Programozzunk Pythonban (1. rész)

A szolid óriáskígyó

Elegáns, hatékony és könnyen tanulható programozási nyelv a Python. Egyaránt megállja a helyét kis feladatokban és nagy csapatmunkában, több platform alatt, akár ragasztónyelvként használva. Egy kezdőnek is lehet vele gyorsan sikerélménye, a profik kezében erőteljes szoftverfejlesztő eszköz.

Miért kezdjünk (ki) egy óriáskígyóval?

A programozási nyelvek száma szerencsére megdöbbentően nagy, ám ezekből egy átlagos felhasználó vagy fejlesztő általában csak néhány nevével találkozott már. Népszerűbb nyelvek többnyire azért válnak ismertté, mert adott feladatokra vagy adott környezetben jól használhatóak, vagy épp ellenkezőleg: általános célokra is, és többféle platformon is megfelelően működnek. A *Python* utóbbi szempontok alapján válhatna igazán elterjedtté, hiszen változatos futtatókörnyezetekben is megállja a helyét, legyen szó akár egy *Debian Linux*-disztribúciót használó munkaállomásról, *Symbian* operációs rendszert futtató mobiltelefonról vagy Solaris alapú webszerverről.

Mégsincs könnyű dolga, ha a fejlesztőket szeretné meghódítani, hiszen az informatikai világban nem is oly rég végbement változások következtében ma már roppant népszerűségnek örvend általános célú nyelvként a *Java*, a webes szkriptnyelvek között a *PHP* az egyik kedvenc, bővülni és megújulni látszik a *Perl*, a *JavaScript*, erősödik és függetlenedik a *C#*, töretlen adatbázismanipulációkban az *SQL* dominanciája, újra reflektorfénybe kerültek a héjprogramok, és még néhány, területspecifikus nyelv is tolong az elsőbbségért. Mit tud hát felmutatni a már nem is oly fiatal *Python*, amiért érdemes egyáltalán megpróbálnunk megszeretni, s reménykednünk abban, hogy a befektetett tanulás örömteli élményekkel, s talán komolyabb tudással is gazdagít?

Szerzője *Guido van Rossum*, aki a *Google* csapatát gazdagítja egy ideje, s jelenleg is foglalkozik alkotásával és használja is azt. Nevét nem az óriáskígyóról, hanem a *Monthy Python repülő cirkusza* című angol humorban gazdag alkotásról adta, ennek ellenére sokan előszeretettel használnak vidám zöld óriáskígyót különböző piktogramokon.

A nyelvhez elérhető – általában hobbiból vagy egy-egy projekt kapcsán született, s később kanonizálódott – kiegészítőkön, fejlesztői könyvtárakon és modulokon kívül is találhatunk önálló keretrendszerré fejlődött, *Python* alapú vagy azt hasznosítható eszközöket, legismertebbek talán a komplett tartalomkezelő *Zope* és *Plone*, vagy a webes *Qweb*, *Cherrypy* és a *TurboGears* (utóbbi ráadásul felhasználja előbbit), a *BitTorrent* fájlcserélő, és a *Trac* verziókövető rendszer. Némelyikükkel már a *Linuxvilág* hasábjain is találkozhattunk, például *Juhász Attila* több keretrendszert összehasonlító remek kedvcsináló áprilisi cikkében.

1. ábra Egy jó Python kis helyen is elfér – akár mobilon is

➲ *http://www.linuxvilag.hu/system/files/cikk_63_16_18.pdf*

Fantasztikus történeteket olvashatunk e nyelv tanulhatóságáról, tömörségéről és hatékonyságáról az interneten. Mindez leginkább két dolognak köszönhető: szkriptnyelv volta ellenére objektum-orientált, és nagyon magas szintű nyelvnek nevezhető. Előbbivel többször fogunk majd találkozni, és részletezzük ezen tulajdonság előnyeit, utóbbi pedig leginkább azt jelenti, hogy rengeteg előre megírt objektum és felhasználható elem könnyíti életünket. Nézzünk egy könnyű példát,

vessük össze a *C* nyelvvel két változó értékének felcserélését, előbb *C* nyelven:

```
void main()
{
char x="kutya";
char y="macska";
char z; /*ideiglenes változó,
➥csak a csere idejére kell */
z=x;
x=y;
y=z;
}
```

Ugyanez *Python*ban:

```
x="kutya"
y="macska"
x,y = y,x
```

Még nyilvánvalóbb lenne a példa, ha a nyelv alapvető eszköztárának elemeit vetnénk össze, leginkább az ún. absztrakt adatszerkezeteket, melyek összetettségükkel hasznos és gyors segítőinkké válhatnak – ilyenek például a listák, szótárak, szekvenciák, asszociatív tömbök – s melyek megalkotása sok más nyelvben vagy a programozó feladata, vagy a nyelvnek nem szerves részét képező kiegészítő modulokban találhatók.
Gyakorta hasonlítják össze a sokszínű és szövegmanipulációban kiváló *Perl* nyelvvel, ilyen rövid cikk például – a magyar *Python* oldalról is elérhető egy hivatkozás segítségével – leginkább A *katedrális és a bazár* című művével hazánkban is népszerű *Eric S. Raymond* írása, melyben a *Python* forráskódok könnyű karbantarthatóságát, nagyobb projektekhez is jól használható tulajdonságait emeli ki. Ne törjünk pálcát egyetlen programozási nyelv felett sem, mert elsődlegesen a programozó feladata megoldani az adott problémát, s a nyelv ebben segítheti vagy gátolhatja őt, de gondolkodni helyette bizonyosan nem képes.

Milyen is egy Python?

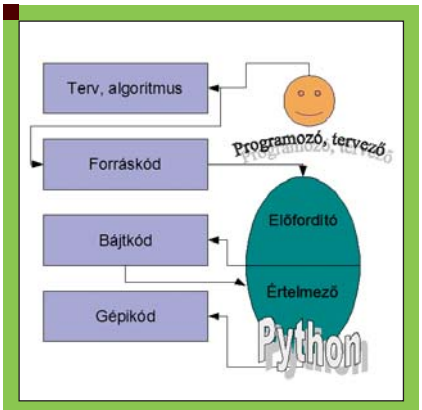
Kiválóan használható programozási nyelv, jó alapot biztosíthat a továbblépésre komplex és elterjedt objektumorientált programozási nyelvek irányába, mint amilyen a *C++* és a *Java* (feltéve hogy nem szeretünk bele túlságosan is kényelmes és gyors fejlesztést lehetővé tevő nyelvezetébe, és emiatt nem is akarunk továbblépni). Néhány fontosabb tulajdonsága:

- objektum-orientált szemléletű programozást támogat
- nem kell a memóriakezelés terhétől roskadozni, mert átvállalja helyettünk
- szkriptszerű, azonnal láthatjuk a változtatások eredményét
- rengeteg előre megírt kiegészítő és csatolófelület található hozzá
- jól áttekinthető, egyszerű, tömör
- szigorú: kötelező behúzásokat alkalmazni a tagoláshoz
- támogatja és elvárja a szabványos kivételkezelést
- kitűnő ragasztónyelv, jól illeszkedik más eszközökhöz (*C* nyelv, adatbázisok stb.)

Alapvetően szkriptnyelvnek tekinthetjük, bár kétségkívül kissé kilóg a sorból, csakúgy, mint például a *Java* a fordított nyelvekéből. Itt is úgynevezett bájtkóddal dolgozik az értelmező, azaz az általunk megírt forráskódból előbb egy átmenet – az értelmező számára emészthető és könnyen gépi nyelvre alakítható közteskód – s nem egyből gépikód keletkezik.
Két hátrányát mindjárt meg is említhetjük: elméletileg *Python* verziónként eltérhetne az általa készített közteskód értelmezése, s a dinamikus típusátadás miatt elég nehéz jó gyorsító-optimalizáló megoldást találni hozzá. Előbbit a forráskód mellékelésével, utóbbit a standard nyelv mások által átalakított, statikus típusátadást utánzó verziójával vagy modulokkal, például a *Pyrexszel* lehetne orvosolni – bár igazán csak akkor lehet ennek létjogosultsága, ha erősen számolásigényes feladatot próbálnánk meg kizárólag *Python* segítségével megoldani.
Létezik még egy viszonylag elterjedt, *Java* nyelven újraírt értelmező is, a *Jpython*, mely *Java* által értelmezhető közteskódot generál.

Kezdődjék a (repülő) cirkusz!

Mire lesz szükségünk? Természetesen magára a Python nyelvre, mely a legtöbb disztribúció telepítőlemezén megtalálható, jó eséllyel az olvasó is rátalálhat saját rendszerében, s kiirathatja verziószámát parancssorból:



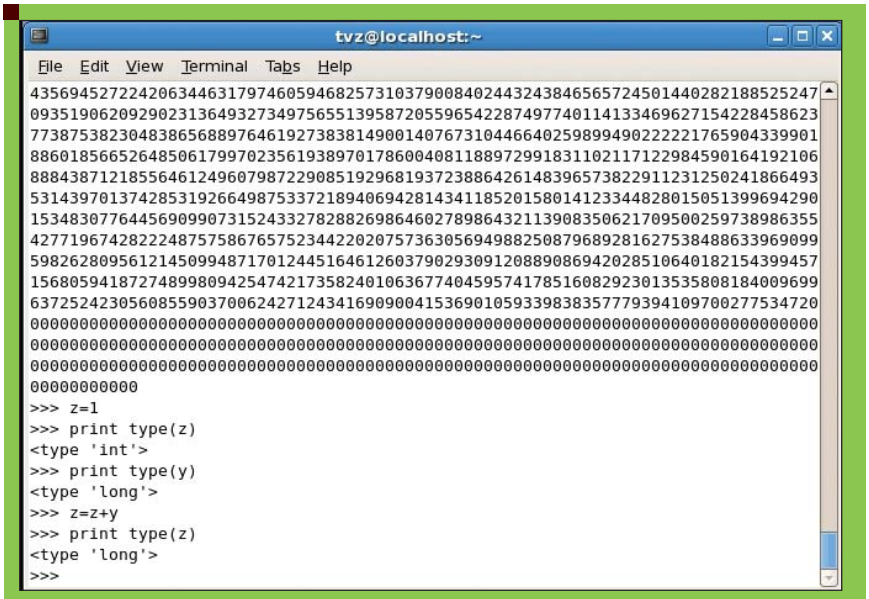
2. ábra A köztes bájtkód szerepe

```
python -V
```

Ha mégsem így lenne, a használt Linux változat hivatalos honlapján érdemes körülnézni először az elérhető csomagok között. Egy Debian rendszerben például aktív internetkapcsolat esetén általában elég kiadni megfelelő jogosultságokkal:

```
apt-get install python
```

Szükség van továbbá egy szövegszerkesztőre is, melynek segítségével a forráskódot írjuk, gyakorlatilag értelmezhető fájlokat készítünk.
S már csak a legalapvetőbb programozási ismeretek kellenek a sikeres kezdéshez – ezzel feltételezhetően mindenki rendelkezik, aki tanulmányai során egy csepp informatikával találkozott, vagy akár önszorgalomból *Linux*ot használva próbálkozott már egyszerűbb szkripteket írni. Oktatóanyagból és speciális feladatokra optimalizált leírásokból is jócskán találhatunk elektronikus és nyomtatott formában dokumentumokat, s talán a legkézenfekvőbb mű a nyelv alkotójának elektronikus formátumokban szabadon hozzáférhető írása.
Különböző trükkökkel elérhető akár programunk „közvetlen futtathatósága” is, például a *Google* nyílt forrású fejlesztéseiért felelős *Greg Stein* blogjában ➲ *http://www.lyra.org/greg/python/#dev* még a jóval korábbi verziókhoz írt egy kétsoros *Bash-szkript*et, mely által a webszerver */cgi-bin* könyvtárába helyezve a megírt *py* kiterjesztésű programunkat azonnal rávehetjük a rendszert annak értelmezésére. Hasonló hatást kelt,



3. ábra Típuskonverzió automatikusan – az óriáskígyó esete az óriásszámmal

ha héjprogramokban (***shellszkript***) is használható módon „futtathatóvá" tesszük programunkat, azaz először megadjuk egy direktívában a ***Python*** elérési útvonalát – egyszerű megtudnunk, például egy terminálablakban a `which python` paranccsal kiírathatjuk – majd megírjuk a programot, elmentjük, végül futtathatóvá tesszük. Példa ***vi*** szövegszerkesztő segítségével:

```
vi programunk.py
```

Ezzel megnyitottunk egy egyelőre még üres, de már névvel rendelkező fájlt, ez fogja tartalmazni a forráskódot, majd ***ESC*** billentyűt leütve tudatjuk a programmal, hogy beszúrni szeretnénk szöveget az üres sorba, erre szolgál a ***:i*** és utána az ***Enter*** billentyű. Jöjjenek a programsorok, természetesen ügyelve a kisbetű/nagybetű használatára:

```
#! /usr/bin/python
kiirando="Fut a program!"
print kiirando
```

ESC billentyű, majd `:w` és ***Enter*** (azaz mentjük), majd `:q` és ***Enter*** (azaz kilépünk).
A kész fájlt futtathatóvá a `chmod +x` paranccsal tehetjük, s nincs is más dolgunk, mint elindítani:

```
./programunk.py
```

Még egy sort érdemes beszúrni a forráskód elejére:

```
# -*- coding:Utf-8 -*-
```

Ezzel tudatjuk az értelmezővel, hogy mi a magyar ékezeteket is helyesen kezelő ***UTF-8*** karakterkódolást szeretnénk programunkban használni – bár tapasztalataim szerint enélkül is általában ékezethelyesen működik egy jól beállított, magyar nyelvű linuxon, mobiltelefonnal már nem voltam ilyen szerencsés.
A ***vi*** szövegszerkesztő használata ízlés kérdése, léteznek sokkal egyszerűbb és látványosabb programok is e célra (például ***Van Rossum*** által írt ***IDLE*** nevezetű fejlesztőkörnyezet), vitathatalan előnye viszont, hogy a legtöbb ***Unix/Linux*** disztribúcióban megtalálható valamilyen formában.
Használhatjuk a ***Python*** értelmezőjét úgynevezett interaktív módban is, ekkor egy hagyományos ***Linux*** héjhoz hasonlóan fog működni, azaz beírjuk a parancsokat, s ő végrehajtja vagy hibaüzenttel jelez vissza. Indítása roppant egyszerű:

```
python
```

Elszámolunk

Majdnem minden népszerű leírás megemlíti, hogy a ***Python*** remekül használható interaktív módban számológépként, ha erre kíváncsiak vagyunk, írjunk be egy egyszerű műveletet a készenléti jel (>>>) után:

```
20/6
```

Az eredmény 3 lesz, amit úgy kell érteni, mintha az eredménynek csak az egész számú részét írta volna ki az értelmező. Ha még nem programoztunk más nyelven korábban, valószínűleg újdonságot jelent, hogy van külön olyan művelet, mely képes az osztás maradékának kijelzésére is, ekkor a % jelet kell használnunk:

```
12%5
```

és megkapjuk a 2-t. E két művelettel például könnyedén meg tudjuk mondani egy számról, hogy páros-e:

```
a=15
if a%2==0 :
        print "Páros"
else :
        print "Páratlan"
```

Egyúttal több dolgot is megtanulhatunk e rövid példából. A legfontosabb: a ***Python*** nagyon ügyel a jó tagolásra, konkrétan a sorvégejeleket és a behúzásokat figyeli. Legyünk következetesek: e célból ne keverjük a szóköz és a tabulátor használatát (hacsak nem olyan szerkesztőprogrammal dolgozunk, ahol beállítható, hogy a tabulátorjeleket automatikusan szóközökre cserélje), mert a fordító különböző szintű tagolásnak fogja őket tekinteni.
Az értékadás (`a=15`) utáni sorban egy feltételes vizsgálat következett (`if a%2==0`), megnéztük, hogy az a változó értékét 2-vel osztva 0-t kapunk-e maradékul. A dupla egyenlőségjel (==) már nem értékadást, hanem összevetést jelent, azaz a bal és jobb oldalán szereplő értékeket próbáljuk meg összehasonlítani segítségével. Egy kettőspont zárta ezt a sort, mivel még nincs vége a kiértékelésnek, ez csak a művelet feje volt, és a több sorban leírt tartalmat így tesszük az értelmező számára egyértelművé. Ha teljesült a műveletfejben megfogalmazott feltétel, akkor a beljebb húzással jelölt utasítás (`print "Páros"`) hajtódik végre, míg ha nem igaz, akkor az `else` ága kerül

kiértékelésre. Itt már nem volt szükség újabb feltételt vizsgálni, de ez is egy műveletfejnek minősül, így itt is találunk kettőspontot. Az utolsó sor tartalmazza azt a kiírandó szöveget, mellyel akkor találkozunk, ha nem volt igaz az első összehasonlítás.
Itt beljebb húzott tartalom tudatja velünk, hogy logikailag alá van rendelve a feltételvizsgálatnak (ezúttal az `else` ágnak).
Nyomát sem láthattuk a példában a más nyelveken gyakori `;` , `{}` vagy egyéb blokkhatároló jeleknek, s mégis jól látható, meddig tart egy logikai egység. Bármennyire is kellemetlennek tűnik eltérő logika alapján szerveződő programozási nyelv használata után megbarátkozni a kötelező behúzáshasználattal, egy idő után rájövünk, valóban a mi érdekünket is szolgálja, hiszen átlátható kódot eredményez. Ez pedig elsőrendű szemponttá válhat egy nagy projekt esetén, ha csapatmunkáról van szó, vagy ha egyszerűen csak saját programunkat szeretnénk karbantartani hosszú idő elteltével. Ehhez kapcsolódva érdemes még két programozói szokást magunkévá tennünk: a # jellel bevezetett megjegyzések használatát (ezt az értelmező figyelmen kívül hagyja, nekünk viszont roppant hasznos lehet később), és a hibák – valamint kivételek kezelését.

Dinamikus átalakulások

Bonyolódjunk bele még egy kicsit a számokba, nézzünk egy egyszerűnek tűnő példát: a faktoriálist. Az általános képlete szerint n! = n * (n-1) * (n-2) * … * 2 * 1. Ennek kiszámítására több megoldás is lehetséges, a rekurzív függvényhívás egyik klasszikus példájaként is hivatkozhatnánk rá, most nézzünk inkább egy egyszerűbb megoldást:

```
x=4 #ciklusváltozó és a képlet
➥része is egyben
y=x #tárolja a részleges
➥eredményeket és a végleges n!
➥-t
while x>1 :
        x=x-1
        y=y*x
print y #kiíratjuk n!-t
```

Az x-et úgynevezett ciklusváltozóként használtuk, értékét folyamatosan csökkentjük, s a while utasítás segítségével mindig megvizsgáljuk, vajon teljesül-e még az a feltétel, hogy nagyobb 1-nél. Ha igaz, akkor minden ami a kettőspont után van végrehajtásra kerül, ha hamis, akkor kilép a ciklusból, s esetünkben be is fejezi futását a program.
A példában egy kis számot, a négyet néztük meg, de érdemes tudni, hogy a faktoriálisok roppant gyorsan növekszenek, 13! már egy tízjegyű számot jelent. Próbáljuk ki a programot egy nagyobb számra, például 1000-re (`x=1000`). Azonnal észrevehető, hogy a gép sokáig „gondolkodik”, hiszen hatalmas számokkal kell dolgoznia. Ami kevésbé látható: a dinamikus változókezelés következtében észrevétlenül átváltott az addig használt ***integer*** típusú egész számokról ***long*** típusú hosszú egészekre, mivel előbbi csak körülbelül kétmilliárdig képes a számokat fogadni, míg utóbbi kis közelítéssel a fizikai memória határáig. Ha nem hisszük, próbáljuk ki, előbb egy új változóval:

```
z=1
print type(z)
```

Válaszul azt kapjuk, hogy a z változó típusa ***int*** , azaz ***integer***. Kérdezzük le az 1000! értékét tartalmazó y változó típusát is:

```
print type(y)
```

Valóban ***long*** típusú a változó. Mi történik akkor, ha e két típus keveredik, azaz például egy összeadás során mindkettő szerepel?

```
z=z+y
print type(z)
```

Az eredménynek olyan típusúnak kell lennie, amelyben elfér a nagyon nagy szám és a kisebb összege, azaz logikusan ***long*** típusú lesz. Ehhez hasonló ún. implicit típuskonverziót a *C* nyelv is használ.
Az eddigi példákban használt egész számokat vegyíthetjük lebegőpontos számokkal (***float***) is, s ott is hasonló jelenséget tapasztalunk, azaz például egy ***int*** típusú és egy ***float*** típusú házasságából ismét csak ***float*** kategóriába tartozó szám születik.

```
a=1
        print type(a)
b=2.5
        print type(b)
c=a+b
        print type(c)
```

Az ***int***, ***long***, ***float***, ***complex*** (eddig még nem említettük, a komplex számokat jelenti) kulcsszavak segítségével rákényszeríthetjük egy változóra az adott típus használatát (*C* és hasonló nyelveken ez az ún. explicit típuskonverzió). Ennél talán még érdekesebb a következő típusváltás:

```
az_elet_ertelme="42"
szam = 42
eredmeny=int(az_elet_ertelme)
➥+szam
print "Az élet értelme
➥bizonyosan nem " , eredmeny,
➥"!"
```

Az első változó egy egyszerű szöveg típusú változó volt, noha szemmel láthatólag csak egy számnak látszó értéket tárolt. Az `eredmeny` változóban ezt ki is használtuk, s ***int*** típusú számmá konvertálva már össze tudtuk adni egy igazi számmal.
A dokumentációt olvasva sok érdekes átalakítással és hasonló „trükkel” találkozhatunk, melyeket jól használva saját munkánkat könnyíthetjük meg. Kisebb példaprogramok írásával már ennyi ismeret birtokában is tudunk saját szerzeményünkkel hasznot hajtani. A különböző információforrások (levelezőlisták, hivatalos honlap, wiki stb.) jó alapot nyújthatnak, és a továbblépést is megkönnyítik ha elakadnánk valahol. A számok után a szöveg típusú változókkal, műveleti sorrenddel érdemes foglalkozni, s mindeközben észrevétlenül barátkozhatunk az objektum-orientált szemlélettel is.

Tóth Virgil Zoltán
(m_v@c2.hu)
Szoftverfejlesztő informatikus és rendszergazda, kedvence a Debian disztribúció. Szabadidejét legszívesebben felesége és szépirodalmi regények társaságában tölti. Lenyűgözőnek tartja a Linux rugalmasságát, és a vele dolgozók aktivitását.