Seminario 1 TDAM

A trabajar!!

Explicaciones iniciales ¬¬

Cómo declarar un array

Cómo declarar un array

Declarando un Array de Strings con tamaño fijo:

```
val miArrayDeStrings = Array(5) { "" }
```

Declarando un Array de enteros con tamaño fijo:

```
val miArrayDeEnteros = Array(5) { 0 }
```

Declarando un Array de cualquier tipo con elementos específicos:

```
val miArray = arrayOf("elemento1", "elemento2", "elemento3")
```

Declarando un Array de enteros con elementos específicos:

```
val miArrayDeEnteros = intArrayOf(1, 2, 3, 4, 5)
```

Utilizando intArrayOf() con elementos como argumentos:

val miLista = intArrayOf(1, 2, 3, 4, 5)

Declarando una lista de enteros inmutable con val:

val miLista = intArrayOf(1, 2, 3, 4, 5)

Declarando una lista de enteros modificable con var:

var miListaMutable = intArrayOf(1, 2, 3, 4, 5)

Declarando un IntArray y luego convirtiéndolo en una lista:

val miArray = intArrayOf(1, 2, 3, 4, 5)

val miLista = miArray.toList()

Utilizando intArrayOf() para inicializar un IntArray vacío y luego agregando elementos:

```
val miListaVacia = IntArray(5)

miListaVacia[0] = 1

miListaVacia[1] = 2

miListaVacia[2] = 3

miListaVacia[3] = 4

miListaVacia[4] = 5
```

Formas de declarar una lista

Declarando una lista utilizando listOf():

```
val miLista = listOf("manzana", "banana", "cereza")
```

Declarando una lista modificable utilizando mutableListOf():

```
val miListaMutable = mutableListOf("manzana", "banana", "cereza")
```

Declarando una lista modificable basada en ArrayList utilizando arrayListOf():

```
val miArrayList = arrayListOf("manzana", "banana", "cereza")
```

Declarando una lista vacía y luego agregando elementos a ella:

```
val miListaVacia = mutableListOf<String>()

miListaVacia.add("manzana")

miListaVacia.add("banana")

miListaVacia.add("cereza")
```

Utilizando el operador listOf con elementos como argumentos:

```
val miLista = listOf("manzana", "banana", "cereza")
```

Declarando una lista de un tipo específico utilizando la notación de tipo genérico:

```
val miLista: List<String> = listOf("manzana", "banana", "cereza")
```

Utilizando la función arrayOf() para crear una lista basada en un array:

```
val miLista = arrayOf("manzana", "banana", "cereza").toList()
```

Utilizando el constructor de una lista:

val miLista = List(3) { index -> "Elemento $index" }

val a = List(5) { it + 1 }

```
@SinceKotlin
@InlineOnly
public inline fun <T> List(
    size: Int,
    init: (Int) -> T
): List<T>
```

Creates a new read-only list with the specified `size`, where each element is calculated by calling the specified `init` function.

The function `init` is called for each list element sequentially starting from the first one. It should return the value for a list element given its index.

Declarando una lista inmutable utilizando la palabra clave val:

val miLista = listOf("manzana", "banana", "cereza")

Declarando una lista mutable utilizando la palabra clave var:

var miListaMutable = mutableListOf("manzana", "banana", "cereza")

La razón por la que la inicialización de elementos se coloca dentro de llaves {} en lugar de paréntesis () en la función List se debe al uso de una lambda (una función anónima) para determinar cómo se deben inicializar los elementos de la lista.

En Kotlin, las llaves {} se utilizan para definir bloques de código, como funciones anónimas o lambdas. En este caso, la lambda se utiliza para especificar cómo se debe inicializar cada elemento de la lista. La lambda toma un parámetro (index en este caso) y utiliza ese parámetro para generar el valor de cada elemento de la lista. Esto permite una personalización completa de cómo se crean los elementos de la lista en función de su posición.

Por lo tanto, al usar {} en lugar de (), estás indicando que deseas proporcionar una lógica personalizada para la inicialización de los elementos de la lista, lo que te permite tener un control más preciso sobre el contenido de la lista.

Otro ejemplo de funciones anónimas

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

Utilizando la función filter de List con una función anónima como parámetro

```
val numerosPares = numeros.filter { numero -> numero % 2 == 0 }
```

Utilizando la función map de List con una función anónima como parámetro

```
val cuadrados = numeros.map { numero -> numero * numero }
```

Si necesitas dos funciones anónimas en Kotlin, puedes pasarlas como argumentos a una función que acepte dos funciones como parámetros.

Por ejemplo:

```
fun operaciones(num1: Int, num2: Int, operacion1: (Int, Int) -> Int, operacion2: (Int, Int) -> Int): Int {
    val resultado1 = operacion1(num1, num2)
    val resultado2 = operacion2(num1, num2)
    return resultado1 + resultado2
}
fun main() {
    val suma = operaciones(5, 3, { a, b -> a + b }, { a, b -> a + b })
    val resta = operaciones(5, 3, { a, b -> a - b }, { a, b -> a - b })
    println("Suma: $suma")
    println("Resta: $resta")
}
```

val miLista = List(3) { index -> "Elemento $index" }

Esta línea de código crea una lista inmutable (List) en Kotlin con un tamaño fijo de 3 elementos y utiliza una lambda para inicializar cada elemento de la lista.

Aquí está la descomposición de la línea:

- **val miLista:** Esto declara una variable inmutable llamada miLista que almacenará la lista que estamos creando.
- **List(3)**: Aquí estamos utilizando el constructor de la clase List para crear una lista con un tamaño de 3 elementos. Esto significa que la lista tendrá espacio para 3 elementos, pero inicialmente estará vacía.
- **{ index -> "Elemento $index" }:** Esta parte es una **lambda o función anónima.** La palabra index es un parámetro de la lambda que representa la posición del elemento en la lista (0, 1 o 2 en este caso, ya que tenemos una lista de tamaño 3). La lambda devuelve "Elemento $index", que es una cadena que combina la palabra "Elemento" con el valor de index. Entonces, para cada posición en la lista, la lambda generará un elemento con un valor diferente.

Por lo tanto, después de ejecutar esta línea de código, miLista será una lista inmutable con los siguientes elementos:

"Elemento 0"

"Elemento 1"

"Elemento 2"

Es una forma concisa de crear una lista con valores específicos basados en una lógica definida en la lambda.

arrayOfNulls

Utilizando arrayOfNulls para un Array de cadenas (String) con tamaño especificado:

```
val miArrayDeNulos = arrayOfNulls<String>(5)
```

Utilizando arrayOfNulls para un Array de enteros (Int) con tamaño especificado:

```
val miArrayDeNulosEnteros = arrayOfNulls<Int>(3)
```

Utilizando arrayOfNulls para un Array de cualquier tipo (Any) con tamaño especificado:

```
val miArrayDeNulosAny = arrayOfNulls<Any>(4)
```

Creando un Array de cadenas (String) e inicializándolo con elementos nulos:

```
val miArrayDeNulos = Array<String?>(5) { null }
```

Creando un Array de enteros (Int) e inicializándolo con elementos nulos:

```
val miArrayDeNulosEnteros = Array<Int?>(3) { null }
```

Creando un Array de cualquier tipo (Any) e inicializándolo con elementos nulos:

```
val miArrayDeNulosAny = Array<Any?>(4) { null }
```

Utilizando el operador null para inicializar un Array de cadenas (String) con elementos nulos:

```
val miArrayDeNulos = arrayOfNulls<String?>(5)
```

Ejercicio 1. Crea una función que obtenga el número máximo de una lista de números

Ejercicio 2. Crea una función que obtenga la sumatoria de una lista de números

Ejercicio 3. Crea una función que dada una distancia en millas calcule su correspondiente en kms

Ejercicio 4. Crea una función que determine si una cadena de texto es un palíndromo.

Ejercicio 5. Crea una función que cuenta cuántas veces aparece una letra en un texto.

Ejercicio 6. Crea una función que cuenta cuántas veces aparece una subcadena en un texto.

Ejercicio 7. Crea una función que pone en mayúscula la primera letra de cada palabra de un texto

Ejercicio 8. Crea una función que sume los dígitos de un número. Ejemplo: sumaDigitos(245) = 2 + 4 + 5 = 11

Ejercicio 9. Crea una función que calcule el máximo común divisor de dos números naturales

Ejercicio 10. Crea una función que calcule el término n-ésimo de la sucesión de Finbonacci.

En matemática, la sucesión de Fibonacci se trata de una serie infinita de números naturales que empieza con un 0 y un 1 y continúa añadiendo números que son la suma de los dos anteriores: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597...

Ejercicio 11. Crea una función que determine si dos números son primos relativos.

Se dice que dos números son relativamente primos si su factor común más grande (FCG) es 1. Ejemplo 1: Los factores de 20 son 1, 2, 4, 5, 10 y 20. Los factores de 33 son 1, 3, 11, y 33.

Ejercicio 12. Crea una función que determine si un número dado es capicúa

Ejercicio 13. Crea una función que dada una cadena de texto con formato Emmet devuelva su correspondiente etiqueta HTML, teniendo en cuenta sólo los atributos de clase e id.

Ejemplos:

in: a -> out: <a>

in: div.oferta -> out:

In: div.coche#VWPolo -> out:

Ejercicio 14. Crea una función que dado un número n imprima el siguiente ‘mosaico’

(para n = 6):

1

22

333

4444

55555

666666

Ejercicio 15. Crear una función que reciba dos arrays de enteros y devuelva un array de booleanos que determine si los elementos, uno a uno, de ambos arrays son iguales

Ejercicio 16: Crea una función que calcule el producto de todos los elementos en una lista de números.

Ejercicio 17: Crea una función que dada una lista de números, devuelva una nueva lista con solo los números pares.

Ejercicio 18: Crea una función que determine si un número es primo.

Ejercicio 19: Crea una función que, dada una cadena de texto, elimine todas las vocales de la cadena.

Ejercicio 20: Crea una función que calcule el factorial de un número.

Ejercicio 21: Crea una función que invierta una cadena de texto. Por ejemplo, "hola" debería convertirse en "aloh".

Ejercicio 22: Crea una función que, dado un número, devuelva True si es un número perfecto (la suma de sus divisores propios positivos es igual al número), o False en caso contrario.

Ejercicio 23: Crea una función que, dado un número entero, devuelva True si es un número Armstrong (un número que es igual a la suma de sus propios dígitos elevados a una potencia). Por ejemplo, 153 es un número Armstrong porque 1^3 + 5^3 + 3^3 = 153.

Ejercicio 24: Crea una función que encuentre el número más grande en una matriz bidimensional (una lista de listas).

Ejercicio 25: Crea una función que encuentre el número más pequeño en una matriz bidimensional (una lista de listas).

Ejercicio 26: Crea una función que, dada una lista de palabras, devuelva la palabra más larga.

Ejercicio 27: Crea una función que, dada una lista de palabras, devuelva la palabra más corta.

Ejercicio 28: Crea una función que determine si una cadena de texto contiene solo caracteres alfabéticos (letras) y espacios en blanco.

Ejercicio 29: Crea una función que determine si una cadena de texto es un anagrama de otra cadena. Dos palabras son anagramas si tienen las mismas letras, pero en un orden diferente.

Ejercicio 30: Crea una función que, dado un número entero, devuelva True si es un número triangular (puede representarse como un triángulo equilátero de puntos), o False en caso contrario.

Subir nota

Subir nota

Ejercicio 31: Duplicar Elementos en una Lista

Escribe una función que tome una lista de números y utilice la función map para duplicar cada número en la lista. La función debe devolver una nueva lista con todos los números duplicados.

Por ejemplo, si la entrada es [1, 2, 3, 4], la función debe devolver [2, 4, 6, 8].

Subir nota

Ejercicio 32: Crear un Diccionario a partir de Listas

Escribe una función que tome dos listas, una lista de claves y otra lista de valores, y cree un diccionario utilizando mapOf para combinar las listas en un diccionario clave-valor. La función debe devolver el diccionario resultante.

Subida nota

Ejercicio 33: Crea un programa se encargue de transformar un número decimal a binario sin utilizar funciones propias del lenguaje que lo hagan directamente.

Ejercicio 34: Crea una función que sea capaz de encriptar y desencriptar texto utilizando el algoritmo de encriptación de Karaca (debes buscar información sobre él).

Subida nota

Ejercicio 35: Crea una función que ordene y retorne una matriz de números.

- La función recibirá un listado (por ejemplo [2, 4, 6, 8, 9]) y un parámetro adicional

- "Asc" o "Desc" para indicar si debe ordenarse de menor a mayor o de mayor a menor.

- No se pueden utilizar funciones propias del lenguaje que lo resuelvan automáticamente.