Pitonológia

Ki kell ábrándítanom a bilológusokat, mert nem a pitonokról lesz szó. Ahelyett, hogy a hüllők világát tanulmányoznánk én inkább egy programozási nyelvből, a Pythonból szeretnék átadni néhány villanást, ezzel remélve hogy minél többen fogjátok használni és ezzel végső soron rengeteg időt fogtok megtakarítani amelyet mindenféle értelmesebb dolgokra fogtok használni, mint például elmentek bulizni, megsétáltatjátok a kutyát, elutaztok nyaralni, vagy esetleg vesztek néhány tucat Python könyvet, és egész hátralevő életetekben már csak Pythonban fogtok kódolni és ki sem mozdultok a barlangotokból.

Gondolkodtam rajta hogy hosszasan regélek a különböző típusú nyelvek közötti alapvető szemléletbeli különbségekről, a Python helyéről köztük és hasonló mély és tanulságos dolgokról, de végülis úgy döntöttem, hogy minél emészthetőbb és játékosabb, de nem lebútított stílusban fogom írni ezt a cikket. Néhány sor bekezdés után rögtön elkezdünk kódolni és van egy olyan érzésem hogy akit kicsit is érdekel a téma az rettentően fogja élvezni.

Hogy miért? Mert a Python az a nyelv, amely valóban képessé tesz arra, hogy a problémára koncentrálj. Úgy tervezték meg, hogy ne a nyelv sajátosságain kelljen rágódnod, hanem amennyire csak lehet magától értetődő legyen a használata. Gyorsabban fogod megtanulni, mint bármely más eddigi nyelvet amit ismersz és pillanatok alatt képes leszel benne kódot írni. Értékelni fogod a nyelv páratlan egyszerűségét és tisztaságát.

De várjunk csak, már hallom hogy kérdezed: “Ha ez egy ilyen csúcs nyelv, akkor miért nem hallottam róla vagy miért nem tanítják?”. A legtöbb nyelv, amiket tanítanak mind komoly múlttal rendelkező ipari szabványok. Ilyen a C, C++, Java, vagy majd pár év múlva a C#. Mindegyik ilyen nyelv mögött neves szabványosító szervezetek és cégek állnak komoly anyagi érdekekkel. A Python nem egy ilyen nyelv. Éppen ezért az egyetemeken nem (nagyon) tanítják és csak akkor szerzel tudomást róla ha te magad teszel kitérőt az alternatív nyelvek világába. Több hasonló nyelv is van a Python súlycsoportjában, (pl. a Perl, a Tcl vagy a Ruby) de az a tapasztalatom (és sok más ember tapasztalata), hogy a Python a legjobban méretezhető és legtisztább ezek közül. Tehát ha írsz egy programot és az természetszerűleg egyre hosszabra nő, akkor nem fog “szétesni”, hanem átlátható marad, mert a nyelvet jól megtervezték. Persze ez azért rajtad is múlik, de a nyelven a legkevésbé.

Hogy mire tudod használni? Tulajdonképpen szinte akármire. Írhatsz benne szövegszerkesztőt, játékot, webszervert, kalkulátort vagy szinte amit csak akarsz. Rengetek különböző kiterjesztés tölthető le hozzá amelyekkel a legkülönbözőbb feladatokat oldhatod meg könnyedén. Talán egyszerűbb lenne arról írni hogy mire nem használhatod. Nem írhatsz benne például villámgyors 3D-s engine-t. Általános szabályként elmondható, hogy a Python sebességben kicsit gyenge a gyengén típusos mivolta miatt (amiről később majd ejtek néhány szót). Összehasonlításként dúrván elmondható, hogy ami C-ben 1 másodperc alatt lefut, az mondjuk Javában 5 másodperc és Pythonban 20. Bár ez jelentős sebességveszteségnek tűnhet, a gyakorlatban általában nem az. Ez utóbbit alátámasztja jónéhány nagyon sikeres, robosztus Python alkalmazás is, például a Plone <http://plone.org>, vagy a Mailman <http://gnu.org/projects/mailman>.

A nyelv prototipizáláshoz is kiváló. Ez azt jelenti, hogy kezdetben van valalmi elképzelésed egy feladat megoldására vonatkozóan. A fejedben vagy papiron már elkészítettél egy algoritmust és most eljött az idő hogy kipróbáld. Ahelyett hogy bármelyik másik nyelven valósítanád meg és tesztelnéd az algoritmusod, inkább írd meg Pythonban. Pillanatok alatt kész leszel vele és ha hibásnak bizonyul, akkor minimális erőfeszítéssel korrigálhatod. Majd ha kész vagy és úgy működik ahogy akartad, akkor átírhatod egy más nyelvre az immáron helyes algoritumst, hogy gyorsabban fusson.

Pár szót illene ejtenem a nyelv történelméről is, nem igaz? Rövid leszek, egyrészt mert engem mindig is untatott a történelem másrészt pedig a cikk gyakorlatias jellege miatt. A Pythont Guide Van Rossum hozta létre. Guido egy osztott operációs rendszeren, az Amoeba projecten kezdett el dolgozni 1986-ban az Amszterdami Egyetem berkein belül. Az Amoebának szüksége volt egy szkriptnyelvre. Guidora hárult a feladat és elég nagy szabadságott kapott a munkájában. Korábban ugyanitt már dolgozott az ABC-n, egy nyelven, amelyet úgy terveztek hogy könnyen tanulható és használható legyen a nem-programozók számára is. Az ABC-vel kapcsolatos tapasztalatait felhasználva kifejlesztett egy új nyelvet, amely az ABC minden előnyös tulajdonságát magában foglalta és a hibáit elhagyta, így született meg a Python. A projecthez rengetegen csatlakoztak. Az Internet elterjedésével felkarolta a Szabad Szoftver Közösség és egyre növő népszerűségnek örvend azóta is.

Pár szó a használatáról: Ha Linux közelében vagy, könnyű a dolgod, mivel minden valamirevaló disztribúcióban jelen van az értelmező. Ha pedig Windowst használsz, akkor letöltheted a http://python.org címről. Kis példáknál egyszerűen indítsd el az értelmezőt (csak símán írd be hogy “python”) és máris egyenként pötyögheted bele a sorokat és a végeredményt rögtön látodhatod. Ha kezdetben kísérletezgetsz vele, mindenképpen próbáld ki.

Ha a programjaidat forrásfájlokban akarod tárolni (ami nagyobb programoknál persze nélkülözhetetlen), akkor nyisd meg a kedvenc szövegszerkesztődet, hozz benne létre egy fájlt .py kiterjesztéssel, mondjuk `filenev.py` néven és ha megvagy vele, akkor etesd meg az értelmezővel a következőképpen:

```
python filenev.py
```

Ennyi bevezető után már igazán megírhatjuk az első programunkat. Legyen valami rendhagyó. Mondjuk írjuk ki, hogy “Mizu?”:

```
print 'Mizu?'
```

Talán néhányotoknak feltűnt hogy nincs pontosvessző a sor végén, mint ahogy olyan sok nyelvben megszoktad. Ez nem a véletlen műve. Valld be, amúgy is csak zavart, nem igaz? Miután kigyönyörködtük magunkat a kimenet esztétikus mivoltában és túl vagyunk a nehéz feladat okozta mentális sokkon, találjunk ki valami komolyabbat. Például irjunk egy mesemondó programot ami megirja helyettünk a bevezetést:

```
dolgok = ['óceán', 'cápa', 'Bill Gates']
for dolog in dolgok:
    print 'Egyszer volt egy ' + dolog + '.'
```

A nem túl meglepő kimenet pedig a következőképpen fest:

```
Egyszer volt egy óceán.
Egyszer volt egy cápa.
Egyszer volt egy Bill Gates.
```

Ha a példából messzemenő következtetéseket vonnál le a Bill Gates-szel kapcsolatos véleményemről, akkor valószínűleg nem jársz túl messze az igazságtól. Kicsit boncolgassuk a programot hogy mindenki pontosan megértse minden apró darabját. Az elején a `dolgok` egy változó, amelynek rögtön értéket is adunk. Ez az érték az egyenlőségjel jobb oldalán egy tömb, amely ahogy látod három sztringet tartalmaz. A tömb elemei szögletes zárójelek között szerepelnek és vesszővel vannak elválasztva egymástól. Minden sztringet aposztrófok határolnak. Ez után egy `for` ciklussal végigiteráljuk a `dolgok` elemeit amelyek a cikluson belül egyenként `dolog` néven jelennek meg. Azt hiszem mostanra már rájöttél, hogy a `print` paranccsal írhatsz ki sztringeket. A + operátor az egyedi sztringek összefűzésére való. Így már biztos tiszta a kép.

Most nézzünk egy nagyon népszerű szösszenetet amit kis korunkban olyan nagy hévvel csacsogtunk:

```
ki = ['Az egyik', 'A másik', 'harmadik', 'A negyedik', 'Az icurka-picurka']
mitcsinal = ['elment vadászni', 'meglőtte', 'hazavitte', 'megsütötte', 'mind megette']
for i in range(5):
    print ki[i] + ' ' + mitcsinal[i] + '.'
```

Itt az egyetlen újdonság amit felfedezhetsz, a `range` függvény. Ez egyike a Python beépített fügvényeinek. Ha egy paramétert adsz át neki, akkor a megadott paraméternek megfelelő nagyságú tömböt hoz létre. Így pl. a `range(5)` függvényhívás a `[0, 1, 2, 3, 4]` tömböt adja vissza, amelynek az elemeit a `for` ciklus végigjárja. A `[]` operátorral pedig a tömböket indexelhetjük, mint ahogy azt már megszokhattuk máshonnan.

Legutolsó példaként ugorjunk egy nagyot és hozzunk létre valami használhatót: írjuk meg a világ legegyszerűbb szótárprogramját. Nade hogy is működjön? Mondjuk rögtön az elindulása után várjon parancsokra. Két féle parancsot fogadjon el: lekérdezést és feltöltést. A lekérdezés értelemszerűen egy szó visszakeresése a szótárból, a feltöltés pedig egy szó megadása. Kicsit írjuk le pontosabban a működését. Minden feltöltést vezessen be egyenlőségjel, rögtön utána a megadandó szóval, majd a következő sorban a jelentésének a beírásával. Egy szó lekérdezése magának a szónak a beírásával legyen elvégezhető és a szótár teljes szókészletének a kiírását egy kérdőjellel lehessel lekérdezni. Lássuk hogy is néz ki egy ilyen program Pythonban:

```
szavak = {}
while True:
    parancs = raw_input('> ')
    if parancs == '?':
        for szo in szavak:
            print szo + ' -> ' + szavak[szo]
    elif parancs[0] == '=':
        szo = parancs[1:]
        jelentes = raw_input(szo + ' -> ')
        szavak[szo] = jelentes
    else:
        if szavak.has_key(parancs):
            print szavak[parancs]
        else:
            print 'A szo nem talalhato a szotarban.'
```

Micsoda? Hogy lehagytam a kapcsos zárójeleket? Az az igazság hogy itt nincs olyan. Ha egy új blokkot kezdesz, akkor azt beljebb kell igazítanod, nem pedig kapcsos zárójelekkel határolnod. Bár ez a szintaxis kezdetben talán zavarni fog, csak hozzászoksz majd. A kapcsos zárójelek hiányának kellemes mellékhatása az olvashatóbb kód amelyet idővel biztos te is értékelni fogsz. Hely hiányában megpróbálom a lehető legtömörebben és legérthetőbben elmagyarázni a fenti kódot. Egy furcsa értékadással kezdődik. Ez egy üres asszociatív tömböt hoz létre `szavak` néven. Gyakorlatilag ez ebben a példában sztringeket képez le sztringekre, tehát például az egyes szavak angol megfelelőjét magyarra. A későbbi sorokból könnyen ki tudod olvasni a használatát. Ez után egy végtelen ciklus következik, ami ugyebár itt nem túl elegáns. Azért ne ijedj meg, ilyenkor a Control-D -vel mindig ki tudsz lépni. Ennek a ciklusnak minden egyes ismétlődésében egy-egy parancsot fogadunk. A `raw_input` függvénnyel bekérünk a felhasználótól egy sztringet. A `raw_input` argumentuma a bevitelkor megjelenítendő szöveg. Az if-elif-else szerkezet nem meglepő módon a Python feltételes vezérlési szerkezete. Az if ágban ha kérdőjelet kapunk, akkor az összes szót kiirjuk. Az elif ágban az új szó feltöltését kezeljük le. Itt kihámozzuk a szót a parancsból az első karakter, az egyenlőségjel lehagyásával. A parancs[1:] kifejezésben a szeletelő operátor használaba kaphatsz betekintést, ami eldobja a sztring első karakterét. Az else ágban talán a `has_key` szó lehet homályos néhányotoknak. A `szavak.has_key` egy metódushívás. Ugyebár a `szavak` egy asszociatív tömb. Minden asszociatív tömb rendelkezik egy `has_key` metódussal, amely megmondja, hogy a argumentuma (a keresett szó) eleme -e az asszociatív tömbnek. Ettől függően írjuk ki a keresett szót vagy a hibaüzenetet.

Csak hogy lássuk élőben is a szótárunkat, itt egy próba a felhasználó által beírt részt kövér betűtípussal kiemelve:

```
> =apple
apple -> alma
> =honey
honey -> méz
> =sin
sin -> bűn
> ?
honey -> méz
apple -> alma
sin -> bűn
> honey
méz
```

Ennyi erre a hónapra. A Python legyen veled, meg a jó szerencse és ami még jól esik. Minden cikkel kapcsolatos kritikát vagy bármi mást boldogan fogadok a laci@monda.hu címemre vagy ha gondolod a honlapomat is megnézheted a http://monda.hu/~laci címen.

Monda László