```
package list

fun main(){
    val mylist = Mylist()
    val numbers =mylist.list1()
    val thirdElement = numbers[2] //23. elem megkeresése a listában

    val fruits:List<String> = mylist.fruits()
    val listSize = fruits.size
    val containsBanana = fruits.contains("Banana")
    val indexOfOrange = fruits.indexOf("Orange")

    val colors :MutableList<String> = mylist.colors()



    println("A lista eredményei: $numbers")
    println()//uj sor
    println("A lista eredményei:\n${numbers.joinToString ("\n") }") //uj sorba
irja
    println()//uj sor
    println("A lista harmadik eleme: $thirdElement")
    println()//uj sor
    println("Gyümölcsök : $fruits")
    println("Gyümölcsök egymás alá írva")
    println("Gyümölcsök :\n${fruits.joinToString("\n")}")
    println()//uj sor
    println("Gyümölcsök Lista mérete:$listSize")//hány elem található benne
    println("A banán megtalálható benne: $containsBanana")
    println("A narancs hanyadik elem a listában: $indexOfOrange")
    println()
    println("Színek : ${colors.joinToString()}")//ha nem írsz a joinToStringbe
semmit
    //eltünteti a [] zárójelet
    colors.remove("Blue")
    println("A kék szín eltávolítása : ${colors.joinToString()}")
    //colors.add("Blue")
    //colors.add("Orange")
    //vagy külön vagy listában adod hozzá a színeket
    colors.addAll(listOf("Blue", "Orange"))
    //eltünteti a [] zárójelet
    println("Színek hozzáadva : ${colors.joinToString()}")
    println("Ez volt mutableList !")
}
```

```
package list

import kotlin.collections.List

class Mylist {
    fun list1(): List<Int> {
      return  listOf(1, 2, 3, 4, 5, 6)
```

```
    }
    fun fruits(): List<String> {
          return listOf("Apple", "Banana","Orange","Apple")
    }
    fun colors() :MutableList<String> {
        val myColors:MutableList<String> =  mutableListOf("Red", "Green",
"Blue")
        return  (myColors)
    }

}
```

✅ 1. Fájl: `main.kt` (főprogram)

🔹 Listák használata

- `numbers = mylist.list1()`
  → Visszaad egy `List<Int>` típusú számlistát: `[1, 2, 3, 4, 5, 6]`
  → `numbers[2]` a harmadik elem → értéke `3` ✅
- `fruits = mylist.fruits()`
  → `List<String>`: `["Apple", "Banana", "Orange", "Apple"]`
  → `fruits.contains("Banana")` ✅
  → `fruits.indexOf("Orange")` = `2` ✅

🔹 MutableList használat (a színek)

```
val colors: MutableList<String> = mylist.colors()
```

- `colors` egy módosítható lista → `["Red", "Green", "Blue"]`
- `colors.remove("Blue")` → eltávolítja a `"Blue"` elemet
- `colors.addAll(listOf("Blue", "Orange"))` → újra hozzáadja a `"Blue"`-t, plusz `"Orange"`-t

Végeredmény:

```
["Red", "Green", "Blue", "Orange"]
```

📩 Kimenet (output):

A programod végig:

```
A lista eredményei: [1, 2, 3, 4, 5, 6]

A lista eredményei:
1
2
3
4
5
```

```
6

A lista harmadik eleme: 3

Gyümölcsök : [Apple, Banana, Orange, Apple]
Gyümölcsök egymás alá írva
Gyümölcsök :
Apple
Banana
Orange
Apple

Gyümölcsök Lista mérete:4
A banán megtalálható benne: true
A narancs hanyadik elem a listában: 2

Színek : Red, Green, Blue
A kék szín eltávolítása : Red, Green
Színek hozzáadva : Red, Green, Blue, Orange
Ez volt mutableList !
```

✅ 2. Fájl: `Mylist.kt` (adatszolgáltató osztály)

Három jól elkülönített metódus:

```kotlin
fun list1(): List<Int> {
    return listOf(1, 2, 3, 4, 5, 6)
}
fun fruits(): List<String> {
    return listOf("Apple", "Banana", "Orange", "Apple")
}
fun colors(): MutableList<String> {
    return mutableListOf("Red", "Green", "Blue")
}
```

- A `list1()` és `fruits()` csak olvasható `List`, nem lehet módosítani (nem tudsz `add` vagy `remove`-olni).
- A `colors()` már `MutableList`, így **lehet módosítani** a listát.

🌟 Értékelés / Fejlesztési javaslat

Rész	Értékelés	Javaslat
Kódszerkezet	✅ Szép, olvasható	Csoportosíthatod a `println` sorokat blokkokba
Listakezelés	✅ Jó példa `List` vs `MutableList`	Esetleg próbáld ki `distinct()`, `sort()`, `reverse()`
Funkcionalitás	✅ Működik minden	Később kipróbálhatod `map`, `filter`, `groupBy` is

---

🧪 Bónusz ötlet (próba):

Ha érdekel, kipróbálhatod a következőt:

```
val uniqueFruits = fruits.distinct()
println("Egyedi gyümölcsök: ${uniqueFruits.joinToString()}")
```

Következő példa:

```
package sets

val uniqueNumber : Set <Int> = setOf(1,2,3,4,5,3,2,1)

fun setOperations(){
    val colors :Set <String> = setOf("Red","Green","Blue")
    val setSize:Set <String> =colors
    val containsRed : Boolean = colors.contains("Red")
    val isempty :Boolean = colors.isEmpty()
    val fruits :MutableSet <String> = mutableSetOf("Apple","Banan","Orange")
    println("Colors is empty? : $isempty")
    println()//uj sor
    println("Colors is set size? $setSize")
    println()
    println("Set of colors : $colors")
    println()//uj sor
    println("Contains Red ? : $containsRed")
    println()
    println("Fruits : $fruits")
    fruits.add("Grapes")
    println("Add Grapes to fruits : $fruits ")
    fruits.removeAll(listOf("Banana","Apple"))
    println("Remove some fruits : $fruits")
}

fun main(){

    println("OriginalSet : $uniqueNumber") //ki iratjuk a szamok listat
    println()//uj sor
    //ki iratjuk a setOperations-t
    val operations =Operations()
    val myList2: List<String> = operations.mySetOperations()
```

```
    val formated: String = myList2.joinToString("\n")
    println("Operations kt adatai : $formated")
}
```

```
package sets

class Operations {
    fun mySetOperations(): List<String> {
        val set1: Set<Int> = setOf(1,2,3,4,5,6,7)
        val set2: Set<Int> = setOf(8,1,9,10)
        //most a közös tartalmukat ellenörizük
        val unionSet: Set<Int> = set1.union(set2)
        //
        val intersectionSet: Set<Int> = set1.intersect(set2)
        //
        val differentSet: Set<Int> = set1.subtract(set2)
        // vissza adjuk a listák értékét mapOf-al return mapOf-al
        return listOf  (
            "set1 : $set1",
            "set2 : $set2",
            "union : $unionSet",
            "intersection : $intersectionSet",
            "differentSet : $differentSet"
            )
    }
}
```

elemzés:

---

1. `val uniqueNumber : Set <Int> = setOf(1,2,3,4,5,3,2,1)`

- **Set** típusú változó, amely egy halmazt tárol, azaz nem tartalmaz ismétlődő elemeket.
- Itt az ismétlődő számok (`3, 2, 1`) csak egyszer jelennek meg, mert a Set automatikusan kiszűri a duplikátumokat.
- Az eredmény: `uniqueNumber = [1, 2, 3, 4, 5]` (bár a Set nem garantál sorrendet).

---

2. `fun setOperations(){ ... }`

Ez egy függvény, amely különböző `Set` és `MutableSet` műveleteket mutat be:

- **colors**: immutable `Set` ("Red", "Green", "Blue")
- `setSize`: Itt hibásan egy Set-et adsz meg egy változóként, amelynek a neve azt sugallja, hogy szám kellene legyen (méret). Jó lenne `val setSize = colors.size` legyen.
- `containsRed`: boolean, hogy tartalmazza-e a "Red" elemet (true)
- `isempty`: boolean, hogy üres-e a halmaz (false)
- **fruits**: mutable halmaz, amelyhez lehet elemeket hozzáadni, vagy törölni.
- Két mutációs művelet:

- hozzáad "Grapes"-t
- eltávolítja a "Banana" és "Apple" elemeket

A függvény `println`-nel írja ki a változásokat.

---

3. `fun main(){ ... }`

- Kiírja az `uniqueNumber` halmaz elemeit
- Létrehoz egy `Operations` objektumot
- Meghívja annak `mySetOperations()` metódusát, ami egy listát ad vissza
- A visszakapott listát `joinToString("\n")` segítségével egyetlen szöveggé alakítja úgy, hogy minden elem új sorba kerüljön
- Kiírja ezt formázott listát

---

4. `class Operations { fun mySetOperations(): List<String> { ... } }`

Ez a függvény valójában egy sor halmazműveletet végez:

- Két halmaz: `set1` és `set2`
- A `unionSet` a két halmaz egyesített elemei (összes elem, ismétlődések nélkül)
- Az `intersectionSet` a két halmaz közös elemei (metszet)
- A `differentSet` az `set1` elemei, amelyek nincsenek benne `set2`-ben (különbség)

A metódus **egy listát ad vissza**, ahol minden string egy adott halmaz eredményét mutatja (pl. `"union : $unionSet"`).

Összefoglalás

- Ez a kód **halmazokkal** (Set) dolgozik.
- Megmutatja az alapvető műveleteket: unió, metszet, különbség.
- A `mySetOperations` metódus nem ad vissza egyetlen halmazt, hanem szövegesen (stringként) összefoglalt listát ad vissza.
- A `main` függvény szépen formázottan kiírja ezt a listát.

Apró javaslatok, javítások

1. A `setSize` változónál:

```
val setSize = colors.size // int típus, ne Set!
println("Colors size is: $setSize")
```

1. Az elnevezésekben lehetne következetesebb, pl. magyarul vagy angolul, hogy áttekinthetőbb legyen.
2. Ha szeretnéd kiírni a halmazokat szebben, akkor a `joinToString`-et is alkalmazhatod:

```
println("set1 : ${set1.joinToString(", ")}")
```

Második kód:

```
package map

fun main(){

    val myMap = Map1()
    val fruitPriceList: Map<String, Double> = myMap.fruitPrices()
    val myList:Map<String, List<String>> = myMap.mySettList
        println(fruitPriceList)
    val carModels:Map<String, String> = myMap.mapOperation()
    val student : MutableMap<String, Int> = myMap.mapModification()
    val country: Map<String, Int> =myMap.mapIteration()

    println()// uj sor
    //stringé alakitjuk jobban egymás alatt megjelenítjük
    val formattedList:List<String> = fruitPriceList.map { "${it.key}:
${it.value}" }
    println(formattedList.joinToString("\n"))
    //vagy
    val formattedList2: String =fruitPriceList.map { (k:String, v:Double) ->
        "$k: $v" }.joinToString("\n")
    println()//uj sor
    println(formattedList2)
    println()//uj sor
    //Stringé alakitjuk
    val formattedList3:String = myList.map { (_:String , value:List<String>) ->
        value.joinToString()
    }.joinToString("\n")
    println(formattedList3)
    println()//uj sor
    println("Car models:$carModels ")
    println()//uj sor
    val carmodelformatedList = carModels.map { (k ,v) ->
        "${k.replace("{","")} ${v.replace("}","")}"}
    //a replace függvény 2 karaktert vár azért kel a zárojel megvonása után egy
üres "" string
    println(carmodelformatedList.joinToString("\n"))
    println()//uj sor
    println("Tanulók és jegyeik:")
    println(student.map { (k,v) ->"$k: $v" }.joinToString ("\n"))
    // a kután kex után a : hozzáadja a a listához és akkor a kimenet
    //nem Alice 95 hanem Alice : 95
    student["Bob"] = 106
    println("New Student add: $student")
    println()//uj sor
    println("Országok néppeségeinek száma:")
    println(country.map { (k,v) ->"$k $v" }.joinToString("\n"))

    }
```

```
package map

class Map1 {
   fun fruitPrices(): Map<String, Double> = mapOf(
      "Apple " to 1.0,
      "Banana" to 0.75,
      "Orange" to 1.25
   )

   val mySettList: Map<String, List<String>> = mapOf(
      "set1" to listOf("1", "2", "3"),
      "set2" to listOf("4", "5", "6")
   )

   fun mapOperation(): Map<String, String>{
   val carModel: Map<Int, String> = mapOf(
      2022 to "Model Y", 2021 to "Model 3",
      2020 to "Model S"
   )
   val carSize: Int = carModel.size
   val contains2021:Boolean = carModel.containsKey(2021)
   val containsModel = carModel.containsValue("Model S")

   return mapOf(
   "Carmodel" to carModel.toString(),
   "Carsize" to carSize.toString(),
   "Contains 2021" to contains2021.toString(),
   "ContainsModell" to containsModel.toString()
   )
}
   fun mapModification(): MutableMap<String, Int> {
      val studentGrades:MutableMap<String,Int>
      = mutableMapOf("Alice" to 95,"Bob" to 87,"Charlie" to 92)
      //println("OriginaList: $studentGrades")
      studentGrades["David"] = 88
      //println("Add-ed list: $studentGrades")
      studentGrades.remove("Bob")
      //println("Remove List: $studentGrades")
      return studentGrades
   }
   fun mapIteration(): Map<String, Int> {
      val countryPopulation:Map<String,Int> =
         mapOf("Usa" to 331,"China" to 1441,"India" to 1393)
      for ((country,population) in countryPopulation){
         println("Population to $country : $population million")
      }
      return countryPopulation
   }

}
```

elemzés:

Persze, nézzük át lépésenként a kódodat, hogy teljesen tiszta legyen, mit csinál és hogyan működik!

1. `Map1` osztály

a) `fruitPrices()`

```
fun fruitPrices(): Map<String, Double> = mapOf(
    "Apple " to 1.0,
    "Banana" to 0.75,
    "Orange" to 1.25
)
```

- Visszaad egy **immutábilis Map-et** (kulcs: gyümölcs neve, érték: ár).
- Kulcsok String, értékek Double típusúak.

---

b) `mySettList`

```
val mySettList: Map<String, List<String>> = mapOf(
    "set1" to listOf("1", "2", "3"),
    "set2" to listOf("4", "5", "6")
)
```

- Egy `Map`, ahol a kulcs egy String (`"set1"`, `"set2"`), az érték pedig egy String lista (`List<String>`).
- Pl. `"set1"` kulcshoz tartozó lista: `["1", "2", "3"]`

---

c) `mapOperation()`

```
fun mapOperation(): Map<String, String> {
    val carModel: Map<Int, String> = mapOf(
        2022 to "Model Y", 2021 to "Model 3",
        2020 to "Model S"
    )
    val carSize: Int = carModel.size
    val contains2021: Boolean = carModel.containsKey(2021)
    val containsModel = carModel.containsValue("Model S")

    return mapOf(
        "Carmodel" to carModel.toString(),
        "Carsize" to carSize.toString(),
        "Contains 2021" to contains2021.toString(),
        "ContainsModell" to containsModel.toString()
    )
}
```

- Létrehoz egy `carModel` Map-et év és autó modell párokkal.
- Lekéri a méretét, ellenőrzi, hogy tartalmaz-e bizonyos kulcsot és értéket.
- Végül mindezt String formában visszaad egy új Map-ként, ahol kulcsok szöveges címkék, értékek pedig a fentiek Stringgé alakított változatai.

---

d) `mapModification()`

```
fun mapModification(): MutableMap<String, Int> {
    val studentGrades: MutableMap<String, Int> = mutableMapOf(
        "Alice" to 95, "Bob" to 87, "Charlie" to 92
    )
    studentGrades["David"] = 88  // hozzáad egy új elemet
    studentGrades.remove("Bob")   // eltávolít egy elemet

    return studentGrades          // visszaadja a módosított térképet
}
```

- MutableMap-ot használ, azaz módosítható map.
- Hozzáad egy új diákot `"David"` névvel és 88 ponttal.
- Eltávolítja `"Bob"`-ot.
- Visszaadja a végső állapotot.

---

e) `mapIteration()`

```
fun mapIteration(): Map<String, Int> {
    val countryPopulation: Map<String, Int> = mapOf(
        "Usa" to 331, "China" to 1441, "India" to 1393
    )
    for ((country, population) in countryPopulation) {
        println("Population to $country : $population million")
    }
    return countryPopulation
}
```

- Egyszerűen végigmegy a `countryPopulation` térképen, és kiírja az ország nevét és népességét.
- Visszaadja a Map-et.

---

2. `main()` függvény

a) Objektum létrehozás és adatok lekérése

```
val myMap = Map1()
val fruitPriceList: Map<String, Double> = myMap.fruitPrices()
val myList: Map<String, List<String>> = myMap.mySettList
println(fruitPriceList)
val carModels: Map<String, String> = myMap.mapOperation()
val student: MutableMap<String, Int> = myMap.mapModification()
val country: Map<String, Int> = myMap.mapIteration()
```

- Létrehozol egy példányt a Map1 osztályból.
- Lekéred a különböző térképeket a metódusokból.
- Kiírod a `fruitPriceList`-et (ez alapértelmezett `toString()` formában jelenik meg).

---

b) Formázott kiírás

```
val formattedList: List<String> = fruitPriceList.map { "${it.key}: ${it.value}"
}
println(formattedList.joinToString("\n"))
```

- Átalakítod a gyümölcsök árát egy listává, ahol minden elem "Kulcs: Érték" formában van.
- Kiírod őket egymás alá.

```
val formattedList2: String = fruitPriceList.map { (k: String, v: Double) -> "$k:
$v" }.joinToString("\n")
println(formattedList2)
```

- Ugyanaz, csak kicsit más szintaxissal.

```
val formattedList3: String = myList.map { (_: String, value: List<String>) ->
value.joinToString() }.joinToString("\n")
println(formattedList3)
```

- A `mySettList` értékeit alakítod String-é, és minden lista elemeit vesszővel elválasztva írod ki.

---

c) Autó modellek kiírása

```
println("Car models:$carModels ")
println()

val carmodelformatedList = carModels.map { (k, v) ->
    "${k.replace("{", "")} ${v.replace("}", "")}"
}
println(carmodelformatedList.joinToString("\n"))
```

- Kiírod a `carModels` Map stringként.
- Utána próbálsz `"{"` és `"}"` karaktereket eltávolítani a kulcsból és értékből (ez felesleges, mert a kulcs és érték nem tartalmaz zárójeleket, csak a string reprezentáció).
- Majd külön sorokban kiírod az elemeket.

---

d) Diákok jegyei és módosítás

```
println("Tanulók és jegyeik:")
println(student.map { (k, v) -> "$k: $v" }.joinToString("\n"))

student["Bob"] = 106
println("New Student add: $student")
```

- Kiírod az aktuális diákokat és pontjaikat.
- Ezután `"Bob"` értékét 106-ra módosítod (vagy hozzáadod, ha nem volt).
- Újra kiírod a módosított `student` térképet.

e) Országok népessége

```
println("Országok néppeségeinek száma:")
println(country.map { (k, v) -> "$k $v" }.joinToString("\n"))
```

- Kiírod az országokat és népességüket egymás alá.

Összegzés

- A kódod tiszta, jól strukturált.
- Map-ek létrehozása, módosítása, lekérdezése, és formázott kiírás egyaránt megvalósított.
- `student["Bob"] = 106` helyesen módosítja az értéket.
- A `replace("{", "")` a string-ekben felesleges, mert a kulcsok és értékek nem tartalmaznak zárójeleket.
- A kommentek segítik az olvashatóságot.