## 1. táblázat: `it`, `let`, `forEach`, `?:` (Elvis), `?.`

| Kifejezés | Mit csinál? | Egyszerű példa |
|---|---|---|
| `it` | A lambda kifejezés **implicit paramétere**, az aktuális elem vagy érték | `list.forEach { println(it) }` (itt `it` az adott listaelem) |
| `let` | Ha az érték nem `null`, végrehajt egy blokkot, a blokkban `it` az érték, visszatér az eredménnyel | `val x: String? = "Hello"\nx?.let { println(it) }` |
| `forEach` | Bejár egy kollekciót, minden elemre végrehajtja a blokkot | `listOf(1,2,3).forEach { println(it) }` |
| `?:` | Elvis operátor: ha bal oldali érték `null`, akkor jobb oldali értéket adja vissza | `val length = name?.length ?: 0` (ha `name` null, 0 lesz az eredmény) |
| `?.` | Nullbiztos hívás: csak akkor hajtja végre a metódust, ha nem `null` az objektum | `val len = name?.length` (`name` null esetén nem dob hibát, hanem `null` lesz a `len`) |

## 2. táblázat: `with`, `apply`, `run`, `also`, `is`, `as`, `::`, `in`

| Kifejezés | Mit csinál? | Egyszerű példa |
|---|---|---|
| `with` | Egy objektumot átad a blokkba, ahol `this` az objektum, visszatér a blokk eredményével | `with(user) { println(name); age }` (visszaadja `age` értékét) |
| `apply` | Konfigurál egy objektumot, `this` az objektum, visszatér az objektummal | `val user = User().apply { name = "Ati"; age = 35 }` |
| `run` | Végrehajt egy blokkot, `this` az objektum, visszatér a blokk eredményével | `val len = "Hello".run { length }` |
| `also` | Végrehajt egy blokkot, `it` az objektum, visszatér az eredeti objektum | `val list = mutableListOf<Int>().also { println("List created") }` |
| `is` | Típusellenőrzés | `if (x is String) println("String vagyok")` |
| `as` | Típuskonverzió (runtime cast), hibát dobhat, ha nem kompatibilis | `val s = x as String` |
| `::` | Függvény- vagy property-referencia | `val lenRef = String::length` |
| `in` | Ciklusban vagy tartalmazás ellenőrzésre használható | `for (i in 1..5) println(i)` vagy `if ("a" in list) println("Van a")` |

---

Persze, itt egy rövid táblázat a `when`, `case` és `break` kulcsszavakról Kotlinban, plusz hogy miben különböznek a Java-tól:

| Kulcsszó | Mit csinál? | Kotlin példa | Megjegyzés |
|---|---|---|---|
| `when` | Kotlin `switch` szerű szerkezet, feltételek ellenőrzése | `kotlin\nwhen(x) {\n 1 -> println("Egy")\n 2 -> println("Kettő")\n else -> println("Más")\n}\n` | Nincs `break`, mert nem esik át automatikusan |
| `case` | Java switch-ben egy eset (case) | `switch(x) { case 1: ...; break; }` | Kotlinban nincs `case`, csak `when` |
| `break` | Kilép a ciklusból vagy switchből | Kotlinban ciklusban használható, `when`-ben nem kell | Kotlin `when` nem igényli, mert automatikusan megáll |

---

Jó kérdés! Kotlinban a fő vezérlési szerkezetek és blokkok, amiket gyakran használsz:

| Vezérlési szerkezet | Rövid leírás | Rövid példa |
|---|---|---|
| `for` | Egyszerű ciklus lista vagy range-en | `for (i in 1..5) println(i)` |
| visszafelé `for` | Csökkenő léptetésű ciklus | `for (i in 5 downTo 1) println(i)` |
| mátrix (kétszintes for) | Beágyazott ciklus mátrixhoz | `for (i in 0..2) for (j in 0..2) println("$i,$j")` |
| `while` | Előtesztelő ciklus | `while (x > 0) { x-- }` |
| `while (true)` | Végtelen ciklus | `while(true) { println("Loop") }` |
| `try-catch-finally` | Hibakezelés, utómunka | `kotlin\ntry { ... } catch(e: Exception) { ... } finally { ... }\n` |
| `if-else if-else` | Feltételes elágazás | `kotlin\nif (x > 0) { ... } else if (x == 0) { ... } else { ... }\n` |

| Egyéb blokk | Mit csinál? | Példa |
|---|---|---|
| `do-while` | Legalább egyszer futó ciklus | `kotlin\ndo { ... } while (cond)\n` |
| `when` | Többágú elágazás (switch) | Már említettük korábban |
| `return` | Visszatérés egy függvényből | `return x` |
| `break` | Kilép a ciklusból | `break` |
| `continue` | Következő iterációra lép | `continue` |

| Ciklus típusa | Működés | Példa | Megjegyzés |
|---|---|---|---|
| `while` | Előtesztelő ciklus: előbb ellenőrzöd a feltételt, majd ha igaz, fut a blokk | `kotlin\nvar i = 0\nwhile (i < 3) {\n println(i)\n i++\n}\n` | Ha a feltétel hamis az elején, egyszer sem fut le |
| `do-while` | Utótesztelő ciklus: először lefut egyszer a blokk, aztán ellenőrzöd a feltételt | `kotlin\nvar i = 0\ndo {\n println(i)\n i++\n} while (i < 3)\n` | Legalább egyszer mindig lefut |

operátorok:

Persze! Itt van egy táblázat a Kotlin leggyakoribb **operátorairól** és szimbólumairól, rövid magyarázattal és példákkal:

| Operátor/szimbólum | Jelentés / Mire jó? | Példa | Megjegyzés |
|---|---|---|---|
| `$` | String interpoláció (változó beszúrása sztringbe) | `val name = "Ati"; println("Hi $name")` | String belsejében változó értékét helyettesíti |
| `?` | Null biztonság, nullable típusjelölés, safe call | `val len = str?.length` | Ha `str` null, akkor `len` is null lesz |
| `!!` | Null ellenőrzés kikapcsolása, ha null, kivétel dob | `val len = str!!.length` | Használatakor vigyázni kell, mert kivételt dobhat! |
| `\|\|` | ha az egyik feltétel igaz | `\|\|` | Logikai VAGY (OR) |
| `&&` | Logikai ÉS (AND) | `if (a && b) { ... }` | Igaz, ha mindkettő igaz |
| `&` | Bitenkénti ÉS (bitwise AND) | `val c = a & b` | Egész számokon használatos |
| `=` | Értékadás | `val x = 5` | |
| `+=` | Összeadás és értékadás egyszerre | `x += 2` (ugyanaz, mint `x = x + 2`) | |
| `-=` | Kivonás és értékadás egyszerre | `x -= 1` | |
| `*=` | Szorzás és értékadás egyszerre | `x *= 3` | |
| `/=` | Osztás és értékadás egyszerre | `x /= 4` | |
| `%` | Maradékos osztás (modulus) | `val r = 10 % 3` | `r` értéke 1 lesz |
| `==` | Egyenlőség összehasonlítás | `a == b` | Kotlinban érték szerint hasonlít össze |
| `!=` | Nem egyenlő | `a != b` | |
| `<`, `>`, `<=`, `>=` | Összehasonlító operátorok | `if (x < 5) { ... }` | |

| Operátor/szimbólum | Jelentés / Mire jó? | Példa | Megjegyzés |
|---|---|---|---|
| `++` | Növelés 1-gyel | `x++` | Elő- vagy utó-increment |
| `--` | Csökkentés 1-gyel | `x--` | |
| `!!.` | Nullbiztosítás kikapcsolása metódushíváskor | `str!!.length` | Kivételt dob, ha `str` null |

---

Persze, itt egy összefoglaló táblázat Kotlin alapvető típusokról és operátorokról rövid példákkal:

| Típus | Leírás | Példa deklaráció | Megjegyzés |
|---|---|---|---|
| `Int` | Egész szám (32 bit) | `val x: Int = 42` | Leggyakoribb egész típus |
| `Long` | Nagy egész szám (64 bit) | `val y: Long = 123456789L` | L-es végződés kötelező literálban |
| `Double` | Lebegőpontos szám (64 bit) | `val z: Double = 3.14` | Pontosabb, mint `Float` |
| `Float` | Lebegőpontos szám (32 bit) | `val f: Float = 3.14F` | F végződés kell literálnál |
| `String` | Szöveg típus | `val s: String = "Hello"` | Több karakterből álló szöveg |
| `Char` | Egyetlen karakter | `val c: Char = 'A'` | Egyszerű karakter, aposztróf között |
| `Boolean` | Igaz/hamis érték | `val b: Boolean = true` | `true` vagy `false` |
| `Byte` | Kis egész szám (8 bit) | `val b: Byte = 127` | -128 és 127 között |
| `Short` | Közepes egész szám (16 bit) | `val s: Short = 32767` | -32768 és 32767 között |

---

| Operátor | Jelentés / Mire jó? | Példa | Megjegyzés |
|---|---|---|---|
| `+` | Összeadás / string összefűzés | `3 + 4`, `"Hi " + "Ati"` | Típustól függően működik |
| `-` | Kivonás | `5 - 2` | |
| `*` | Szorzás | `3 * 6` | |
| `/` | Osztás | `10 / 2` | Egész osztás esetén int lesz |
| `%` | Maradékos osztás (modulus) | `10 % 3` | |
| `==` | Egyenlőség összehasonlítás | `a == b` | Érték szerint hasonlít |
| `!=` | Nem egyenlő | `a != b` | |
| `<`, `>` | Kisebb, nagyobb | `x < 5`, `x > 3` | |
| `<=`, `>=` | Kisebb vagy egyenlő, nagyobb vagy egyenlő | `x <= 7`, `x >= 2` | |
| `&&` | Logikai ÉS | `a && b` | Mindkettő igaz kell legyen |
| \|\| | ha az egyik feltétel igaz. | a \|\| b | Logikai VAGY |
| `!` | Logikai NEM | `!a` | Igazból hamis, hamisból igaz |
| `=` | Értékadás | `x = 10` | |
| `+=` | Összeadás és értékadás | `x += 5` | Ugyanaz, mint `x = x + 5` |
| `-=` | Kivonás és értékadás | `x -= 3` | |
| `++` | Növelés 1-gyel | `x++` | Elő- vagy utó-increment |
| `--` | Csökkentés 1-gyel | `x--` | |

`!= nem egyenlő`

---

Ha szeretnéd, csinálhatok még részletesebb példákat is hozzá! Szóljon, ha kell!