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 forma de organizar los datos en la computadora, de tal manera que nos permita realizar unas operaciones con ellas de forma muy eficiente.

b. Ventajas de utilizar estructuras de datos en Kotlin

Las estructuras de datos son medios para manejar grandes cantidades de información de manera fácil y nos permiten, como desarrolladores, organizar la información de manera eficiente. Además, nos permiten hacer un software más eficiente optimizando recursos

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

En kotlin las estructuras de datos se pueden crear sin definir el tipo de datos que tendrá, en cambio en Java no

2. Arreglos en Kotlin

a. ¿Qué es un arreglo?

Un arreglo (matriz) es una **colección ordenada de datos** (tanto primitivos u objetos dependiendo del lenguaje).

b. Creación de arreglos en Kotlin

kotlin

```
 val arreglo = arrayOf(1, 5, 7, 3)
```

java

```
String arreglo [ ]=new String [5]
```

c. Accediendo a los elementos de un arreglo

kotlin

```
println(arreglo[0])
```

java

```
println(arreglo[0])
```

d. Modificando los elementos de un arreglo

Kotlin

```
arreglo[0]=”hola”
```

JAVA

```
arreglo[0]=”hola”
```

e. Recorriendo un arreglo

kotlin

```
for( i in 0  until arreglo.size){
print(i)
}
```

java

```
for(int i=0;i<arreglo.lenght;i++){
system.out.println(arreglo[i])
}
```

f. Funciones útiles para trabajar con arreglos en Kotlin
size:devuelve el tamaño del arreglo
slice:devuelve un subconjunto del arreglo
contains:devuelve si contiene un elemento
index:devuelve el índice
filter:devuelve otro arreglo con unas condiciones dadas

3. Listas en Kotlin
a. ¿Qué es una lista?
En codelabs anteriores, aprendiste los tipos de datos básicos en Kotlin, como `Int`, `Double`, `Boolean` y `String`. Estos te permiten almacenar un determinado tipo de valor dentro de una variable. Pero ¿qué sucede si deseas almacenar más de un valor? En este caso, veremos que resulta muy útil tener un tipo de datos de `List`.
b.Creación de listas en Kotlin
Kotlin
val lista= mutableListOf()
Java
ArrayList lista=new Arraylist()
c. Accediendo a los elementos de una lista
kotlin
lista.get(2)
java
lista.get(2)
d. Modificando los elementos de una lista
kotlin
lista.set(2,”juan”)
java
lista.set(3,”juan”)

e. Recorriendo una lista
kotlin
for(i in lista){
print(i)
}
java
for(int i=0;i<lista.size;i++){
system.out.println(lista[i])
}
f. Funciones útiles para trabajar con listas en Kotlin
size:devuelve el tamaño del arreglo
subList:devuelve un subconjunto del arreglo
contains:devuelve si contiene un elemento
index:devuelve el índice

filter:devuelve otro arreglo con unas condiciones dadas

4. Conjuntos en Kotlin

a. ¿Qué es un conjunto?

es una colección sin orden de elementos únicos, esto es, no puede tener ningún duplicado.

b. Creación de conjuntos en Kotlin

Kotlin

toMutableSet() también es útil para modificar un conjunto

```
val dias= setOf("lunes", "martes", "miercoles")
```

Java

```
HashSet dias=new HashSet()
```

c. Accediendo a los elementos de un conjunto

Kotlin

```
print(dias)
```

javab

```
print(dias)
```

d. Modificando los elementos de un conjunto

kotlin:

```
val dias = setOf(“lunes”, “martes”, “miercoles”)
val newDias = dias.map { if (it == 2) 4 else it }
```

java:

```
Set<Integer> dias = new HashSet<>(Arrays.asList(“lunes”, “martes”, “miercoles”));
set.remove(“martes”);
set.add(“viernes”);
```

e. Recorriendo un conjunto

kotlin

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

java

```
for(object dia:dias){
system.out.println(dias)
}
```

f. Funciones útiles para trabajar con conjuntos en Kotlin

size:devuelve el tamaño del conjunto

contains:verifica si un elemento se encuentra en el conjunto

union:une dos conjuntos

intersect:devuelve un conjunto solo con los elementos que coincidan en dos

subtract:devuelve un conjunto solo con los elementos que están en el primer conjunto y no en el segundo

map:devuelve un conjunto después de aplicar una operación

5. Mapas en Kotlin

a. ¿Qué es un mapa?

La opción de map en Kotlin se define como una **herramienta del sistema que funciona como colección e incluye pares de objeto clave y valor**, al tiempo que admite que se lleven a cabo los procesos que admiten la recuperación de forma eficiente del valor que corresponde a cada clave.

b. Creación de mapas en Kotlin

kotlin

val materias = mutableMapOf(“matematicas” to “kebyn”, “religion” to “tulio”)+

java

hashMap <Integer,String>materias=new HashMap <Integer,String>()

c. Accediendo a los elementos de un mapa get()

kotlin

println(“llaves ${materias.keys})

println(“valores ${materias.values})

println(“todo $materias)

java

System.out.println(materias.keyset())

System.out.println(materias.values())

System.out.println(materias)

d. Modificando los elementos de un mapa

kotlin

materias[“matematicas”]= “ramiro”

java

materias.replace(“matematicas”, “ramiro”)

e. Recorriendo un mapa

kotlin

```
 for ((clave, valor) in materias) {
 println("$clave = $valor")
}
```

java

Iterator<integer>Iterator=materias.keyset().iterator()

while(iterator.hasNext()){

Integer lave=iterator.next()

sistem.out.println(llave+”-”+materias.get(llave)}

f. Funciones útiles para trabajar con mapas en Kotlin

get:recuperar el valor de una clave

remove:eliminar un valor por medio de su clave

6. Pares en Kotlin

a. ¿Qué es un par?

En Kotlin, un "par" es una estructura de datos que representa una colección de dos elementos ordenados. Se puede pensar en un par como un objeto que contiene dos valores, que pueden ser de cualquier tipo, incluyendo tipos de datos primitivos y clases.

b. Creación de pares en Kotlin

kotlin

```
val num = Pair("Hello", 42)
```

java

```
Pair<String, Integer> num = new Pair<>("Hello", 42);
```

c. Accediendo a los elementos de un par

kotlin

```
pair.first
pair.second
```

java

```
pair.first
pair.second
```

d. Modificando los elementos de un par

par modificar un pair se debe crear otro que guarde lo del anterior par y agregar los cambios nuevos

kotlin

```
val newNum = Num(num.first + 1, num.second - 1)
```

java

```
Pair<Integer, Integer> newNum= new Pair<>(num.first + 1, num.second - 1);
```

e. Recorriendo un par

```
for ((a, b) in num) {
   println("$a, $b")
}
```

f. Funciones útiles para trabajar con pares en Kotlin

first y second: estas son propiedades de solo lectura que devuelven el primer y segundo elemento del par, respectivamente.

copy(): esta función devuelve una copia del par original con los mismos valores de first y second.

toString(): esta función devuelve una cadena que representa el par, en el formato "(first, second)".

equals(): esta función se utiliza para comparar dos pares. Devuelve true si los valores de first y second son iguales en ambos pares.

7. Prácticas de estructuras de datos en Kotlin

a. Ejercicios prácticos para aplicar los conceptos aprendidos

```
fun main(){
    val numbers=FloatArray(3)
    var promedio=0.0



    for( i in 0 until numbers.size){
       println("ingrese la nota ${i+1}:")
       numbers[i]=readLine()!!.toFloat()
    }

    for(i in 0 until numbers.size){
         promedio +=numbers[i]

    }
    promedio=promedio/numbers.size
    if(promedio>=3.5){
         println("gano la materia")
    }else{
         println("perdio la materia")
    }
    println("nota final: $promedio")

}
```

listas

```
fun main(){

var notas=mutableListOf<Float>();

for(i in 0 until 3 ) {
    println("ingrese la nota ${i+1}")
    var nota=readLine()!!.toFloat()
    notas.add(nota)

}
println(notas.size)
val menu="""
1.calcular promedio
2.mostrar si gano o perdio
3.volver a ingresar notas
```

```
4.salir
"""
var opcion=0
do{
println(menu)
opcion=readLine()!!.toInt()
when(opcion){
    1->{
        var promedio=0.0;
        for (i in notas) {
        promedio+=i
        }
        promedio=promedio/notas.size
         println("el promedio de las notas  es $promedio")
    }
    2->{
         var promedio=0.0;
        for (i in notas) {
        promedio+=i
        }
        promedio=promedio/notas.size
        if(promedio>=3.5){
            println("gano la materia")
        }else{
            println("perdio la materia")
        }
    }
    3->{
         notas.clear()
        for (i in 0 until 3) {
            println("ingrese la nota ${i+1}")
            var nota=readLine()!!.toFloat()
            notas.add(nota)
        }
    }
}

}while(opcion!=4)
}
```

conjuntos

```
fun main() {

var usuarios=setOf("kebyn","juan","tulio");

do{
println("ingrese el nombre a consultar")
var nombre=readLine()!!.toString()

if(usuarios.contains(nombre)){
    println("$nombre si se encuentra en el sistema")

}else{
    println("no se encuentra en el sistema")
}

println("ingrese 'si' si desea volver a ejecutar el programa o 'no'
para parar la ejecucion ")
var opcion=readLine()!!.toString()
}while(opcion=="si" || opcion=="Si" || opcion=="SI")


}
```

mapas

```
fun main(){

val matematicasNotas =mutableMapOf("kebyn" to 3.0)

val menu="""
1.agregar nuevo estudiante
2.consultar nota de un estudiante
3.listado de estudiantes
4.eleiminar un estudiante
5.salir
"""
var opcion=0
do{
println(menu)
opcion=readLine()!!.toInt()

when(opcion){
    1->{
```

```
        println("ingrese el nombre del estudiante")
        var nombre=readLine()!!.toString()
        var validator=false
        if(matematicasNotas.contains(nombre)){
         validator=true
          while(validator==true){
              println("el estudiante ya existe por lo cual debera
escribir su apellido o un codigo mas ")
            nombre=readLine()!!.toString()
             if(matematicasNotas.contains(nombre)){
         validator=true
        }else{
            validator=false
             println("ingrese la nota 1")
            var nota1=readLine()!!.toDouble()
            println("ingrese la nota 2")
            var nota2=readLine()!!.toDouble()
            println("ingrese la nota 3")
            var nota3=readLine()!!.toDouble()
            var notaF=(nota1+nota2+nota3)/3
            matematicasNotas[nombre]=notaF
        }
        }
        }else{

            println("ingrese la nota 1")
            var nota1=readLine()!!.toDouble()
            println("ingrese la nota 2")
            var nota2=readLine()!!.toDouble()
            println("ingrese la nota 3")
            var nota3=readLine()!!.toDouble()
            var notaF=(nota1+nota2+nota3)/3
            matematicasNotas[nombre]=notaF
        }
    }
    2-> {
        println("ingrese el nombre del estudiante")
        var nom=readLine()!!.toString()
        if(matematicasNotas.contains(nom)){
            var valor=matematicasNotas[nom]
            println(valor)
        }else{
```

```
            println("el estudiante no se encuentra en el sistema")
        }


    }
    3->{

        println("estudiantes ${matematicasNotas.keys}")
    }
    4->{
         println("ingrese el nombre del estudiante")
        var nom=readLine()!!.toString()
        if(matematicasNotas.contains(nom)){
           matematicasNotas.remove(nom)
        }else{
            println("el estudiante no se encuentra en el sistema")
        }
    }
    else{ println("ingrese una opcion valida")}
}

}while(opcion!=5)

}
```

pares

```
fun main(){
    val par = Pair("kebyn", 25)

      val nombre = par.first


    val edad = par.second
if(par.second>=18){
println("$nombre tiene $edad años por lo cual es mayor de edad")
}else{

println("$nombre tiene $edad años por lo cual es menor de edad")
}

}
```