```
//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
fun main(args: Array<String>) {

    //var age :Int = 25
    var age = 25 //automatikusan tudja a fordító hogy mibe forditson
    //azaz hogy Int-be
    //var weight : Double = 65.5
    var weight = 65.5
    //var numbers : List<Int> = listOf(1, 2, 3 ,4 ,5)
    var numbers = listOf(1, 2, 3 ,4 ,5)

    //var nullableName : String? = null
    var nullableName = null
}
/*
Előnyök az automatikus típusmegállapításnál (`type inference`):
1. Kevesebb kód:
Nem kell mindig kiírni a típust, így a kód rövidebb és olvashatóbb lesz:
val name = "Ati" // a fordító automatikusan tudja, hogy String
Gyorsabb írás,Gyorsabb fejlesztés, főleg egyszerű értékadásnál.
Tisztább szintaxis:
Nem kell ismételni magad, amikor a típus egyértelmű:
val numbers = listOf(1, 2, 3) // List<Int>, nem kell külön kiírni
Modern nyelvi stílus:
Illeszkedik a Kotlin "smart compiler" filozófiájához – a nyelv megpróbálja minél
jobban segíteni a programozót.
Hátrányok:
1. Kevésbé egyértelmű típus, ha nem nyilvánvaló*:
Nem mindig látszik első ránézésre, hogy milyen típusú egy változó.
val something = getSomething() // ?? milyen típus? – lehet találgatni
2. Nullable típus nem mindig egyértelmű:
val name = null // Ez nem `String?`, hanem `Nothing?` lesz!
Ez **nem lesz `String?`, hanem `Nothing?`, amit később nem lehet csak úgy más
típusra használni.
Ha ezt akarod:
var name: String? = null // A típusra itt szükség van, különben a fordító nem
tudja, hogy mit vársz.
Generikus típusoknál veszélyes lehet:
val list = listOf() // Ez List<Nothing> lesz!

 */
```