Juhász Tibor:

Kerekítés felfelé, lefelé vagy tetszés szerint ☺

Kiegészítés a Programozási ismeretek című könyvhöz (Műszaki Kiadó 2011)

A feladatok megoldásánál előfordulhat, hogy a kapott értékeket kerekíteni kell. A 2010. májusi emelt szintű informatika érettségi programozás feladatában például többféle módszer alkalmazását várták el. Az alábbiakban bemutatjuk a kerekítés lehetőségeit a .NET eszközeinek felhasználásával, illetve ezek nélkül, némileg általánosítva a problémát.

Hangsúlyozzuk, hogy nem a numerikus értékek kerekített (adott számú tizedesjegyre) történő megjelenítését tárgyaljuk, hanem a változó/kifejezés aktuális értékét „konvertáljuk” a kerekített értékre (általában további felhasználás céljából).

Kerekítés a Math osztály metódusaival

A kerekítést alapvetően a *Math* osztály metódusaival végezzük. Ezek a metódusok a kerekítésre kerülő kifejezést *Double* vagy *Decimal* típusúként értelmezik, és ugyanilyen típusú lesz a visszatérési érték. Ügyeljünk a függvények pontos definíciójára!

Math.Ceiling

A *Ceiling* függvény azt a legkisebb egész számot adja vissza, amely nem kisebb az argumentumnál (ceiling: mennyezet). Például:

```
Math.Ceiling(3.1) = 4
Math.Ceiling(5) = 5
Math.Ceiling(–3.9) = –3
```

Alkalmazására például a fent említett érettség feladat megoldásánál kerülhet sor. A feladat szerint a távolsági autóbuszjáratokon megkezdett km-enként 71 Ft-ba kerül a jegy. A jegy árát a következő kifejezéssel határozhatjuk meg:

```
Jegyár = Math.Ceiling(Távolság / 10) * 71
```

Megjegyezzük, hogy ha a jegyárat és a távolságot egész típusú változóként tároljuk, akkor a következő kifejezéssel is elvégezhetjük a számítást (a zárójelekre a műveletek precedenciája miatt van szükség):

```
Jegyár = ((Távolság + 9) \ 10) * 71
```

Az MS Office 2010 Excel *Kerek.Fel* függvénye a *Ceiling*-hez hasonló szerepet tölt be, de megadhatjuk, hogy hány jegyre történjen a kerekítés.

Math.Floor

A *Floor* függvény azt a legnagyobb egész számot adja vissza, amely nem nagyobb az argumentumnál (floor: padló). Például:

```
Math.Floor(3.9) = 3
Math.Floor(5) = 5
Math.Floor(–3.1) = –4
```

Ez a függvény megfelel a matematika *Egészrész* függvényének. Az egészrészt többek között felhasználhatjuk a típuskonverzióra, ha kerekítés nélkül szeretnénk végrehajtani. Például:

```
Math.Floor(CDbl(TextBox1.Text))
```

Egy szám törtrészét az egészrész segítségével kaphatjuk meg:

Törtrész = *Szám* – *Egészrész*(*Szám*)

Például:

```
3.21 – Math.Floor(3.21) = 0.21
–4.8 – Math.Floor(–4.8) = +0.2
```

Figyeljünk a negatív számok törtrészére! A fenti érték megfelel a törtrész matematikai definíciójának.

Az *Egészrész* függvényt sok programozási nyelvben (és az Excelben is) *Int*-tel jelölik. Megjegyezzük, hogy Visual Basic-ben rendelkezésünkre áll a *Microsoft.VisualBasic* névtér, amelynek *Conversion* osztálya tartalmazza az egészrészt meghatározó *Int* metódust. A fejlesztőrendszer alapértelmezés szerint importálja a VisualBasic névteret, így nincs szükség minősített hivatkozásra:

```
Int(–4.8) = –5
```
Az Excel *Kerek.Le* függvénye a *Floor*-hoz hasonló szerepet tölt be, de megadhatjuk, hogy hány jegyre történjen a kerekítés.

Math.Truncate

A *Truncate* függvény elhagyja a szám tizedesvessző utáni számjegyeit (truncate: levág, csonkol). Például:

```
Math.Truncate(3.9) = 3
Math.Truncate(5) = 5
Math.Truncate(–3.1) = –3
```

Mint látható, a *Floor* és a *Truncate* a negatív számokra ad egymástól eltérő eredményt. A *VisualBasic* névtér *Conversion* osztályának *Fix* metódusa megfelel a *Truncate* metódusnak.

Egy szám tizedesvessző utáni részét (tehát nem a matematikai értelemben vett törtrészét!) a következő kifejezéssel kaphatjuk meg:

```
Tizedesrész = Szám – Math.Truncate(Szám)
```

Például:

```
3.21 – Math.Truncate(3.21) = 0.21
–4.8 – Math.Truncate(–4.8) = –0.8
```

Az Excel *Csonk* függvénye a *Truncate*-hoz hasonló szerepet tölt be, de megadhatjuk, hogy hány tizedesjegyet hagyjon meg a csonkolás.

Math.Round

Az argumentumként megadott kifejezést a legközelebbi egészre kerekíti. A kerekítésnél kétféle módszer között választhatunk.

- Szabványos kerekítés: az 1, 2, 3, 4 számjegyet lefelé, az 5, 6, 7, 8, 9 számjegyet felfelé kerekíti. Például 3,4 ≈ 3; 3,5 ≈ 4. Ez a nálunk megszokott módszer.
- Bankárkerekítés: az 1, 2, 3, 4 számjegyeket lefelé, a 6, 7, 8, 9 számjegyeket felfelé kerekíti. Az 5-ös számjegyet azonban a legközelebbi páros szám felé kerekíti. Például 3,5 ≈ 4, de 4,5 ≈ 4.

> A bankárkerekítés valójában „igazságosabb” módszer, mint a nálunk megszokott kerekítési szabály. Jelenleg például a boltokban Ft pontossággal számolják ki a fizetendő összeget, majd az 5 többszöröseire kerekítik. Ennek következtében az összegnél kevesebbet fizetünk, ha az utolsó számjegy 1, 2, 6 vagy 7; és többet fizetünk, ha az utolsó számjegy 3, 4, 8 vagy 9. Sok vásárlást tekintve ugyanannyiszor fizetünk többet, mint kevesebbet. Idővel majd az infláció következtében már az 5 Ft-os érméket is kivonják a forgalomból (☹). Ekkor az összegnél kevesebbet fogunk fizetni, ha az utolsó számjegy 1, 2, 3 vagy 4, azaz négy esetben; és többet fogunk fizetni, ha az utolsó számjegy 5, 6, 7, 8 vagy 9, azaz öt esetben. Hosszabb távon tehát a kereskedő jár jól. A bankárkerekítés éppen ezt az igazságtalanságot szünteti meg.

A *Math.Round* függvényt négyféleképpen hívhatjuk meg:

```
Math.Round(kifejezés)
Math.Round(kifejezés, jegyekszáma)
Math.Round(kifejezés, módszer)
Math.Round(kifejezés, jegyekszáma, módszer)
```

A *kifejezés* a kerekítésre kerülő kifejezést jelenti. A kerekítés *jegyekszáma* darab tizedesre történik. Ha a jegyekszáma 0, akkor egészre történik a kerekítés.

A *módszer* argumentum a *MidpointRounding* típusú felsorolás egy eleme, amely meghatározza a kerekítés módját. Lehetséges értékei:

MidpointRounding.AwayFromZero	szabványos kerekítés
MidpointRounding.ToEven	bankárkerekítés (even: páros)

Ezeket a „csúnya” kifejezéseket nem kell megtanulnunk, mert a kódszerkesztő az argumentumokat elválasztó vessző begépelése után kilistázza a lehetséges értékeket. ☺

Jegyezzük meg, hogy **a *Round* metódus alapértelmezés szerint bankárkerekítést használ**! Tehát a nálunk szokásos kerekítéshez mindig meg kell adnunk a kerekítési módszert!

Az Excel *Kerekítés* függvényének megadhatjuk, hogy hány számjegyre történjen a kerekítés.

Kerekítés egy szám többszöröseire

A bevezetésben említett érettségi feladatban a jegy árát az 5 többszöröseire kellett kerekíteni (mivel a legkisebb pénzérme 5 Ft). Tetszőleges szám többszöröseire történő kerekítést visszavezethetünk az egész értékekre való kerekítésre. Jelöljük *k*-val azt a számot, amelynek a többszöröseire kerekítünk, *x*-szel pedig a kerekítendő értéket! A kerekítéshez az *x*-et elosztjuk *k*-val, a hányadost egészre kerekítjük, majd a kerekített értéket megszorozzuk *k*-val:

```
Math.Round(x / k, MidpointRounding.AwayFromZero) * k
```

Az Excel *Többsz.Kerekít* függvénye közvetlenül lehetővé teszi ezt a fajta kerekítést.

Számolás valós értékekkel

A valós típusú (*Single*, *Double* stb.) értékekkel történő számolás során mindig figyelembe kell vennünk a számábrázolás pontatlanságait. A számítógép kettes számrendszerben tárolja a numerikus értékeket. A törtrész konvertálásánál általában kerekítésre van szükség (a legtöbb véges tizedes tört végtelen szakaszos tört lenne a kettes számrendszerben). Ennek meglepő következménye például az alábbi utasítások eredménye:

```
Dim A As Single = 0.1
Dim B As Single = 3 * A
MsgBox(3 * A & "=" & B)
If 3 * A = B Then
    MsgBox("Helyes.")
Else
    MsgBox("Hibás!")
End If
```

A fenti ellentmondást ugyan kiküszöbölhetjük *Double* típus alkalmazásával, de így is kaphatunk meglepő eredményeket. A következő függvényhívás visszatérési értéke például 2,13 lesz annak ellenére, hogy előírtuk a szabályos kerekítést:

```
Math.Round(2.135, 2, MidpointRounding.AwayFromZero)
```

A valós számok kezelését részletesebben a tankönyv *26. Valós típusú változók* című leckéje mutatja be.

Kerekítés az egészrész függvény alkalmazásával

A lebegőpontos műveletvégzés következtében előforduló pontatlanságokat kiküszöbölhetjük, ha visszavezetjük a számolást egész értékekre.

Először tekintsük a következő ötletet. Az egészre történő kerekítésnél adjunk hozzá a számhoz 0,5-et, majd vegyük az így kapott érték egészrészét:

```
Math.Floor(Szám + 0.5)
```

Például:

```
Math.Floor(4.49 + 0.5) = Math.Floor(4.99) = 4
Math.Floor(4.5 + 0.5) = Math.Floor(5) = 5
Math.Floor(4.51 + 0.5) = Math.Floor(5.01) = 5
```

Látható, hogy a szabályosan kerekített értékeket kapjuk meg. A következő függvényhívás azonban már hibás eredményre vezet (a *D* a *Decimal* típusazonosítója):

```
Math.Floor(4.4999999999999999D + 0.5)
```

Ha ki akarjuk küszöbölni a lebegőpontos számábrázolás pontatlanságait, akkor számoljunk az összeadandók tízszeresével, majd osszuk el az eredményt 10-zel:

```
Math.Floor(10 * Szám + 5) / 10
```

Ellenőrizzük a kifejezés helyességét egy egyszerű program segítségével!

A fenti módszert könnyen általánosíthatjuk tetszőleges szám többszöröseire történő kerekítésnél. Két tizedesre történő kerekítés esetén például a 0,01 többszöröseire kerekítünk, a pénztárnál pedig az 5 többszöröseire kerekítenek.

Jelöljük ismét *k*-val azt a számot, amelynek a többszöröseire kerekítünk, *x*-szel pedig a kerekítendő értéket! A kerekítést megint visszavezetjük az egész számokra történő kerekítésre úgy, hogy az *x*-et elosztjuk *k*-val. Ezt a hányadost már egészre kerekítjük, majd a kerekített értéket megszorozzuk *k*-val:

```
Math.Floor(x / k + 0.5) * k
```
illetve az összeadandók tízszeresével számolva:
```
Math.Floor((10 * x / k + 5) / 10) * k
```
Ha a *k* törtszám, akkor az *x*/*k*-t szorozzuk is és osszuk is 10 azon *m* kitevőjű hatványával, amelyre a $10^m \cdot k$ már egész szám (például $k = 0{,}06$ esetén $m = 2$)
```
Math.Floor((10 * 10^m * x / (10^m * k) + 5) / 10) * k
```
Programunkkal határozzuk meg mindhárom kifejezés értékét $x = 2{,}135$ és $k = 0{,}01$ esetén! Láthatjuk, hogy míg az első két kifejezés a hibás 2,13 értékre vezet, az utolsó kifejezés már a helyes 2,14 értéket szolgáltatja.

Esetenként előfordulhat, hogy az utolsó kifejezést egyszerűbb alakra hozhatjuk. $k = 0{,}01$-nál például:
```
Math.Floor((1000 * x + 5) / 10) / 100
```
A bevezetésben említett érettségi feladatban az 5 többszöröseire kellett kerekíteni a fizetendő összeget. Mivel az 5 egész szám, nincs szükség a 10^m-nel történő szorzásra:
```
Math.Floor((10 * x / 5 + 5) / 10) * 5
```