

R/= - **MutableList y List:** En Kotlin, hay 2 tipos de listas, “List” e “MutableList”. “List” es inmutable, lo que quiere decir que no se puede modificar después de ser creada. “MutableList”, por otra parte, es mutable y se puede agregar o elementos. En cambio en Java las listas son siempre mutables.

- **Set:** El tipo “Set” en Kotlin representa un conjunto inmutable, mientras que “MutableSet” representa un conjunto mutable. En Java, no existe distinción entre conjuntos inmutables y mutables.

- **Map:** En Kotlin, “Map” es una interfaz que define un mapa inmutable, mientras que “MutableMap” representa un mapa que es mutable. En cambio en Java “Map” es mutable.

- **Data classes:** Kotlin brinda una estructura de datos conveniente llamada “data class”, que nos permite definir clases que contienen solo datos sin comportamiento adicional: En Java, no hay estructura de datos similar.

- **Nullability:** Kotlin tiene un sistema de tipos que permite la diferenciación entre valores nulos y no nulos. Esto se refleja en sus estructuras de datos, lo que quiere decir que se puede tener una lista que no permita valores nulos, por ejemplo. En

Java, todos los tipos son nulos por defecto a menos que se utilice la anotación “@NonNull”.

- **Rangos:** Kotlin tiene un tipo de datos llamado ‘rango’, que se puede utilizar para iterar sobre un conjunto de valores dentro de un rango especifico. Java no tiene una estructura de datos similar.

- **Tipos de datos:** Kotlin tiene tipos de datos mas expresivos que Java, como el tipo de dato ‘enum class’, que nos permite definir conjuntos de constantes con un comportamiento especifico, o el tipo de datos ‘data class’, que proporciona implementaciones estándar de `toString()`, `equals()`, `hashCode()`, y `copy()`.

2. Arreglos en Kotlin

   a. ¿Qué es un arreglo?

   R/= Un arreglo es una estructura de datos que permite almacenar una colección de elementos del mismo tipo, y se puede acceder a los elementos mediante su índice y manipularlos utilizando una variedad de funciones.

   b. Creación de arreglos en Kotlin

   R/= Los arreglos se pueden declarar utilizando la palabra clave `arrayOf` seguida de una lista de valores separados por comas entre corchetes. Por ejemplo, en el siguiente declaramos un arreglo de enteros.

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

   c. Accediendo a los elementos de un arreglo

   R/= A los elementos de un arreglo se puede acceder mediante su índice que comienza en 0, por ejemplo, para acceder al segundo numero del arreglo “numeros”, se puede utilizar la siguiente forma:

```
val segundoNumero = numeros[1] // el resultado será 2
```

d. Modificando los elementos de un arreglo

R/= Los elementos de un arreglo se pueden modificar utilizando la sintaxis de asignación. Para modificar a un elemento en especifico, se debe utilizar el índice del elemento y asignar el nuevo valor en el índice dado. Por ejemplo:

```
val numeros = arrayOf(1,2,3,4,5)
numeros[1] = 9
```

En el ejemplo anterior, el valor del segundo elemento del arreglo "numeros" se cambia de 2 a 9.

e. Recorriendo un arreglo

R/= Para recorrer un arreglo en Kotlin, podremos usar un bucle “for” que itere a través de los elementos del arreglo. Esto también puede hacerse de otras maneras, lo que significa que depende sus necesidades y preferencias personales.

A continuación lo hare con algunas de las formas en las que se puede recorrer:

- De forma tradicional mediante un bucle `for`:

```
val arreglo = arrayOf("Pepe","Juan","Jose")

for (i in 0 until arreglo.size){
  println(arreglo[i])
}
```

Este código anterior imprimirá cada elemento del arreglo en la consola.

- Usando el método `forEach`:

```
val arreglo = arrayOf("elemento 1","elemento 2","elemento 3")

arreglo.forEach {
```

```
  println(it)
}
```

Este código imprimirá cada elemento del arreglo en la consola utilizando el método `forEach`.

- Por medio del operador `in`:

```
val arreglo = arrayOf("Julian","Carlos","Maria")

for (elemento in arreglo){
  println(elemento)
}
```

Este código imprimirá cada elemento del arreglo en consola haciendo uso del operador `in`.

f. Funciones útiles para trabajar con arreglos en Kotlin

R/= A la hora de trabajar con arreglos, Kotlin tiene una gran cantidad de funciones útiles para trabajar con ellos. A continuación hare uso de algunas de las mas comunes:

- `size` : Nos devuelve la cantidad de elementos en el arreglo.

  Ejemplo:

```
val array = arrayOf(1,2,3,4,5)
val size = array.size // El resultado es 5
```

- `get`: Nos devuelve el elemento en la posición especificada.

  Ejemplo:

```
val array = arrayOf(1, 2, 3, 4, 5)
val element = array.get(2) // El resultado es 3
```

- `set`: Este asigna un valor a la posición que se especifique.

  Ejemplo:

```
val array = arrayOf(1, 2, 3, 4, 5)
array.set(2, 10) // Cambia el valor de la posición 2 a 10
```

- `map`: Crea un nuevo arreglo aplicando una función a cada elemento.

  Ejemplo:

```
val array = arrayOf(1,2,3,4,5)
val mappedArray = array.map { it * 2 } // [2, 4, 6, 8, 10]
```

- `sorted`: Crea un nuevo arreglo ordenando los elementos en orden ascendente.

  Ejemplo:

```
val array = arrayOf(5,2,3,1,4)
val sortedArray = array.sorted() // [1, 2, 3, 4, 5]
```

3. Listas en Kotlin

   a. ¿Qué es una lista?

   R/= Una lista es una estructura de datos que permite almacenar un conjunto ordenado de elementos del mismo tipo. A los elementos de una lista se puede acceder por su posición, y se puede agregar, eliminar o modificar elementos.

b. Creación de listas en Kotlin

R/= Para crear una lista en Kotlin, se puede utilizar la función `listOf()`, que toma una serie de elementos y los agrega a la lista:

```
val numeros = listOf(1,2,3,4,5)
val palabras = listOf("Juan","Jose")
```

c. Accediendo a los elementos de una lista

R/= Para acceder a los elementos de una lista en Kotlin, se puede utilizar el operador de indexación `[]` con el índice del elemento que se desea obtener. Los índices en Kotlin comienzan en cero, lo que significa que el primer elemento de una lista tiene un índice de cero.

```
val numeros = listOf(1,2,3,4,5)
```

Para acceder al primer elemento que es (1) se hace de la siguiente manera:

```
val primerElemento = numeros[0]  // primerElemento = 1
```

d. Modificando los elementos de una lista

R/= Para poder modificar los elementos de una lista en Kotlin, es posible si la lista es mutable. Para modificar un elemento en una lista mutable, se puede acceder al índice del elemento y actualizar su valor.

Ejemplo:

```
// Se crea la lista mutable
val listaMutable = mutableListOf("uno","dos","tres","cuatro")

// Se imprime la lista original
println("Lista original: $listaMutable")
```

```
// Se modifica el segundo elemento
listaMutable[1] = "nuevo valor"

// Se imprime la lista modificada
println("Lista modificada: $listaMutable")
```

e. Recorriendo una lista

R/=Para recorrer una lista en Kotlin, existen diferentes métodos cada uno según lo que se necesite hacer, algunos de ellos son:

- Por medio de un ciclo `for`:

```
val lista = listOf(1,2,3,4,5)

for (elemento in lista) {
    println(elemento)
}
```

- Por medio del método `forEach`:

```
val lista = listOf(1,2,3,4,5)

lista.forEach { elemento ->
    println(elemento)
}
```

- Utilizando el método `forEachIndexed`, si se necesita acceder tanto al índice como a el elemento:

```
val lista = listOf(1,2,3,4,5)

lista.forEachIndexed { indice, elemento ->
    println("Índice: $indice, Elemento: $elemento")
}
```

f. Funciones útiles para trabajar con listas en Kotlin

R/= Kotlin proporciona muchas funciones de gran utilidad para trabajar con listas, algunas de las mas comunes son:

- `forEach`: Esta función permite iterar sobre cada elemento de una lista y realizar una acción para cada una de ellos.

```
val list = listOf("a","b","c")
list.forEach { println(it) }
```

- `map`: Esta función transforma cada elemento de una lista en un nuevo elemento, y devuelve una nueva lista con los elementos transformados.

```
val list = listOf(1,2,3)
val doubledList = list.map {it * 2}
println(doubledList) // resultado = [2, 4, 6]
```

- `filter`: Esta función devuelve una nueva lista que contiene solo los elementos que cumplen con una condición especificada.

```
val list = listOf(1,2,3,4,5)
val evenList = list.filter { it % 2 == 0 }
println(evenList) // resultado = [2, 4]
```

- `reduce:` Esta función combina todos los elementos de una lista en un único valor, utilizando una función que toma dos elementos y devuelve un valor.

```
val list = listOf(1,2,3,4,5)
val sum = list.reduce { acc, i -> acc + i }
println(sum) // resultado = 15
```

4. Conjuntos en Kotlin

   a. ¿Qué es un conjunto?

   R/= En Kotlin, un conjunto (Set) es una colección de elementos que son únicos, es decir, no puede contener elementos que sean duplicados. Al igual que las listas, los conjuntos son mutables y nos permiten agregar y eliminar los elementos. Por otro lado, a diferencia de lo que son las listas, los conjuntos no tienen un orden definido, lo que significa que no se puede acceder a los elementos por índice.

   b. Creación de conjuntos en Kotlin

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

   Por ejemplo:

```
// Crear un conjunto inmutable
val set1 = setOf("a","b","c")

// Crear un conjunto mutable
val set2 = mutableSetOf(1, 2, 3)

// Agregar elementos a un conjunto mutable
set2.add(4)
set2.add(5)

// Eliminar elementos de un conjunto mutable
set2.remove(3)
```

   c. Accediendo a los elementos de un conjunto

   R/= Anteriormente dicho, los conjuntos se representan mediante la interfaz `Set`. Para acceder a los elementos de un conjunto podemos hacerlo de varias maneras:

   - **Usando el operador `in` :** El operador `in` es usado para verificar si un elemento esta en el conjunto. Por ejemplo:

```
val mySet = setOf("h","k","o")
println("h" in mySet) // Esto devolvera: True
println("d" in mySet) // Esto devolvera: False
```

- **Iterando sobre los elementos del conjunto:** Podemos usar un bucle `for` o la función `forEach` para iterar sobre los elementos del conjunto. Por ejemplo:

```
val mySet = setOf("Henry","Cristian","Edwin")
for (element in mySet) {
    println(element)
}

// Usando - forEach
mySet.forEach { element ->
    println(element)
}
```

- **Otra manera es convirtiendo el conjunto en una lista:** Podemos convertir el conjunto en una lista y acceder a los elementos por su índice. Sin embargo, hay que tener en cuenta que los conjuntos no tienen un orden definido, por lo que no es garantizado que los elementos se almacenen en el mismo orden cada vez. Por ejemplo:

```
val mySet = setOf("Jimena","Luis","Camila")
val myList = mySet.toList()
println(myList[0]) // El resultado sera: "Jimena"
```

d. Modificando los elementos de un conjunto

R/= En Kotlin, los conjuntos inmutables `Set` no pueden ser modificados una vez que ya están creados. Si necesitamos modificar los elementos de un conjunto, debemos crear un conjunto mutable utilizando la clase `MutableSet`.

Para modificar un conjunto mutable, podemos utilizar los siguientes métodos:

- `add(element: E)`: Agregara un elemento a el conjunto.

```
val myMutableSet = mutableSetOf("l","m","n")
myMutableSet.add("a")
println(myMutableSet) // ["l","m","n","a"]
```

- `addAll(elements: Collection<E>)`: Agregara todos los elementos de una colección al conjunto.

```
val myMutableSet = mutableSetOf("a","b","c")
val newElements = listOf("f","z")
myMutableSet.addAll(newElements)
println(myMutableSet) // ["a","b","c","f","z"]
```

- `remove(element: E)`: Elimina un elemento del conjunto.

```
val myMutableSet = mutableSetOf("Pepe","Juan","Pedro")
myMutableSet.remove("Juan")
println(myMutableSet) // ["Pepe","Pedro"]
```

- `removeAll(elements: Collection<E>)`: Elimina todos los elementos de una colección del conjunto.

```
val myMutableSet = mutableSetOf("Lina","Luisa","Jorge")
val elementsToRemove = listOf("Luisa","Jorge")
myMutableSet.removeAll(elementsToRemove)
println(myMutableSet) // ["Lina"]
```

- `retainAll(elements: Collection<E>)`: Elimina todos los elementos del conjunto que no están en una colección dada.

```
val myMutableSet = mutableSetOf("Camilo","Eimy", "Camila")
val elementsToRetain = listOf("Eimy", "Camila")
myMutableSet.retainAll(elementsToRetain)
println(myMutableSet) // ["Eimy", "Camila"]
```

e. Recorriendo un conjunto

R/= Para recorrer un conjunto `Set` en Kotlin, podemos utilizar un bucle `for` o `forEach`.

Aquí un ejemplo de como recorrer un conjunto de enteros utilizando un bucle `for`:

```
val conjuntoEnteros = setOf(1,2,3,4,5)

for (entero in conjuntoEnteros) {
    println(entero)
}
```

El código anterior imprime los números del 1 al 5 en consola.

También podemos recorrer un conjunto utilizando el método `forEach`. En el siguiente ejemplo se mostrara como imprimir cada elemento del conjunto utilizando el método `forEach`:

```
val conjuntoEnteros = setOf(1, 2, 3, 4, 5)

conjuntoEnteros.forEach { entero ->
    println(entero)
}
```

El código anterior al igual que el anterior imprime los números del 1 al 5 en la consola. La única diferencia es que hacemos uso de la sintaxis de lambda para definir una función que se ejecutara para cada elemento del conjunto.

f. Funciones útiles para trabajar con conjuntos en Kotlin

R/= En Kotlin existen varias funciones a la hora de trabajar con conjuntos.

Algunas de ellas son:

- `setOf`: Es una función utilizada para crear un conjunto inmutable. Toma como parámetros una lista de elementos y devuelve un conjunto que contiene los mismos elementos.

- `mutableSetOf()`: Es una función que se utiliza para crear un conjunto mutable. Toma una lista de elementos como parámetros y devuelve un conjunto que contiene los mismos elementos.

- `intersect()`: Es una función que se utiliza para obtener la intersección entre dos conjuntos.

- `union()`: Es una función que se utiliza para obtener la unión entre dos conjuntos.

- `subtract()`: Es una función que se utiliza para obtener la diferencia entre dos conjuntos.

- `sorted()`: Es una función que se utiliza para obtener un conjunto ordenado en orden ascendente.

- `sortedDescending()`: Es una función que se utiliza para obtener un conjunto ordenado en orden descendente.

- `contains()`: Esta función se utiliza para comprobar si un valor se encuentra dentro de una colección, como una lista, un conjunto, un mapa, entre otros.

- `remove()`: Esta función es utilizada para eliminar un elemento en especifico de una colección, lista o conjunto mutable.

Las anteriores funciones son algunas de las muchas funciones para trabajar con conjuntos en Kotlin.

5. Mapas en Kotlin

   a. ¿Qué es un mapa?

   R/= Un mapa en Kotlin es una estructura de datos que almacena pares de clave-valor.

   La clave es un objeto único que se utiliza para identificar y acceder al valor asociado en el mapa. Los valores pueden ser cualquier objeto, incluyendo tipos primitivos como números y cadenas de texto, así como objetos personalizados.

   b. Creación de mapas en Kotlin

   R/= Para crear mapas en Kotlin, podemos utilizar la interfaz `Map`. Hay 2 tipos de implementaciones de `Map` en Kotlin: `MutableMap` y `Map.MutableMap` nos permite agregar, actualizar y eliminar elementos del mapa, mientras que `Map` es solo de lectura y no nos permitira hacer cambios en el mapa.

   Para crear un mapa en Kotlin, podemos usar la función `mapOf` o `mutableMapOf` .

   La función `mapOf` nos devuelve un mapa de solo lectura, mientras que `mutableMapOf` devuelve un mapa que podemos modificar.

   Por ejemplo:

```
// creando un mapa de solo lectura con mapOf
val map = mapOf("a" to 1, "b" to 2, "c" to 3)

// creando un mapa mutable con mutableMapOf
val mutableMap = mutableMapOf("a" to 1, "b" to 2, "c" to 3)

// agregando elementos al mapa mutable
mutableMap["d"] = 4
mutableMap["e"] = 5

// imprimiendo el mapa mutable
println(mutableMap)
```

```
// Resultado
{a=1, b=2, c=3, d=4, e=5}
```

c. Accediendo a los elementos de un mapa

R/= Para acceder a los elementos de un mapa en Kotlin, podemos utilizar la sintaxis de corchetes `[]` y pasar la clave del elemento al que queremos acceder.

Un ejemplo seria:

```
val map = mapOf("a" to 1, "b" to 2, "c" to 3)

// accediendo al valor de un elemento usando su clave
val valueOfA = map["a"] // devuelve 1
val valueOfB = map["b"] // devuelve 2

// accediendo a un elemento que no existe devuelve null
val valueOfD = map["d"] // devuelve null
```

Si estamos trabajando con un MutableMap, podemos utilizar la misma sintaxis para acceder a los elementos.

```
val mutableMap = mutableMapOf("a" to 1, "b" to 2, "c" to 3)

// accediendo al valor de un elemento usando su clave
val valueOfA = mutableMap["a"] // devuelve 1
```

d. Modificando los elementos de un mapa

R/=Para modificar los elementos de un mapa en Kotlin, podemos utilizar la sintaxis de corchetes `[]` y pasar la clave del elemento que deseamos modificar, y luego asignarle un nuevo valor.

Aquí hay un ejemplo de cómo modificar los elementos de un mapa en Kotlin:

```
val mutableMap = mutableMapOf("a" to 1, "b" to 2, "c" to 3)
```

```
// modificando el valor de un elemento usando su clave
mutableMap["a"] = 4 // actualiza el valor del elemento "a" a 4

// agregando un nuevo elemento al mapa
mutableMap["d"] = 5 // agrega un nuevo elemento con clave "d" y valor 5

// eliminando un elemento del mapa usando su clave
mutableMap.remove("b") // elimina el elemento con clave "b"
```

e. Recorriendo un mapa

R/= Para recorrer un mapa en Kotlin, podemos usar cualquiera de las siguientes formas:

- Usando un bucle `for` para iterar sobre las entradas del mapa:

```
val map = mapOf("a" to 1, "b" to 2, "c" to 3)

for ((key, value) in map) {
    println("$key = $value")
}
```

- Usando el método forEach para iterar sobre las entradas del mapa:

```
val map = mapOf("a" to 1, "b" to 2, "c" to 3)

map.forEach { (key, value) ->
    println("$key = $value")
}
```

- Iterando sobre las claves del mapa y obteniendo los valores correspondientes:

```
val map = mapOf("a" to 1, "b" to 2, "c" to 3)

for (key in map.keys) {
    val value = map[key]
```

```
    println("$key = $value")
}
```

Hay que tener en cuenta que los mapas en Kotlin son inmutables por defecto, por lo que no es posible modificar las entradas del mapa durante el recorrido. Si necesitamos modificar el mapa durante el recorrido, deberemos utilizar un mapa mutable en su lugar.

f. Funciones útiles para trabajar con mapas en Kotlin

R/= Para trabajar con mapas en Kotlin, hay varias funciones, algunas son:

- `mapOf`: Esta función crea un mapa inmutable a partir de pares clave-valor.

  Ejemplo:

  ```
  val map = mapOf("a" to 1, "b" to 2, "c" to 3)
  ```

- `mutableMapOf`: Esta función crea un mapa mutable a partir depares clave-valor.

  Ejemplo:

  ```
  val map = mutableMapOf("a" to 1, "b" to 2, "c" to 3)
  ```

- `get`: Esta función agrega un par clave-valor a un mapa mutable o actualiza el valor correspondiente si la clave ya existe.

  ```
  val map = mapOf("a" to 1, "b" to 2, "c" to 3)
  val value = map["a"] // devuelve 1
  ```

- `put`: Esta función agrega un par clave-valor a un mapa mutable o actualiza el valor correspondiente si la clave ya existe.

```
val map = mutableMapOf("a" to 1, "b" to 2, "c" to 3)
map.put("d", 4) // agrega el par clave-valor ("d", 4) al mapa
map.put("b", 5) // actualiza el valor correspondiente a la clave "b" a 5
```

- `keys()`: Esta función se utiliza para obtener una lista de las claves del mapa.

```
val mutableMap = mutableMapOf(1 to "uno", 2 to "dos", 3 to "tres")
val keys = mutableMap.keys
```

6. Pares en Kotlin

   a. ¿Qué es un par?

   R/=En Kotlin, un par es un objeto que representa una tupla de dos valores, es decir, un par de valores. La clase para representar un par se llama `Pair`.

   b. Creación de pares en Kotlin

   R/= Un par se puede crear utilizando la función `Pair()` o el operador `to`. Por ejemplo, el siguiente código crea un par que contiene una cadena y un entero:

```
val myPair = Pair("Pablo", 42)
```

   o también, utilizando el operador `to`:

```
val myPair = "Pedro" to 42
```

c. Accediendo a los elementos de un par

R/= A los elementos de un par se puede acceder utilizando las propiedades “first” y “second”.

Por ejemplo:

```
val myPair = Pair("Cristian", 42)
val myString = myPair.first // myString tiene el valor de "Cristian"
val myInt = myPair.second // myInt tiene el valor de 42
```

d. Modificando los elementos de un par

R/= En Kotlin los pares son inmutables, lo que significa que no se pueden modificar una vez creados. Sin embargo, es posible crear un nuevo par con los mismos valores pero con uno o ambos valores actualizados.

Para crear un nuevo par con un valor actualizado, se puede utilizar la función `copy()` y especificar el nuevo valor como argumento. Por ejemplo:

```
val myPair = Pair("Arle", 42)
val myNewPair = myPair.copy(second = 84)
```

En este ejemplo, `myPair` es un par con los valores "Hola" y 42, y `myNewPair` es un nuevo par con el valor de `second` actualizado a 84.

e. Recorriendo un par

R/= Un par en Kotlin, se puede recorrer mediante la función `forEach()` o mediante la destructuracion de valores.

- Utilizando forEach, se puede hacer lo siguiente:

```
val myPair = Pair("Luis", 42)
myPair.forEach { println(it) }
```

En este ejemplo, se imprime cada uno de los valores del par utilizando la función `forEach()`.

- También es posible desestructurar los valores del par en variables individuales para acceder a ellos directamente. Por ejemplo:

```
val myPair = Pair("Juan", 42)
val (myString, myInt) = myPair
println(myString) // imprime "Juan"
println(myInt) // imprime "42"
```

En este ejemplo, se desestructura el par en dos variables, `myString` y `myInt`, para acceder directamente a los valores del par.

- Por medio de un bucle `for`, el cual recorrerá cada valor del par y luego convierte el par a una lista utilizando la función `toList()`, que devuelve una lista con los elementos del par.

  Por ejemplo:

```
val par = Pair("Hola", 123)

for (valor in par.toList()) {
    println(valor)
}
```

f. Funciones útiles para trabajar con pares en Kotlin

R/= Kotlin nos brinda varias funciones útiles para trabajar con pares. A continuación, se mostraran algunas de ellas:

- `toList()`: esta función convierte un par en una lista que contiene sus dos valores. Por ejemplo:

```
val pair = Pair("Pipe", 123)
val list = pair.toList() // [ "Pipe", 123 ]
```

En este ejemplo, se intercambian los valores del par utilizando la función `swap()` y se crea un nuevo par con los valores intercambiados.

- `equals()`: Esta función se utiliza para comparar dos pares y determinar si tienen los mismos valores. Por ejemplo:

```
val myPair1 = Pair("Pepe", 42)
val myPair2 = Pair("Pepe", 42)
val areEqual = myPair1 == myPair2
```

En este ejemplo, se comparan dos pares utilizando la función `equals()` y se asigna el resultado a la variable `areEqual`.

- `hashCode()`: Esta función se utiliza para obtener el código hash de un par. El código hash es utilizado para identificar el objeto de manera única en las colecciones. Por ejemplo:

```
val myPair = Pair("Pepe", 42)
val myHashCode = myPair.hashCode()
```

En este ejemplo, se obtiene el código hash del par utilizando la función `hashCode()` y se asigna el resultado a la variable `myHashCode.`

7. Prácticas de estructuras de datos en Kotlin

a. Ejercicios prácticos para aplicar los conceptos aprendidos

R/=

- **Arreglos:** Cree un arreglo de 8 números enteros, después modifique uno de los elementos por 8, luego recorra dicho arreglo y multiplique cada elemento.
- **Listas:** Cree una lista mutable con 7 palabras, luego modifique 2 elementos de la lista, posteriormente recorra la lista e imprima el índice y el elemento, por ultimo ordene la lista de forma ascendente.
- **Conjuntos:** Cree 2 conjuntos mutables de 4 números, después elimine 1 elemento del primer conjunto y otro del segundo conjunto , luego haga una unión con los 2 conjuntos, finalmente haga uso de una función para cualquiera de los 2 conjuntos.
- **Mapas:** Cree un mapa mutable con 5 elementos, luego actualice el valor de 2 elementos, después elimine 1 elemento, también recorra el mismo y por ultimo usando una función imprima las claves del mapa.
- **Pares:** Cree 2 pares, luego sume los valores de los 2 pares por separado, después recorra el primer par, finalmente convierta el segundo par a una lista por medio de una función.

b. Solución a los ejercicios prácticos

R/=

- **Arreglos:**

```
fun main(){
    val arregloNumeros:IntArray = intArrayOf(5,7,3,4,2,5,9)

    arregloNumeros[2]=8

    println(arregloNumeros.contentToString())

    var producto = 1

    for (elemento in arregloNumeros) {
        producto *= elemento
    }
    println("El producto es: $producto")
}
```

- **Listas:**

```
fun main(){
    val palabras = mutableListOf("g", "f", "a", "l", "k", "b", "p")

    palabras[2]="z"
    palabras[3]="v"

    palabras.forEachIndexed { indice, elemento ->
        println("Indice: $indice, Elemento: $elemento")
    }

    val listaOrdenada = palabras.sorted()
    println(listaOrdenada)
}
```

- **Conjuntos:**

```
fun main(){
    val conjunto1 = mutableSetOf(1,2,3,4)
    val conjunto2 = mutableSetOf(5,6,7,8)

    conjunto1.remove(4)
    conjunto2.remove(8)

    println(conjunto1)
    println(conjunto2)

    val unionDeConjuntos = conjunto1.union(conjunto2)
    println(unionDeConjuntos)

    if(conjunto1.contains(3)){
        println("El conjunto 1 contiene el numero 3")
    }
}
```

- **Mapas:**

```
fun main(){
    val mapa = mutableMapOf("a" to 1, "b" to 2, "c" to 3, "d" to 4, "e" to 5)

    mapa["a"] = 8
    mapa["c"] = 7
```

```
    mapa.remove("b")

    println(mapa)

    mapa.forEach { (clave, valor) ->
        println("$clave = $valor")
    }

    val claves = mapa.keys
    println(claves)
}
```

- **Pares:**

```
fun main() {
    val par1 = Pair(1, 4)
    val par2 = Pair(2, 8)

    val sumaPar1 = par1.first + par1.second
    val sumaPar2 = par2.first + par2.second

    println("La suma del primer par es: $sumaPar1")
    println("La suma del segundo par es: $sumaPar2")

    for (element in par1.toList()) {
        println(element)
    }

    val lista = par2.toList()
    println(lista)
}
```

Código fuente - GitHub:

https://github.com/Alexxis4ever/Exercises_Kotlin.git

Recursos adicionales:

- 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.