# Taller Estructuras de Datos en Kotlin

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

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

Una colección puede ser una lista ordenada, una agrupación de valores únicos o una asignación de valores de un tipo de datos a valores de otro tipo. te permiten almacenar varios valores, generalmente del mismo tipo de datos, de manera organizada.

### b. Ventajas de utilizar estructuras de datos en Kotlin

Facilidad de uso, Eficiencia, Seguridad del tipo, inmutabilidad, flexibilidad.

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

**Nullabilidad:** Kotlin admite la nullabilidad de forma segura en sus estructuras de datos

**Inmutabilidad:** En Kotlin, muchas de las estructuras de datos son inmutables por defecto, lo que significa que una vez que se crean, no se pueden modificar. En Java, muchas de las estructuras de datos son mutables por defecto, lo que permite cambiar los valores de sus elementos.

**Sintaxis simplificada**: Kotlin ofrece una sintaxis más simplificada y expresiva para las estructuras de datos.

**Rangos:** Kotlin tiene un tipo de dato "rango" que puede usarse para crear una secuencia de valores consecutivos de manera simple y eficiente.

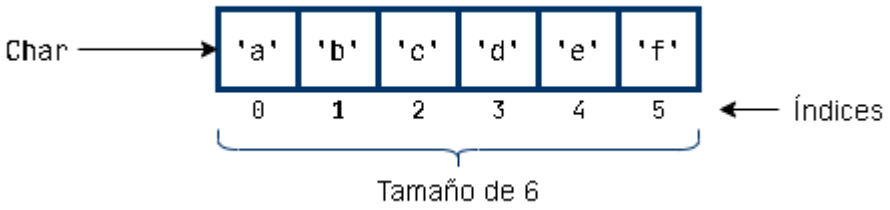
**Otras estructuras de datos:** Kotlin ofrece algunas estructuras de datos adicionales, como las secuencias, que permiten trabajar con colecciones de datos sin necesidad de cargar todos los datos en la memoria al mismo tiempo.

## 2. Arreglos en Kotlin

### a. ¿Qué es un arreglo?

Un arreglo es una estructura con valores de datos, que están almacenados de forma contigua en memoria. Todos los elementos son referenciados por un mismo nombre y tienen el mismo tipo de dato.

Los elementos estarán indexados tomando como base el 0 y el tamaño declarado del arreglo será fijo.




### b. Creación de arreglos en Kotlin

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 variable con elementos de tipo T y retorna el arreglo que los contiene.

### c. Accediendo a los elementos de un arreglo

La clase array nos brinda dos operadores **set()** y **get()**


Esto se llama sintaxis de subíndice. Consta de tres partes:

- El nombre del array
- Un corchete angular de apertura ([) y uno de cierre (])
- El índice del elemento de array entre corchetes

### d. Modificando los elementos de un arreglo

### e. Recorriendo un arreglo

Una de las formas más convencionales de recorrer arreglos es a través del bucle for a través del operador in junto a la propiedad de extensión índices. Esta contiene el rango válido de los índices del array.

```
for(i in planets.indices){
    println("${planets[i]} está en la posición ${i+1}")
}
```

### f. Funciones útiles para trabajar con arreglos en Kotlin

**arrayOf():** Crea un nuevo arreglo de elementos del mismo tipo y longitud.

**arrayOfNulls():** Crea un nuevo arreglo de elementos nulos del mismo tipo y longitud.

**size:** Devuelve el tamaño del arreglo.

**get(index):** Devuelve el elemento del arreglo en el índice especificado.

**set(index, value):** Asigna el valor especificado al elemento en el índice especificado.

**drop():** Devuelve un nuevo arreglo que omite los primeros n elementos del arreglo original.

**filter():** Devuelve un nuevo arreglo que contiene solo los elementos del arreglo original que cumplen con la condición especificada.

**map():** Devuelve un nuevo arreglo que contiene los elementos del arreglo original transformados por la función especificada.

**reduce():** Devuelve un solo valor obtenido mediante la aplicación repetida de la operación especificada a los elementos del arreglo.

## 3. Listas en Kotlin

### a. ¿Qué es una lista?

Una lista es una colección redimensionable y ordenada que, por lo general, se implementa como un array que puede cambiar de tamaño. Cuando el array alcanza su capacidad máxima y tratas de insertar un nuevo elemento, el array se copia en un nuevo array más grande.

### b. Creación de listas en Kotlin

- Si deseas crear una lista debes primero definir si será de solo lectura o mutable.

  ○ **Listas De Solo Lectura:** Para crear una lista de solo lectura usa la función listOf(), la cual recibe como argumentos un grupo de ítems de un mismo tipo.

```
fun main() {
    val oneToFiveList: List<Int> = listOf(1, 2, 3, 4, 5)
    println(oneToFiveList) // [1, 2, 3, 4, 5]
}
```

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

```
interface MutableList<E> : List<E>, MutableCollection<E>
```

## c. Accediendo a los elementos de una lista

Al igual que un array, puedes acceder a un elemento en un índice específico desde un elemento List con la sintaxis de subíndice. Puedes hacer lo mismo si usas el método get().

Además de obtener un elemento a través de su índice, puedes buscar el índice de un elemento específico con el método indexOf(). El método indexOf() busca en la lista un elemento determinado (pasado como un argumento) y muestra el índice de la primera instancia de ese elemento. Si el elemento no aparece en la lista, se muestra -1.

## d. Modificando los elementos de una lista

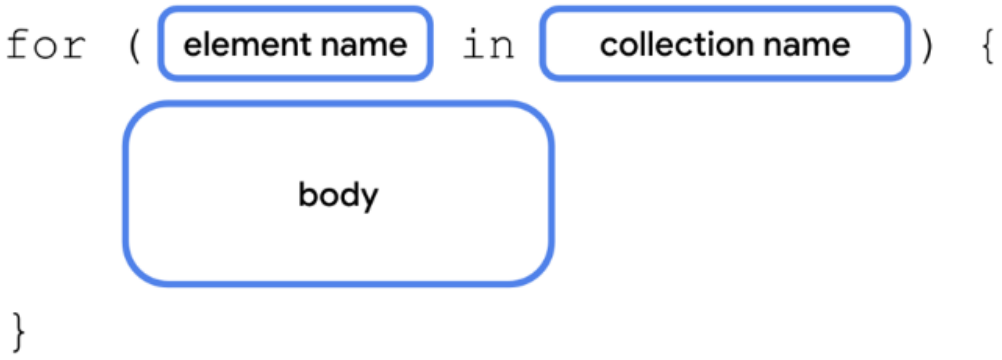
- `set(index: Int, element: E): E` sustituye un elemento en la lista. Devuelve el elemento que estaba antes. También podemos usar `[index]`
- `add(index: Int, element: E)` inserta un elemento.
- `removeAt(index: Int)` elimina el elemento en un índice.

### - Convertir una lista inmutable a mutable

```
val nombres: List<String> = listOf("Homer", "Bart", "Maggie")
val nombresMutables = nombres.toMutableList()
mutableNames1.add("Lisa")
```

## e. Recorriendo una lista

Kotlin incluye una función llamada bucle for para lograr esto con una sintaxis concisa y legible. A menudo, verás que esto se conoce como bucle a través de una lista o iteración sobre una lista.




## f. Funciones útiles para trabajar con listas en Kotlin

**listOf():** Crea una lista inmutable de elementos.

**mutableListOf():** Crea una lista mutable de elementos.

**size:** Devuelve el tamaño de la lista.

**get(index):** Devuelve el elemento de la lista en el índice especificado.

**set(index, value):** Asigna el valor especificado al elemento en el índice especificado.

**subList(fromIndex, toIndex):** Devuelve una vista de la lista que abarca desde el índice de inicio hasta el índice final.

**any():** Devuelve true si al menos un elemento de la lista cumple con la condición especificada.

**all():** Devuelve true si todos los elementos de la lista cumplen con la condición especificada.

**count():** Devuelve el número de elementos en la lista que cumplen con la condición especificada.

**distinct():** Devuelve una lista que contiene solo los elementos distintos de la lista original.

## 4. Conjuntos en Kotlin

### a. ¿Qué es un conjunto?

Un conjunto es una colección que no tiene un orden específico y no permite valores duplicados.

¿Por qué es posible crear una colección como esta? El secreto es un código hash. Un código hash es un Int que produce el método hashCode() de cualquier clase de Kotlin. Se puede interpretar como un identificador semiúnico de un objeto de Kotlin. Un pequeño cambio en el objeto, como agregar un carácter a la String, da como resultado un valor de hash muy diferente. Si bien es posible que dos objetos tengan el mismo código hash (llamado colisión de hash), la función hashCode() garantiza cierto grado de exclusividad. En la mayoría de los casos, dos valores diferentes tienen un código hash único.

**b. Creación de conjuntos en Kotlin**

Para crear un conjunto de solo lectura usa la función setOf() y pasa como argumento la cantidad de elementos que albergará.

```
fun main() {
    val positiveNumbers = setOf(1, 2, 3, 4)
    println(positiveNumbers)
```

**c. Accediendo a los elementos de un conjunto**

Para acceder a los elementos de un conjunto, se puede usar la función forEach() o iterar sobre el conjunto con un bucle for.

se puede acceder a los elementos de un conjunto por índice, ya que los conjuntos no tienen un orden específico

```
val mySet = setOf("a", "b", "c", "a")

mySet.forEach { println(it) } // Imprime "a", "b", "c"

for (element in mySet) {

    println(element) // Imprime "a", "b", "c"

}
```

**d. Modificando los elementos de un conjunto**

**Añadir Elementos**

Agrega elementos al conjunto a través del método add() o usando los operadores de adición (+) o adición compuesta (+=) de las colecciones

```
fun main() {
    // Añadir elementos
    setB.add(1) // [1]
    setB += 2   // [1,2]
    setB += 2   // [1,2]
    println(setB)
}
```

**Remover Elementos**

Como es normal usa el método remove() para remover elementos de un conjunto. O similar a la agregación, usa el operador de resta (-) o resta compuesta (-=) para conseguir el mismo resultado.

```
fun main() {
    //...

    // Remover elementos
    setB.remove(1)   // [2]
    setB -= 2        // []
    setB -= 3        // []

    println(setB)
}
```

**e. Recorriendo un conjunto**

**1. Usando forEach() función**

En Kotlin, puede recorrer fácilmente un conjunto con la ayuda del forEach() función, como se muestra a continuación:

```
fun main() {
    val set: Set<Int> = setOf(1, 2, 3, 4, 5)

    set.forEach { println(it) }
}
```

**2. Usando bucle foreach**

Alternativamente, puede usar un bucle for o foreach para recorrer Set.

```
fun main() {
    val set: Set<Int> = setOf(1, 2, 3, 4, 5)

    for (item in set) {
        println(item)
    }
}
```

### 3. Usando el iterador

También puede llamar al iterator() que devuelve un iterador sobre un conjunto, como se muestra a continuación:

```
fun main() {
    val set: Set<Int> = setOf(1, 2, 3, 4, 5)

    val it = set.iterator()
    while (it.hasNext()) {
        println(it.next())
    }
}
```

### f. Funciones útiles para trabajar con conjuntos en Kotlin

**setOf():** Crea un conjunto inmutable de elementos.

**mutableSetOf():** Crea un conjunto mutable de elementos.

**size:** Devuelve el tamaño del conjunto.

**contains(element):** Devuelve true si el conjunto contiene el elemento especificado.

**add(element):** Agrega el elemento especificado al conjunto.

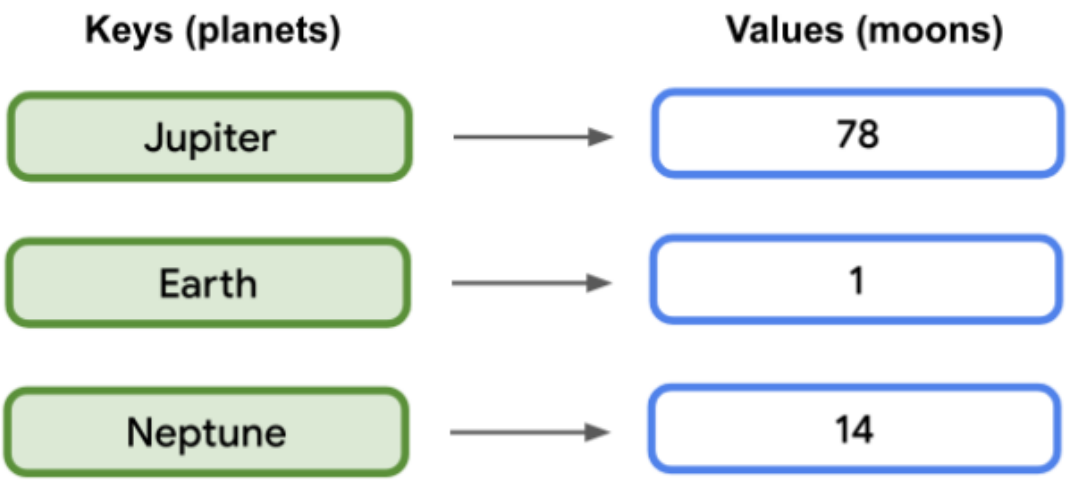
**addAll(elements):** Agrega todos los elementos especificados al conjunto.

**remove(element):** Elimina el elemento especificado del conjunto.

## 5. Mapas en Kotlin

### a. ¿Qué es un mapa?

Un Map es una colección que consta de claves y valores. Se llama un mapa porque las claves únicas se asignan a otros valores. Una clave y su valor complementario suelen llamarse key-value pair.




### b. Creación de mapas en Kotlin

La interfaz que representa a los mapas en Kotlin es Map<K,V>. Donde los parámetros de tipo K y V representan a los tipos para claves (propiedad keys) y valores (propiedad values).

```
interface Map<K, out V>
```

### c. Accediendo a los elementos de un mapa

Debido a la naturaleza de los mapas en Kotlin, es posible desestructurar las declaraciones que comprometan a sus entradas en valores individuales.

Un ejemplo claro de esto es recorrer sobre los elementos de un mapa en un bucle for.

```
fun main() {
    val operationsMap = mapOf(
        "Suma" to '+',
        "Resta" to '-',
        "Multiplicación" to 'x',
        "División" to '÷'
    )

    for ((operation, symbol) in operationsMap) {
        println("$operation -> $symbol")
    }
}
```

**d. Modificando los elementos de un mapa**

actualizar los elementos de un mapa mediante la función map(), que devuelve un nuevo mapa que contiene los elementos transformados por una función dada.

```
val map = mapOf("key" to 1, "foo" to 2, "bar" to 3)
val newMap = map.mapValues { it.value * 2 }
println(newMap) // Imprime "{key=2, foo=4, bar=6}"
```

**e. Recorriendo un mapa**

utilizar la función put(key: K, value: V) para agregar o actualizar elementos en un mapa mutable.

```
val mutableMap = mutableMapOf("key" to 1, "foo" to 2, "bar" to 3)
mutableMap.put("key", 4) // actualiza el valor asociado con la clave "key" a 4
mutableMap.put("baz", 5) // agrega un nuevo elemento con la clave "baz" y el valor
```

se tiene un mapa de cadenas a enteros, se puede actualizar el valor asociado con la clave "key" de la siguiente manera:

```
val map = mapOf("key" to 1, "foo" to 2, "bar" to 3)
map["key"] = 4 // actualiza el valor asociado con la clave "key" a 4
```

**f. Funciones útiles para trabajar con mapas en Kotlin**

**filterValues(predicate):** Devuelve un mapa que contiene solo las entradas cuyos valores cumplen con la condición dada por la función de predicado.

**filter(predicate):** Devuelve un mapa que contiene solo las entradas que cumplen con la condición dada por la función de predicado.

**mapValues(transform):** Devuelve un mapa que contiene los valores transformados por la función de transformación.

**forEach(action):** Ejecuta la función de acción dada para cada entrada en el mapa.

**getOrDefault(key, defaultValue):** Devuelve el valor asociado con la clave especificada, o el valor predeterminado si la clave no está presente en el mapa.

**getOrElse(key, defaultValue**): Devuelve el valor asociado con la clave especificada, o el resultado de una función predeterminada si la clave no está presente en el mapa.

**put(key, value):** Agrega o actualiza un valor en el mapa. Si la clave ya está presente, el valor se actualiza.

**remove(key):** Elimina el valor asociado con la clave especificada del mapa.

**containsKey(key):** Devuelve true si el mapa contiene la clave especificada.

**containsValue(value):** Devuelve true si el mapa contiene el valor especificado.

**keys:** Devuelve un conjunto que contiene todas las claves en el mapa.

**values:** Devuelve una lista que contiene todos los valores en el mapa.

## 6. Pares en Kotlin

### a. ¿Qué es un par?

es un objeto que contiene dos elementos. Se utiliza para agrupar dos valores relacionados y pasarlos juntos como un solo objeto.

### b. Creación de pares en Kotlin

val par = Pair(valor1, valor2)

Donde valor1 y valor2 son los dos valores que se desean agrupar.

También se puede utilizar una sintaxis abreviada para crear un par:

val par = valor1 to valor2

Los pares son objetos inmutables, lo que significa que no se pueden cambiar una vez que se han creado.

### c. Accediendo a los elementos de un par

Los valores contenidos en un par se pueden acceder utilizando las propiedades first y second.

```
val par = Pair("Hola", 123)
println(par.first)  // Imprime "Hola"
println(par.second) // Imprime 123
```

Los pares son comúnmente utilizados en Kotlin para devolver múltiples valores de una función, para emparejar dos elementos en una lista o conjunto, o para agrupar dos elementos relacionados de alguna otra manera.

### d. Modificando los elementos de un par

Una vez creado un par en Kotlin no se puede modificar, Para cambiar los valores de un par, debe crearse un nuevo objeto de par con los valores actualizados.

```
val par = Pair(a, b)
val nuevoPar = Pair(nuevoValor, b)
```

### e. Recorriendo un par

Acceder a los valores individualmente:

```
val par = Pair("Hola", 123)
val a = par.first
val b = par.second
println("a = $a, b = $b") // Imprime "a = Hola, b = 123"
```

Desestructurar el par:

```
val par = Pair("Hola", 123)
val (a, b) = par
println("a = $a, b = $b") // Imprime "a = Hola, b = 123"
```

Utilizar la función forEach:

```
val par = Pair("Hola", 123)
par.forEach { valor ->
    println(valor)
}
```

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

**first y second:** Estas son propiedades de los objetos de par que permiten acceder al primer y segundo elemento del par, respectivamente.

**to:** Esta función permite crear un objeto de par a partir de dos valores.

**component1 y component2:** Estas son funciones que permiten desestructurar un objeto de par en dos variables.

**copy:** Esta función permite crear una copia del objeto de par con uno o ambos elementos actualizados.

**equals y hashCode:** Estas son funciones que permiten comparar dos objetos de par en función de sus elementos.

## 7. Prácticas de estructuras de datos en Kotlin

**a. Ejercicios prácticos para aplicar los conceptos aprendidos**

**Arreglos: Numero divisible por N ingresado**

**Listas: Pida tres notas, almacénelas en una lista e indique si la persona gana, pierde o recupera.**

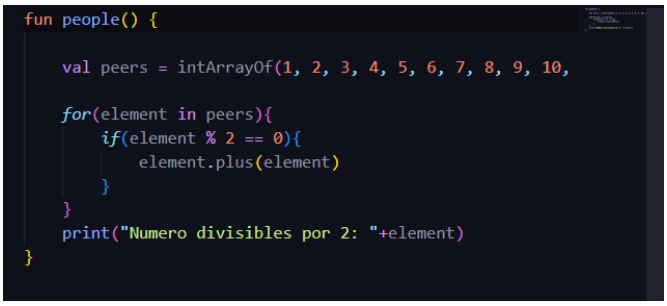
**Conjuntos: Pedirle al usuario ingresar el nombre y la edad de n persona, almacenarlos en mapas distinto e imprimir el nombre con su edad correspondiente.**

**Mapas: Almacene los principales países y sus capitales y pida por consola al usuario ingresar un país y devolver su capital, si no esta decir que no se encontró a capital de ese país.**

**Pares: Por medio de una referencia que identifica la marca de un camión pídasela al usuario por consola e indicar si esta la marca de lo contrario indicar que esa referencia no fue encontrada.**

**b. Solución a los ejercicios prácticos**

**Arreglos:**



```
fun people() {

    val peers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

    for(element in peers){
        if(element % 2 == 0){
            element.plus(element)
        }
    }
    print("Numero divisibles por 2: "+element)
}
```

## Listas:

```
fun  main(){
    println("Ingrese su nombre: ")
    var name = readLine()!!

    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()

    val notas = listOf(nota1, nota2, nota3)

    val sum = notas.sum()

    var prom = sum /3

    if(prom < 2.5){
        println("El estudiante $name Perdio la materia con promedio $prom")
    }else if(prom > 2.6 && prom < 3.4){
        println("El estudiante $name Perdio la materia pero la puede recuperar")
    }else if(prom > 3.5){
        println("El estudiante $name Gano la materia con un promedio de $prom")
    }
}
```

```
Ingrese su nombre:
Jeyson Grisales
Ingrese la nota 1:
4
Ingrese la nota 2:
2
Ingrese la nota 3:
5
El estudiante Jeyson Grisales Gano la materia con un promedio de 3.6666666666666665
```

## Conjuntos:

```
fun main() {
    print("¿Cuántas personas deseas ingresar? ")
    val cantPeople = readLine()!!.toInt()

    val names = mutableSetOf<String>()
    val ages = mutableSetOf<Int>()

    for (i in 1..cantPeople) {
        print("Ingresa el nombre de la persona $i: ")
        val name = readLine()!!
        print("Ingresa la edad de la persona $i: ")
        val age = readLine()!!.toInt()
        names.add(name)
        ages.add(age)
    }

    println("Las personas que ingreso son:")
    for (name in names) {
        val age = ages.elementAt(names.indexOf(name))
        if(age >= 18){
        println("-> La persona $name tiene $age y es MAYOR de edad")
        }else{
        println("-> La persona $name tiene $age y es MENOR de edad")
        }
    }
}
```

```
¿Cuántas personas deseas ingresar? 3
Ingresa el nombre de la persona 1: Jeyson Grisales
Ingresa la edad de la persona 1: 19
Ingresa el nombre de la persona 2: Jorge Valencia
Ingresa la edad de la persona 2: 18
Ingresa el nombre de la persona 3: Jhonatan Melo
Ingresa la edad de la persona 3: 22
Las personas que ingreso son:
-> La persona Jeyson Grisales tiene 19 y es MAYOR de edad
-> La persona Jorge Valencia tiene 18 y es MAYOR de edad
-> La persona Jhonatan Melo  tiene 22 y es MAYOR de edad
```

## Mapas:

```
fun main(){
    var countries = mapOf(
        "COLOMBIA" to "Bogota",
        "ESPAÑA" to "Madrid",
        "VENEZUELA" to "Caracas",
        "ARGENTINA" to "Buenos Aires",
        "ECUADOR" to "Quito",
        "CHILE" to "Santiago de Chile",
        "BRAZIL" to "Brazilia",
        "MEXICO" to "Ciudad de Mexico",
        "ESTADOS UNIDOS" to "Washington D.C."
    )


    print("Ingrese el nombre de un pais: ")
    var nameCountries = readLine()!!.toUpperCase()

    var capitalCountries = countries[nameCountries]

    if(capitalCountries != null){
        print("La capital de $nameCountries es: $capitalCountries")
    }else{
        print("No se encontro la capital de $nameCountries")
    }
```

```
Ingrese el nombre de un pais: Colombia
La capital de COLOMBIA es: Bogota
```

```
Ingrese el nombre de un pais: Rusia
No se encontro la capital de RUSIA
```

## Pares:

```
fun main(){
    val truckers = listOf(
        Pair("R-101", "Volvo"),
        Pair("R-102", "Mercedes-Benz"),
        Pair("R-103", "Scania"),
        Pair("R-104", "MAN"),
        Pair("R-105", "Iveco")
    )

    print("Ingresa la reference del camión que deseas buscar: ")
    val reference = readLine()

    val mark = truckers.find { it.first == reference }?.second

    if (mark != null) {
        println("La Marca del camión con reference $reference es $mark")
    } else {
        println("No se encontró la reference $reference")
    }
}
```

```
Ingresa la reference del camión que deseas buscar: R-103
La Marca del camión con reference R-103 es Scania
```

```
Ingresa la reference del camión que deseas buscar: R-106
No se encontró la reference R-106
```