```
package oops

fun main() {
    //Példányosítás
    val person1 =Person("John", 25)//osztály meghivva deklarálva person1
    val person2 = Person("Jane", 38)//osztály meghivva deklarálva person2
    //példányosítás2

    val example =Example()
    println()//uj sor
    //példányositás3
    val student1 = Student("Alice", 21)
    val student2 = Student("Kartik", ) //18 lesz automatikusan a kora mert
megadtuk aStudent.kt-ban

    student1.displayInfo()
    student2.displayInfo()


    println() //uj sor
    println("${person1.name} is ${person1.age} years old")
    println("${person2.name} is ${person2.age} years old")
}
```

Main.kt

```
package oops

class Student(val name:String, val age:Int = 18) {
    init {
        println("Student Initialized = Name : $name, Age : $age")
    }

    fun displayInfo(){
        println("Student Information - Name : $name, Age : $age ")
    }
}
```

Student.kt

```kotlin
package oops

class Example {
    init {
        println("Initialization Block 1")
    }
    val name :String="Kotlin"

    init {
        println("Initialization Block 2")
        println("Name : $name")
    }
}
```

Example.kt

```kotlin
package oops

class Person(val name:String, val age :Int)
// vagy class Person(val name: String, val age: Int) konstruktor hozzáadása
//es a main fajlban akkor val person1 = Person("John", 25)
//val person2 = Person("Jane", 38)
//vagy data class
//data class Person(val name: String, val age: Int)
//A main fajlban println(person1) // automatikusan: Person(name=John, age=25)
```

Person.kt

🧠 Mi az `init` blokk?

Az `init` blokk a Kotlinban egy **osztály inicializáló blokkja**, amely automatikusan lefut **akkor, amikor egy példányt hozol létre az osztályból** (`val obj = ClassName()`).

Gyakorlatilag úgy viselkedik, mint egy mini-konstruktor, és **bármilyen kezdeti műveletet, logikát, nyomkövetést vagy változóértékadást** el lehet benne helyezni.

🧱 Alapszintű szintaxis:

```kotlin
class MyClass(val name: String) {

    init {
        println("Init blokk fut: $name")
    }
}
```

💡 Mire használjuk?

1. **Logolás** – ellenőrizhetjük, mikor hozunk létre példányokat.
2. **Értékellenőrzés** – például ha a kapott érték nem lehet negatív, kidobhatunk hibát.
3. **Változók beállítása**, ha azok kiszámítása bonyolultabb.
4. **Függőségek, állapot előkészítése**, stb.

🛠️ Több `init` blokk is lehet?

Igen! Az `init` blokkok **fentről lefelé sorrendben futnak le**, amikor példányosítasz.

```
class Test {
    init {
        println("Első blokk")
    }

    init {
        println("Második blokk")
    }
}
```

Kimenet:

```
Első blokk
Második blokk
```

🔁 Init vs Konstruktor

Konstruktor	Init blokk
Paramétereket vesz át	Nem vesz fel paramétert
Objektum létrehozásakor fut	Objektum létrehozásakor fut
Nem lehet több belőle (csak egy primary, és több secondary lehet)	Több `init` blokk lehet
Rögtön az elején van	Bárhol lehet az osztályon belül

📦 Összefoglalás

- Az `init` blokk **nem helyettesíti** a konstruktort, hanem **kiegészíti**.
- Bármilyen inicializáló logikát tehetsz ide.
- Több `init` blokkod is lehet egy osztályon belül.
- Akkor fut, amikor az osztályból példányt hozol létre.
- **Sorrendje az osztály definíció sorrendjét követi.**

📌 Példa – Komplex inicializálás:

```
class User(name: String, age: Int) {
    val username: String

    init {
        println("Név: $name")
    }

    init {
        username = name.lowercase()
        println("Felhasználónév beállítva: $username")
    }
}
```