```
package declaringAndInvokingFunctions

fun greet(name : String){
    println("Hello $name")
}

fun main() {
    //példányosítás
    val calculate =Calculate()
    val person = Person()
    val sumResult = calculate.calculatesum(5,7)//összegEredmény , összeg
sumResult
    val productResult =calculate.multiply(4,6)//szorzateredmény



    greet("Kotlin Learners")
    println()//uj sor
    println("sum result : $sumResult ")
    println()
    println("Product : $productResult")
    println()
    person.greatPerson("Alice", "Hi")
    }
```

```
package declaringAndInvokingFunctions

class Calculate {
    fun calculatesum(a: Int, b: Int): Int {
        return a + b
    }
    fun multiply(x: Int, y: Int): Int = x * y
}
```

```
package declaringAndInvokingFunctions

class Person {
    fun greatPerson(name :String, greeting: String ="Hello"){

        println("$greeting, $name ")
    }
}
```

---

1. `greet` függvény

```
fun greet(name : String){
    println("Hello $name")
}
```

- Ez egy egyszerű függvény, ami kap egy `name` (név) paramétert.
- Kiírja a konzolra: `"Hello <name>"`.
- Pl. ha `greet("Kotlin Learners")` hívod meg, akkor a konzolon ez jelenik meg: `Hello Kotlin Learners`

---

2. `Calculate` osztály

```
class Calculate {
    fun calculatesum(a: Int, b: Int): Int {
        return a + b
    }
    fun multiply(x: Int, y: Int): Int = x * y
}
```

- Ez egy osztály, aminek két metódusa van:
  - `calculatesum(a: Int, b: Int)` — két egész számot összead, és visszaadja az összeget.
  - `multiply(x: Int, y: Int)` — két egész szám szorzatát adja vissza.

Pl.:

```
val calculate = Calculate()
val sum = calculate.calculatesum(5, 7)  // 12
val product = calculate.multiply(4, 6)   // 24
```

---

3. `Person` osztály

```
class Person {
    fun greatPerson(name :String, greeting: String ="Hello"){
        println("$greeting, $name ")
    }
}
```

- Ez az osztály egyetlen metódust tartalmaz: `greatPerson`.
- Ez a függvény két paramétert vár:
  - `name`: személy neve (kötelező)
  - `greeting`: üdvözlés szövege (alapértelmezett: `"Hello"`)
- Kiírja a konzolra a köszönést és a nevet, pl.: `"Hi, Alice"` vagy ha nincs megadva a második paraméter, akkor `"Hello, Alice"`

---

4. `main` függvény (a program belépési pontja)

```
fun main() {
    val calculate = Calculate()       // létrehoz egy Calculate objektumot
    val person = Person()             // létrehoz egy Person objektumot
```

```
    val sumResult = calculate.calculatesum(5,7)   // összeadás eredményét
tárolja (12)
    val productResult = calculate.multiply(4,6)   // szorzás eredményét tárolja
(24)

    greet("Kotlin Learners")          // meghívja a greet függvényt, kiírja:
Hello Kotlin Learners
    println()                         // új sor
    println("sum result : $sumResult ")  // kiírja: sum result : 12
    println()                         // új sor
    println("Product : $productResult")  // kiírja: Product : 24
    println()                         // új sor
    person.greatPerson("Alice", "Hi")   // kiírja: Hi, Alice
}
```

Összefoglalás

- A program létrehoz két osztályt: `Calculate` és `Person`.
- A `Calculate` osztály segítségével összeadás és szorzás műveleteket hajtasz végre.
- A `Person` osztály egy egyszerű üdvözlő szöveget ír ki.
- A `greet` függvény szintén kiír egy egyszerű üdvözlést.
- A `main` függvényben meghívod ezeket a műveleteket, majd az eredményeket kiírod a konzolra.

```
package localFunctions

fun main(){
    val calculategrade =CalculateGrade()
    val result = calculategrade.calculateGrade(85)
    println(result)
}
```

```
package localFunctions

class CalculateGrade {
    fun calculateGrade(score: Int): String {

        fun determineGrade(): String {
            return when (score) {
                in 90..100 -> "A"
                in 80 until 90 -> "B"
                in 70 until 80 -> "C"
                in 60 until 70 -> "D"
                else -> "F"
            }
        }

        return "Grade : ${determineGrade()}"
    }
}
```

Rendben, Ati! Itt van a kódod **világos és lépésenkénti összefoglalása** magyarul:

📜 Mit csinál ez a Kotlin program?

Ez a program **pontszám alapján kiszámítja egy tanuló osztályzatát (A–F)**, majd kiírja a konzolra az eredményt.

🔧 A program működése lépésenként:

1. `main()` függvény:

```
fun main(){
    val calculategrade = CalculateGrade()              // példányosítjuk az
osztályt
    val result = calculategrade.calculateGrade(85)     // meghívjuk az osztály
metódusát 85 ponttal
    println(result)                                    // kiírjuk az eredményt a
konzolra
}
```

- Létrehoz egy példányt a `CalculateGrade` osztályból.
- Meghívja rajta a `calculateGrade(85)` metódust.
- A visszakapott osztályzatot szöveges formában kiírja:
  például: `Grade : B`

---

2. `CalculateGrade` osztály:

```
class CalculateGrade {
    fun calculateGrade(score: Int): String {

        fun determineGrade(): String {
            return when (score) {
                in 90..100 -> "A"
                in 80 until 90 -> "B"
                in 70 until 80 -> "C"
                in 60 until 70 -> "D"
                else -> "F"
            }
        }

        return "Grade : ${determineGrade()}"
    }
}
```

- Ez az osztály egy metódust tartalmaz: `calculateGrade(score: Int): String`
- A metódusban van egy **helyi függvény**: `determineGrade()`, ami belülről osztályoz:
  - 90–100 pont → A
  - 80–89 pont → B

- 70–79 pont → C
- 60–69 pont → D
- Minden más → F

- A `calculateGrade()` visszaadja ezt az eredményt ilyen formában:
  pl. `"Grade : B"`

✅ Futási példa:

Ha a bemenet `85`, akkor:

```
Grade : B
```

✅ Javaslatok (ha szeretnéd szebbé tenni):

- Osztálynevek: `CalculateGrade` → rendben van, mert nagybetűs.
- Függvények neve: `calculateGrade()` → kisbetűs, helyes Kotlin konvenció.
- Ha szeretnéd magyarítani:
  `CalculateGrade` → `Ertekeles`
  `calculateGrade()` → `osztalyzatotSzamit()` stb.

`determineGrade` = „osztályzat meghatározása” vagy „osztályzat megállapítása”

Pontosan:

- `determine` = meghatározni, megállapítani
- `grade` = osztályzat (jegy)
-

---

következő:

```
class Greeatings {
    fun printDetails(name :String, age:Int){//itt fogjuk megadni a mainben az
adatokat
        fun displayMessage(greeating:String){
            println("$greeating , $name You are $age years old.")
        }
        displayMessage("Hello") //ezt fogja hozzaadni
    }
}
```

```
fun main(){
    val greeatings =Greeatings()
    greeatings.printDetails("Alice", 25)

}
```

Ez a Kotlin kód egy **lokális függvény** (azaz függvény egy másik függvényen belül) használatát mutatja be egy osztályon belül. Lássuk részletesen:

🔍 Mit csinál a kód?

`main()` függvény:

```
val greeatings = Greeatings()
greeatings.printDetails("Alice", 25)
```

- Példányosít egy `Greeatings` nevű osztályt.
- Meghívja rajta a `printDetails` metódust, átadva a `"Alice"` nevet és `25` életkort.

`Greeatings` osztály:

```
class Greeatings {
    fun printDetails(name: String, age: Int) {
        fun displayMessage(greeating: String) {
            println("$greeating, $name You are $age years old.")
        }
        displayMessage("Hello")
    }
}
```

- A `printDetails` függvény két paramétert kap: `name` (név) és `age` (életkor).
- **Ezen belül** van egy belső/lokális függvény: `displayMessage`, amely kiírja az üdvözlő szöveget, a nevet és az életkort.
- Ez a belső függvény hozzáfér a külső függvény paramétereihez (`name`, `age`).
- A `displayMessage("Hello")` meghívásával az eredmény:

  ```
  Hello, Alice You are 25 years old.
  ```

💡 Összefoglalás:

- A program egy `Greeatings` nevű osztályt használ, hogy egy személy nevét és életkorát egy üdvözlő szöveggel együtt kiírja.
- A `printDetails` metódus tartalmaz egy **lokális függvényt**, ami csak azon belül érhető el.
- Ez jó példa **lokális függvények** használatára a belső logika elkülönítéséhez.

```
package passingFunctionAsParameters

fun main() {

    val resultSquare= executeOperation(5,::square)
    val resultDouble= executeOperation(7,::square)
```

```
    println("Square Result : $resultSquare")
    println("Double Result : $resultDouble")


}

fun executeOperation(value:Int, operation : (Int) -> Int) :Int{
    return operation(value)
}
fun  square(x:Int) :Int{
    return x*2
}
```

Részletes leírás:

Nagyon jó kérdés, Ati! Ez a Kotlin kód egy **függvényt ad át paraméterként** egy másik függvénynek, vagyis a **függvény mint adat** elvét mutatja be. Lépésenként és részletesen elmagyarázom:

📌 1. Mi történik itt általánosan?

Ez a program bemutatja, hogyan lehet egy függvényt átadni **paraméterként** egy másik függvénynek Kotlin nyelven. Ehhez használja a `::` operátort, ami egy **függvényreferencia**.

🧠 Kód részletes elemzése

`main()` függvény:

```
val resultSquare = executeOperation(5, ::square)
val resultDouble = executeOperation(7, ::square)
```

- `executeOperation` egy olyan függvény, ami egy `Int` értéket és egy **függvényt** kap paraméterként.
- `::square` azt jelenti, hogy a `square` nevű függvényt **referenciaként** átadjuk.
- Vagyis a `square` függvény maga nem hajtódik végre itt, hanem "csak" átadjuk egy másik függvénynek (mint egy változót).

`executeOperation` függvény:

```
fun executeOperation(value: Int, operation: (Int) -> Int): Int {
    return operation(value)
}
```

Ez a függvény:

- Kap egy `value` nevű egész számot.
- Kap egy `operation` nevű **függvényt**, amelynek típusa: `(Int) -> Int`. Ez azt jelenti, hogy:
    - Egy `Int` típusú értéket fogad,

    - És egy `Int` típusú eredményt ad vissza.
- Meghívja az `operation(value)` hívást, vagyis végrehajtja az átadott függvényt a `value` értékkel.

`square` függvény:

```
fun square(x: Int): Int {
    return x * 2
}
```

Ez a függvény:

- Bemenetként kap egy egész számot `x`,
- Visszaadja ennek a **kétszeresét** (nem négyzetét, tehát nem `x * x`).

> Megjegyzés: A függvény neve `square`, de a tartalma szerint **dupláz**, tehát lehet, hogy át kéne nevezni `double`-re.

🔁 Hogyan fut le?

1. `executeOperation(5, ::square)`:
    - `5` kerül a `value` paraméterbe,
    - `::square` kerül az `operation` paraméterbe,
    - Ez végül `square(5)`-öt jelent, ami `5 * 2 = 10`.
2. `executeOperation(7, ::square)`:
    - `square(7)` = `7 * 2 = 14`.

📤 Kiírás:

```
Square Result : 10
Double Result : 14
```

🔧 Mi az a `::` jel?

- A `::` Kotlinban **function reference** = függvényreferencia.
- Ezzel lehet egy már létező függvényt úgy átadni, mintha az egy **változó** lenne.
- Példa:

```
val op = ::square
println(op(3)) // 3 * 2 = 6
```

📝 Összefoglalás magyarul:

- A `square` függvényt átadjuk az `executeOperation` nevű függvénynek.

- Az `executeOperation` meghívja ezt az átadott függvényt egy számra.
- A `::` operátorral a függvény nevét adjuk át **végrehajtás nélkül**.
- A `(Int) -> Int` azt jelenti, hogy a paraméterként átadott függvény **egész számot fogad el**, és **egész számot ad vissza**.
