**Introducción:**

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

Una estructura de datos es una forma de organizar y representar información en una computadora de tal manera que nos permite realizar operaciones muy eficientes y además cuenta con un comportamiento interno, Algunos métodos formales de diseño de lenguajes de programación destacan las estructuras de datos, en lugar de los algoritmos, como el factor clave de organización en el diseño de software. Más precisamente, una estructura de datos es una colección de valores, las relaciones entre ellos y las funciones y operaciones que se pueden aplicar a los datos..

En el ámbito de la informática, las estructuras de datos son aquellas que nos permiten, como desarrolladores, organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.

**Ventajas de utilizar estructuras de datos en Kotlin**

Las estructuras de datos son útiles porque nos permiten tener una colección de herramientas para solucionar diversos tipos de problemas.

Kotlin también nos proporciona diferentes estructuras de datos para almacenar y administrar datos en el programa.

Al igual que en Java, Kotlin tiene arreglos, mapas, listas, etc que nos ayudan a mejorar nuestro código aunque tienen pequeñas diferencias.

**Diferencias entre las estructuras de datos en Kotlin y Java**

Ambos tienen casi la misma velocidad de codificación y comparten unas cuantas palabras de sintaxis, pero Java extiende mas su codificacion requiriendo mas sintaxis. Sin embargo, se dedica menos tiempo a pensar la solución con Java.

En las estructuras de datos son muy parecidos, pero tienen diferentes inicializadores, por ejemplo, en Java se debe especificar de qué tipo es un arreglo, mientras que el Kotlin no es necesario (ya que kotlin tiene tipado múltiple)

En Java:

**int[ ]** edad = new **int**[8, 16, 32, 24];

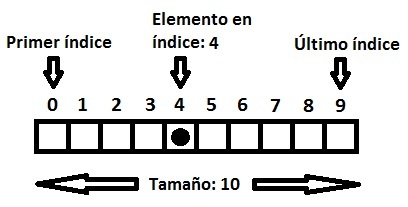
En Kotlin:

val myArray = arrayOf(32, 16, 8, 24)

**Arreglos:**

**¿Qué es un arreglo?**

es una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos del arreglo.​ Desde el punto de vista lógico un arreglo se puede ver como un conjunto de elementos ordenados en fila.



**Creación de arreglos en Kotlin**

Hay diferentes maneras de crear arreglos en kotlin:

* arrayOf(elementos: T) = recibe un argumento variables con elementos de tipo T y retorna el arreglo que los contiene.
* arrayOfNulls(size:Int): crea un arreglo de tamaño size con elementos de tipo T e inicializa los valores con null
* emptyArray(): crear un arreglo vacío con el tipo T

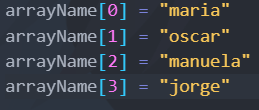
**Accediendo a los elementos de un arreglo**

La clase Array provee operadores set() y get() para asignar y obtener valores en los índices de los arreglos. aunque, es preferible utilizar el operador de acceso indexado “[ ]” para referirse a los elementos.

ejem.

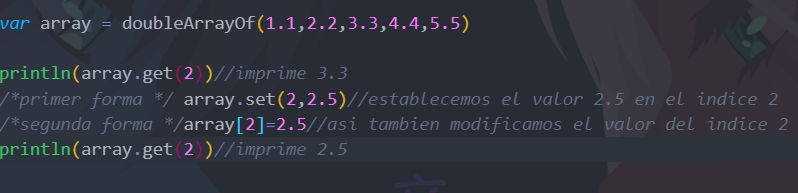
****

para acceder a un índice y agregar un valor a cada uno es de la siguiente manera:

****

**Modificando los elementos de un arreglo**

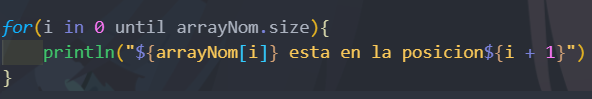
Hay dos maneras de modificar un elemento de un arreglo:

****

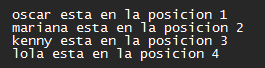
**Recorriendo un arreglo**

Una de las formas más comunes de recorrer arreglos es a través del ciclo **FOR** a través del operador **IN** junto a la propiedad de extensión “.size”. Esta contiene el tamaño del array.

Utilizando el ejemplo de los nombres en puntos anteriores, vamos a recorrerlo utilizando el ciclo **for**:



Esto mostraría lo siguiente en la consola:



**Funciones útiles para trabajar con arreglos en Kotlin**

Los arrays en Kotlin te ofrecen las funciones set() y get()

* set() sirve para asignar el valor de un índice:

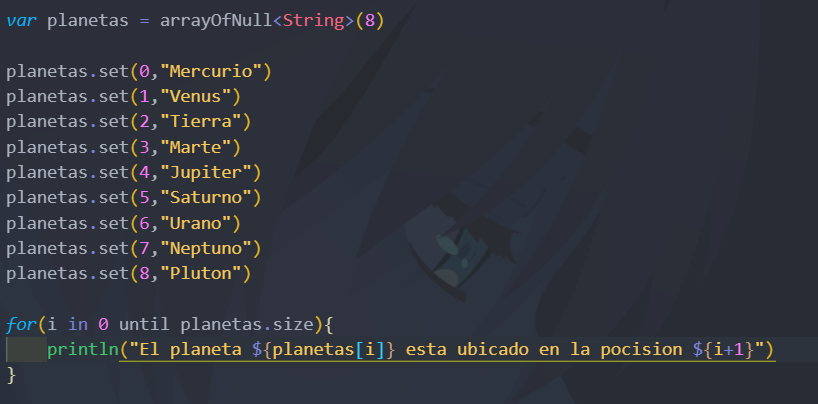


* get() sirve para obtener el valor de dicho índice:



**Ejemplo:**

Realizar un arreglo que almacene los planetas en orden, luego imprimirlos (utilizar set(), get() y el ciclo for)



**Listas:**

**¿Qué es una lista?**

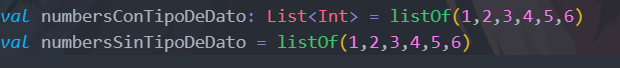
Una lista es una colección de elementos con un orden específico. Hay dos tipos de listas en Kotlin: Lista de solo lectura: List no se puede modificar después de su creación. Lista mutable: MutableList se puede modificar después de su creación, lo que significa que puedes agregar, quitar o actualizar sus elementos.

**Creación de listas en Kotlin:**

Cuando usas List o MutableList, se debe especificar el tipo de elemento que pueden contener. Por ejemplo, List<Int> contiene una lista de números enteros y List<String> contiene una lista de strings. Si se define una clase Car en tu programa, puedes tener una List<Car> que contenga una lista de instancias de objetos Car.

**Listas de solo lectura:**

Creamos una variable llamada numbers de tipo List<Int> porque contendrá una lista de solo lectura de números enteros. Creamos una nueva clase “List” con la función de la biblioteca estándar de Kotlin “listOf()” y pasa los elementos de la lista como argumentos separados por comas. “listOf(1, 2, 3, 4, 5, 6)” muestra una lista de solo lectura de números enteros del 1 al 6.



También se puede omitir el tipo de la variable en función del valor a la derecha del operador de asignación (=).

**Listas mutables:**

****

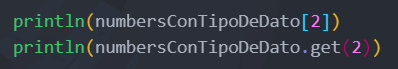
Una lista mutable es representada por la interfaz MutableList<E>. Como puedes intuir, este tipo de listas además de ser consultadas como List<E>, te permiten añadir, cambiar y remover elementos.

**Accediendo a los elementos de una lista:**

La funcionalidad específica de las listas es que puedes acceder a cada elemento de ellas mediante su índice, que es un número entero que representa la posición.

En realidad, el índice es un desplazamiento del primer elemento. Por ejemplo, cuando indicas list[2], no solicitas el segundo elemento de la lista, sino el elemento que está desplazado 2 posiciones respecto del primer elemento. Por lo tanto, list[0] es el primer elemento (ningún desplazamiento), list[1] es el segundo (desplazamiento de 1), list[2] es el tercero (desplazamiento de 2), y así sucesivamente.

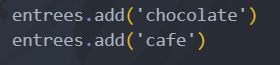
Muestra el primer elemento de la lista en el índice 0. Puedes llamar a la función get() con el índice deseado como numbers.get(0) o puedes usar la sintaxis abreviada con corchetes alrededor del índice como numbers[0].



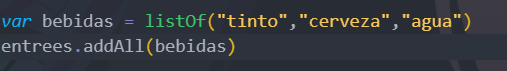
**Modificando los elementos de una lista:**

****

Agrega "chocolate" y “café” a la lista con entrees.add().



En lugar de agregar elementos uno por uno con add(), puedes agregar varios elementos a la vez usando addAll() y pasando una lista.

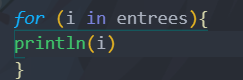
****

Para quitar un elemento llama a remove(). Para quitar el elemento "agua" de la lista se pone “entress.remove(“agua”)”

****

**Recorriendo una lista:**

Para recorrer una lista se utiliza el ciclo “for”, así como con un arreglo

****

**Funciones útiles para trabajar con listas en Kotlin:**

* **add()** // Agregamos algo a la lista
* **addAll()** // Agregamos una lista entera a nuestra lista
* **remove()** // Quitamos un elemento de una lista
* **removeAt()** // Quitamos un elemento a través de un índice
* **clear()** // Eliminamos todo lo de la lista
* **isEmpty()** // Revisamos si la lista está vacía
* **first()** // Muestra el primer elemento de la lista
* **last()** // Muestra el último elemento de la lista
* **contains()** // Muestra una lista nueva en la que los elementos se encuentran en orden inverso
* **reversed()** // Muestra otra en la que los elementos están ordenados en forma ascendente

**Conjuntos:**

**¿Qué es un conjunto?**

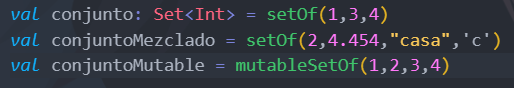
Un conjunto (Set) es una colección sin orden de elementos únicos, esto es, no puede tener ningún duplicado. Set extiende de la interfaz Collection, por lo que es inmutable, y no agrega nuevos métodos.

**Creación de conjuntos en Kotlin**

Para crear conjuntos inmutables utiliza:

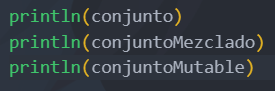
**setOf()** crea un conjunto inmutable y devuelve un objeto de tipo Set.

También se pueden crear conjuntos mutables: **mutableSetOf()**

****

**Accediendo a los elementos de un conjunto:**

No se accede los elementos ya que se utilizan como un solo grupo, se puede ver que hay dentro del conjunto imprimiendo:

****

**Modificando los elementos de un conjunto:**

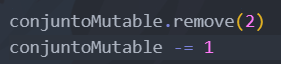
**Agregar elementos:**

Agrega elementos al conjunto a través del método **add()** o adición compuesta **(+=)** de las colecciones:

****

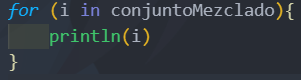
**Remover elementos:**

Como es normal usa el método **remove()** para remover elementos de un conjunto o resta compuesta **(-=)** para conseguir el mismo resultado.

****

**Recorriendo un conjunto:**

También podemos recorrer un conjunto utilizando el ciclo for:

****

**Funciones útiles para trabajar con conjuntos en Kotlin:**

* **add()** // agregar un elemento en el conjunto
* **remove()** // quitar un elemento del conjunto
* **union()** // toma como argumentos dos colecciones y retorna en un conjunto con todos los elementos que pertenezcan a ambas.
* **intersect()** // aislar solo los elementos que estén presentes en dos colecciones
* **subtract()** // calcular la diferencia entre dos colecciones

**Mapas:**

**¿Qué es un mapa?**

Los mapas asocian claves con valores. Las claves deben ser únicas, pero los valores asociados no. De este modo, cada clave puede ser usada para identificar de manera única el valor asociado, ya que el mapa asegura que no puedes duplicar claves en la colección. Internamente, Kotlin usa la colección Java Map para implementar los mapas.

**Creación de mapas en Kotlin:**

**mapOf()** crea un mapa inmutable compuesto por una lista de pares, donde el primer valor es la clave, y el segundo es el valor. Devuelve un objeto de tipo Map.

val paises: Map<Int, String> = mapOf(1 to "Colombia", 2 to "España", 3 to "Islandia")

**mutableMapOf()** sirve para crear mapas mutables

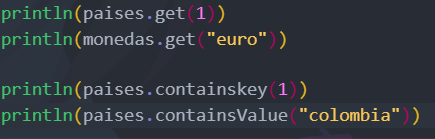
val monedas: MutableMap<String, String> = mutableMapOf("peso" to "Colombia", "euro" to "España")

****

**Accediendo a los elementos de un mapa:**

Podemos obtener la clave de un mapa utilizando el método get() o usando las “[]”

También podemos mirar si el mapa tienen un valor o una clave en específico usando los métodos containsKey() y containsValue()

****

**Modificando los elementos de un mapa:**

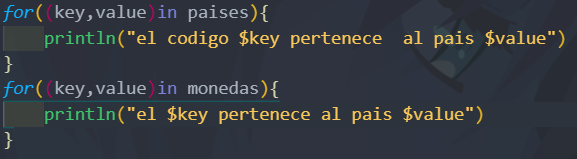
Con **put(key: K, value: V)** inserta el par clave-valor en el mapa. Devolverá el valor previo enlazado con la clave o null si la clave no existía.

(Solo se puede utilizar en mapas mutables)

****

**Recorriendo un mapa:**

Como en los casos anteriores podemos recorrer los mapas utilizando el ciclo for:

****

**Funciones útiles para trabajar con mapas en Kotlin:**

* **get(K)** // obtiene el valor de la clave
* **isEmpty()** // Revisa si el mapa está vacío
* **containsKey(K)** // Revisa si existe la clave en el mapa (devuelve un boolean)
* **containsValue(V)** // Revisa si el valor está en el mapa (devuelve un boolean)
* **keys** // devuelve un Set inmutable con todas las claves en el mapa.
* **values** // Collection inmutable de todos los valores en el mapa.
* **remove(K)** // borra la clave y su valor enlazado. (Solo en mutables)
* **clear()** // elimina todos los elementos del mapa.

**PARES**

**¿Qué es un par?**

Los **data class Pair** son una estructura que permite guardar dos valores. Esto en Android es muy útil ya que podemos emplear los **Pair** Para guardar una pareja de valores, por ejemplo un usuario y contraseña o cualquier pareja de valores que tengan alguna relación.

**Creación de pares en Kotlin**

Para emplear los valores pares en Kotlin podemos hacer lo siguiente:

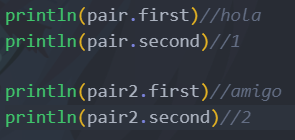
****

También podemos crear valores pares empleando la variable reservada to de la siguiente forma:

****

**Accediendo a los elementos de un par**

Ya teniendo claro para qué sirve la estructura de los Pair en Kotlin, lo siguiente que nos interesa saber es cómo acceder a cada uno de estos valores embebidos dentro de un data class Pair; para esto se emplean las palabras reservadas first y second respectivamente de la siguiente forma; teniendo nuestra variable llamada pair definida de cualquiera de las formas anteriores:

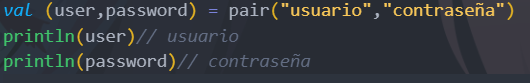
****

Y obtenemos como salida:



Para cada uno de los println respectivamente.

También podemos separar o descomponer los valores de los pares en variables individuales e independientes de la siguiente forma:

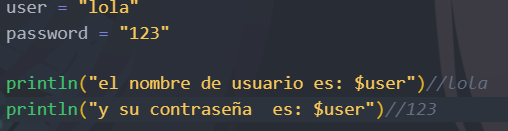


Y se obtiene:



**Modificando los elementos de un par**

Para modificar alguno de los pares se debe independizar primero, luego se le asigna un valor a cada par:



**Recorriendo un par**

No encuentro información al respecto, si encuentra algo por favor deje su comentario.

**Funciones útiles para trabajar con pares en Kotlin**

No encuentro información al respecto, si encuentra algo por favor deje su comentario.

**Prácticas de estructuras de datos**

**Ejercicios prácticos para aplicar los conceptos aprendidos:**

* Realiza un algoritmo que le pida al usuario que ingrese n cantidad de nombres y guardalos en un arreglo.
* Realiza un algoritmo que contenga una lista de comidas y que se imprima en consola, el usuario deberá poder eliminar algo de la lista.
* Realiza un algoritmo que de dos conjuntos saque el número menor que se repita en ambos conjuntos e indique si es mayor a 5.
* Realizar un Mapa que tenga 10 países diferentes y que el usuario pueda acceder a la clave y se imprima el país en consola.
* Realizar un algoritmo que almacene el tipo y la especie de un animal y luego imprima los valores.

**Solución a los ejercicios prácticos:**

Repositorio con los ejercicios: