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

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

**R/=** diferentes formas de organizar información para manipular, buscar e insertar estos datos de manera eficiente.

b. Ventajas de utilizar estructuras de datos en Kotlin

**R/=** permite ahorrar código porque no es necesario definir previamente la forma de los datos antes de usarlos.

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

**R/=** **Kotlin** tiene el soporte de conversión inteligente que identifica tipos inmutables y realiza una conversión implícita por compilador, mientras que en **Java** necesitamos identificar y realizar la conversión.

**2. Arreglos en Kotlin:**

a. ¿Qué es un arreglo?

**R/=** conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados en forma consecutiva.

b. Creación de arreglos en Kotlin

**R/=** En Kotlin podemos crear arreglos con la función **arrayOf** **(): val islands: Array = arrayOf**("Maldives", "Palawan", "Santorini") Y accedemos a cada uno de los elementos mediante el uso de un índice.

c. Accediendo a los elementos de un arreglo

**R/=** puedes acceder y traer elementos de un arreglo usando su índice. Tú necesitas usar la sintaxis square bracket para acceder a los elementos del arreglo. const element = array [index]; Según sus casos de uso.

d. Modificando los elementos de un arreglo

**R/=** Kotlin usa la clase genérica **Array<T>** para representar arreglos. Crear instancias con un tipo parametrizado usa los siguientes métodos: **arrayOf** (vararg elements:T): recibe un argumento variables con **elementos** de **tipo T** y retorna el arreglo que los contiene.

e. Recorriendo un arreglo

**R/=** Para recorrer un arreglo de Kotlin se usa un **ciclo for y el operador in:** En este caso se recorre por el valor; si se desea recorrer a través del índice hay que iterar en arreglo.índices:

f. Funciones útiles para trabajar con arreglos en Kotlin

**R/=** En Kotlin podemos crear arreglos con la función **arrayOf** **(): val islands: Array = arrayOf**("Maldives", "Palawan", "Santorini") Y accedemos a cada uno de los elementos mediante el uso de un índice.

**3. Listas en Kotlin**

a. ¿Qué es una lista?

**R/=** Una lista es una enumeración de cosas, personas, cantidades, entre otras cosas, que se realiza o confecciona con un determinado propósito.

b. Creación de listas en Kotlin

**R/=**

* **lista inmutable**

**var/val nombreLista = listOf(valor1, valor2, valor3)**

Para crear una lista tenemos la clase List predefinida en Kotlin que permite crear listas de objetos genéricos; es decir, de cualquier tipo; para crear una lista de elementos inmutables tenemos: **val list = listOf** (1, 2, 3, 4, 5).

* **lista mutable**

**var/val nombreLista = mutableListOf(valor1, valor2, valor3)**

Una lista mutable se define por el término mutableListOf en lugar de listOf como hemos visto anteriormente.

c. Accediendo a los elementos de una lista

**R/=** Acceder a cualquier elemento por un subíndice: println ("Imprimir el primer elemento de la lista") println (lista1) Si queremos conocer todas las propiedades y métodos de List podemos visitar la documentación de la biblioteca estándar de Kotlin.

d. Modificando los elementos de una lista

**R/=** La lista se puede modificar accediendo a su posición, ejemplo lista = 3 y el valor de la lista cambiará, deberías pedirle al usuario que ingrese la posición que desea cambiar y el valor.

e. Recorriendo una lista

**R/=** La manera más sencilla de hacerlo en una sola línea y al estilo Kotlin es usar un simple **map: val result = (1 until lista.size).map { lista [it-1] + lista [it] }** Te adjunto un ejemplo que puedes testear en el Playground de Kotlin.

f. Funciones útiles para trabajar con listas en Kotlin

**R/=** Existen varias funciones útiles para trabajar con ellas. **val readOnly: List<String> = listOf**("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo") **readOnly.size** //Muestra el tamaño de la lista.

**4. Conjuntos en Kotlin**

a. ¿Qué es un conjunto?

**R/=** Un conjunto es una estructura de datos que consiste en una colección de elementos cuyo orden o cantidad de repeticiones no es observado. Es decir, { 1 2 3 } { 1 3 2 } { 1 2 1 2 3 } son el mismo conjunto.

b. Creación de conjuntos en Kotlin

**R/=**

* **conjunto inmutable:**

**var/val nombreDelConjunto = setOf(valor1, valor2, valor3)**

Un conjunto inmutable se declara usando la palabra clave setOf e indica un conjunto cuyos valores no pueden cambiar a lo largo de un programa.

* **conjunto mutable:**

**var/val nombreDelConjunto = mutableSetOf<Type>()**

El término **mutableSetOf** se utiliza para declarar un conjunto mutable.

c. Accediendo a los elementos de un conjunto

**R/=** Podemos utilizar este orden junto con una función **elementAt()** para acceder y recuperar elementos.

La función **elementAt()** acepta un valor entero y devuelve el elemento en esa posición. Esta función es útil para colecciones que no poseen acceso por medio de un índice.

d. Modificando los elementos de un conjunto

**R/=** Para agregar o eliminar cualquier elemento de un conjunto, debemos utilizar las funciones de conjunto integradas **(o Built-In)** de Kotlin.

e. Recorriendo un conjunto

**R/=** En Kotlin, puede recorrer fácilmente un conjunto con la ayuda del forEach() función.

f. Funciones útiles para trabajar con conjuntos en Kotlin

R/=

* la función **first()** retorna el primer valor almacenado en el conjunto.
* la función **last()** retorna el último valor almacenado en el conjunto.
* la función **sum()** retorna la suma de todos los elementos presentes en el conjunto.
* la función **average()** retorna el promedio de los elementos almacenados en el conjunto.

**5. Mapas en Kotlin**

a. ¿Qué es un mapa?

**R/=** Un mapa consiste en un grupo de pares clave-valor donde cada clave corresponde a un solo valor. Un par clave-valor se denomina entrada y se asemeja a dos datos conectados donde una clave es un identificador único y el valor representa sus datos correspondientes.

b. Creación de mapas en Kotlin

**R/=** Para declarar un mapa inmutable, haremos uso la palabra clave **mapOf** seguida de un par de paréntesis:

**val/var nombreDelMapa = mapOf(clave1 to valor1, clave2 to valor2, clave3 to valor3).**

c. Accediendo a los elementos de un mapa

**R/=** se puede devolver un conjunto de todos los valores del mapa usando la propiedad de **values.**

d. Modificando los elementos de un mapa

**R/=**

* Podemos utilizar la función **put()** incorporada de Kotlin para agregar una nueva entrada al mapa
* para eliminar podemos utilizar **remove().**

e. Recorriendo un mapa

**R/=**Usando la sintaxis [clave], podemos recuperar y reasignar valores clave.

f. Funciones útiles para trabajar con mapas en Kotlin

**R/=**

* **abstract val entries:** [**Set**](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-set/index.html)**<**[**Entry**](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-map/-entry/index.html)**<K, V>>**

Devuelve un [conjunto](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-set/index.html#kotlin.collections.Set) de sólo lectura de todos los pares clave/valor de este mapa.

* **abstract val keys:** [**Set**](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-set/index.html)**<K>**

Devuelve un [conjunto](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-set/index.html#kotlin.collections.Set) de sólo lectura de todas las claves de este mapa.

* **abstract val size: Int**

Devuelve el número de pares clave/valor del mapa.

* **abstract val values:** [**Collection**](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-collection/index.html)**<V>**

Devuelve una [colección](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-collection/index.html#kotlin.collections.Collection) de sólo lectura de todos los valores de este mapa. Tenga en cuenta que esta colección puede contener valores duplicados.

**6. Pares en Kotlin**

a. ¿Qué es un par?

**R/=**En Kotlin, los pares se pueden crear utilizando la clase Pair. Un par es un objeto que contiene dos valores: un valor para la primera posición y otro valor para la segunda posición.

b. Creación de pares en Kotlin

**R/=**

La sintaxis para crear un par en Kotlin es la siguiente:

**val miPar = Pair(valor1, valor2)**

c. Accediendo a los elementos de un par

**R/=**También se puede acceder a cada elemento del par mediante las propiedades first y second. Por ejemplo:

**val miPar = Pair("valor1", valor2)**

d. Modificando los elementos de un par

**R/=**Además, se pueden desestructurar los pares en Kotlin, lo que significa que se pueden asignar los valores de un par a dos variables separadas. Por ejemplo:

**val (Valor1, valor2) = miPar**

**val Valor1 = miPar.first // "valor1"**

**val Valor2 = miPar.second // valor2**

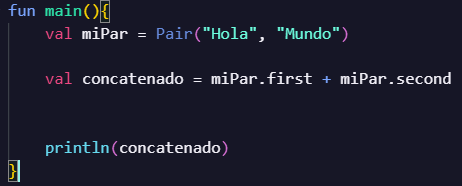
Esto es equivalente a hacer:

**val Valor1 = miPar.first**

**val Valor2 = miPar.second**

e. Recorriendo un par

**R/=**

****

f. Funciones útiles para trabajar con pares en Kotlin

**R/=**

**val Valor1 = miPar.first // "valor1"**

**val Valor2 = miPar.second // valor2**