Python[1] mogyoróhéjban

Összeállította: Siki Zoltán

Tartalom

Bevezetés

A Python egy széles körben elterjedt általános célú szkript nyelv. Lehetőséget biztosít objektum orientált programozásra is. Számos bővítő modullal rendelkezik, például GDAL, OGR, numpy, OpenCV, PCL, stb. Mivel a Python futtatókörnyezet számos operációs rendszeren (Linux, OS X, Android, Windows) rendelkezésre áll, egyszerűen készíthetünk platform független alkalmazásokat. Számos szoftverben futtathatunk Python szkripteket, pl. GIMP, QGIS, Apache, PostgreSQL, GRASS, stb. A Python értelmező a forráskódot byte kóddá fordítja ezért a Python programok gyorsabban futnak mint más szkript nyelven írt programok.

Manapság még széles körben használják a Python 2.x változatát (pl. QGIS), de a 3.x változat is elérhető már 2008 december óta. A nagy verziószám változása miatt a kompatibilitás nincs meg a két verzió között.

A következőkben feltételezzük, hogy az OSGeo Live 8.0 DVD-t, pendrive-ot használja vagy az OSGeo4W telepítővel telepítette a QGIS-t. Az OSGeo4W telepítő esetén a Python 2.7.x verzióját, a Live DVD esetén a Python 2.7.x és a 3.4.x verzióját is használhatnánk, de a példákat csak 2.7 verzióval teszteltem.

A Pythont használhatjuk interaktívan is, az értelmezővel azonnal végrehajtatjuk az utasításainkat. Az operációs rendszer parancs ablakában (Windows-on OSGeo4W Shell a menüből, Linuxon xterm/uxterm/lxterm) indítsuk el a Python értelmezőt. A példákban félkövéren szedtük, ami be kell írnia.

```
[siki@ale ~]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

A Python 3.4.x verzió indításához a `python3` parancsot kell beírnia.

1 A Python nyelv elnevezése a Monty Python Repülő Cirkuszról kapta. Guido Van Rossum a nyelv készítője nagy Monty Python rajongó.

Alapok

Először használjuk interaktívan a Python értelmezőt és ismerkedjünk meg az adattípusokkal, egyszerű kifejezésekkel. A >>> a Python prompt, ezután írjuk be az utasításainkat.

```
>>> 5 * 4
20
>>> 3**4                        # hatványozás
81
>>> (2 + 8) / 3                 # egész osztás
3
>>> (2 + 8) / 3.0               # lebegőpontos osztás
3.3333333333333335
>>> 56.12 // 12.34              # egész osztás
4.0
```

A Python a # karakter utáni szöveget a sor végéig megjegyzésnek értelmezi.
A kiszámított értékeket változókban megőrizhetjük, ilyenkor nem jelenik meg a művelet eredménye.

```
>>> a = 5 * 4
>>> print a                     # Python 3-ban print(a)
20
>>> l = True                    # Logikai érték True vagy False
>>> c = None                    # A None a definiálatlan értéket jelenti
>>> s = 'Hello world'           # karakterlánc, dupla aposztróf is jó
```

Python-ban nincs automatikus típus konverzió a változókra, konstansokra. A type függvénnyel le tudjuk kérdezni egy változó típusát.

```
>>> b = '16'
>>> a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> type(a)                                  # változó típus
<type 'int'>
>>> type(b)
<type 'str'>
>>> type(b) is str                           # is operátor
True
>>> a + int(b)
36
>>> a * b                                    # Karakterlánc többszörözése!
'16161616161616161616161616161616161616'
>>> s[0]                                     # Karakterlánc első eleme
'H'
>>> s[-1]                                    # Karakterlánc utolsó eleme
'd'
>>> s[0] = 'h'                               # Karakterlánc nem módosítható
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s[:5]                                    # Első 5 karakter
'Hello'
>>> s[6:]                                    # Karakterlánc vége
'world'
>>> s[-5:]
'world'
>>> s[6:8]
'wo'
>>> len(s)                                   # Karakterlánc hossza
11
>>> s[:5] + ' Dolly'
```

```
'Hello Dolly'
>>> c = 1
>>> a, c = c, a                          # két változó tartalom cseréje
```

A matematikai függvények (sin, cos, stb.) egy modulban találhatók (math).

```
>>> sin(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
>>> import math
>>> math.sin(1)                           # radián!
0.8414709848078965
>>> print '%6.4f' % math.sin(1)          # formatizálás
0.8415
```

Modulokat kétféleképpen importálhatunk. Az `import` *`modul_név`* parancs használata esetén a modul elemeire úgy hivatkozhatunk, hogy elé írjuk a modul nevét. A másik módszer esetén `from` *`modul_név`* `import` *`elem`*, nem kell a modul nevét használnunk a modul elemére hivatkozásnál, viszont ilyenkor könnyen név ütközés fordulhat elő.

```
>>> from math import sin
>>> sin(1)
0.8414709848078965
```

Ez utóbbi változat esetén vesszővel elválasztva a modul több felhasználandó elemét is megadhatjuk vagy * karakterrel az összes elemet importálhatjuk (pl. `from math import *`).

Adatszerkezetek

Az egyszerű változók mellett listákat is használhatunk. A lista egy rendezett adathalmaz, elemeire 0-tól kezdődő index-szel hivatkozhatunk.

```
>>> l1 = []                              # Üres lista
>>> l2 = ['alma', 5, 4]                  # Különböző adattípusok
>>> l2[0]
'alma'
>>> l2[1:]                               # Rész lista
[5, 4]
>>> len(l2)                              # Lista elemeinek száma
3
>>> l2[0] = 3                            # Lista elemek módosíthatók
>>> l2[3]                                # Nem létező elem
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> l2[3] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>> l2.append(5)                         # Lista bővítése
>>> print l2
[3, 5, 4, 5]
>>> l2[0:3] = 1                          # Lista részének cseréje
>>> print l2
[1, 5]
>>> del l2[0]                            # Lista elem törlése
>>> print l2
[5]
>>> l3 = [[1, 2], [6, 4]]                # Listák listája
>>> l3[0][1]
2
```

```
>>> help(list)
Help on class list in module builtins:
...
```

Listák összefűzését a + művelettel végezhetjük el, a * operátor a karakterláncokhoz hasonlóan ismétlést jelent. Az ***in*** művelettel megvizsgálhatjuk, hogy a lista tartalmaz-e egy elemet.
A listákkal kapcsolatban több függvényt használhatunk (például `sorted`, `reversed`, `min`, `max`) illetve további metódusok tartoznak hozzá (pl. `pop`, `sort`, `reverse`).
A tuple[2] egy nem módosítható lista, a listával azonos módon használható, kivéve a módosító utasításokat (a karakterlánchoz hasonlóan).

```
>>> t = (65, 6.34)
>>> t.append(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> t[0]
65
```

A listákkal, tuple-el kapcsolatban gyakran nagyon tömör kódot írhatunk az elemekkel végzendő műveletre. Ezt angolul list comprehension-nak, magyarul lista értelmezésnek vagy feldolgozásnak mondhatjuk.

```
>>> l = [2, 6, -3, 4]
>>> [x**2 for x in l]             # minden listaelem négyzete
[4, 36, 9, 16]
>>> [x**0.5 for x in l if x > 0]  # pozitív listaelemek gyöke
>>> import math
>>> map(math.sin, l)              # [math.sin(x) for x in l]
[0.9092974268256817, -0.27941549819892586, -0.1411200080598672,
-0.7568024953079282]
>>> [a for a in l if a%2]         # páratlan számok a listából
[-3]
```

A karakterláncokból, listákból tuple-okból halmazokat képezhetünk, a halmazokban ugyanaz az elem nem ismétlődhet.

```
>>> a = "abcabdseacbfds"
>>> s = set(a)
>>> s
set(['a', 'c', 'b', 'e', 'd', 'f', 's'])
>>> b = (1, 3, 2, 4, 3, 2, 5, 1)
>>> t = set(b)
>>> t
set([1, 2, 3, 4, 5])
>>> c = ['alma', 'szilva', 'barack', 'alma']
>>> u = set(c)
>>> u
set(['barack', 'szilva', 'alma'])
```

A halmazokkal műveleteket is végezhetünk.

```
>>> a = set('abcdefgh')
>>> b = set('fghijklm')
>>> a - b                         # halmazok különbsége
set(['a', 'c', 'b', 'e', 'd'])
>>> a | b                         # halmazok uniója
set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l'])
>>> a & b                         # halmazok metszete
set(['h', 'g', 'f'])
```

2 A tuple kifejezésre nincs elfogadott magyar fordítás, néhányan vektornak mondják, de az a listára is igaz lenne, adatbáziskezelésben rekord értelemben használják, a matematikában használják rá a szám n-es kifejezést, de a Pythonban nem csak számokat tárolhat, ... Én maradtam a tuple-nél (kiejtése: tapló :)

```
>>> a ^ b                          # halmazok szimmetrikus különbsége
set(['a', 'c', 'b', 'e', 'd', 'i', 'k', 'j', 'm', 'l'])
```

A szótár egy rendezetlen adathalmaz (asszociatív tömb, hash tábla). A szótár elemei lehetnek listák és további szótárak is.

```
>>> szotar = {}                                 # Üres szótár
>>> szotar['elso'] = 4                          # Egyszerűen bővíthető
>>> szotar[5] = 123                             # szám is lehet index
>>> szotar['elso'] = 'alma'                     # Módosítható
>>> print szotar
{'elso': 'alma', 5: 123}
>>> 'elso' in szotar                            # Létezik ez az index?
True
>>> tomb = {(1,1): 2, (1,2): 4, (2,1): -1, (2,2): 6}  # tuple az index
>>> tomb[1,1]
2
```

Az *in* műveletet halmazokra, listákra és tuple-okra is használhatjuk.
A Python értelmezőből a ***Ctrl/D*** billentyű kombinációval (csak Linuxon) vagy az `exit()` függvény hívásával léphetünk ki.

Python programok

A Python interaktív használata az ad-hoc, egyszerű feladatok megoldásához, egy-egy utasítás kipróbálásához megfelelő. A produktív felhasználás esetén a Python utasításainkat fájlokban (modulokban) helyezzük el és azokat hajtatjuk végre a Python értelmezővel. A megoldandó feladatot kisebb részekre bonthatjuk függvények illetve osztályok segítségével, a megoldás során elágazó és ciklus utasításokat használunk. A programok készítése előtt ki kell emelni a Python egy sajátosságát, a Python programokban a kód blokkokat a sor elején található szóközökkel jelöljük. Más nyelvekben a blokk elejét és végét jelölik meg, például '{' és '}' zárójelekkel. A nyelv ezen tulajdonsága kikényszeríti a könnyen olvasható kód készítését, másik oldalon nagyobb figyelmet igényel a sorok írásánál. A szóköz és tabulátor karaktereket nem lehet keverni, általában négy szóközös tagolást alkalmaznak. Összetettebb programok esetén valamilyen fejlesztő környezetet érdemes használni, ahol a hibákat hatékonyabban lokalizálhatjuk. Ezek közül az egyszerűen kezelhető IDLE, mely a Pythonnal együtt települ, most is rendelkezésünkre áll. Az OSGeo4W-ben az IDLE indításához készített parancsfájlban (\OSGeo4W\app\Python27\Lib\idlelib\idle.bat) a *pythonw.exe* fájl elérési útja sajnos hibás. A *start* kezdetű sort erre javítsa ki (egy sorba írandó!):

```
start „IDLE” „%CURRDIR%..\..\..\..\bin\pythonw.exe” „%CURRDIR%idle.pyw” %1 %2 %3 %4 %5
%6 %7 %8 %9
```

Az idle.bat fájlt csak a saját könyvtárából tudjuk elindítani. Egy tetszőleges könyvtárból az alábbi paranccsal indíthatjuk az IDLE-t (ha az OSGeo4W-t a C: egységre telepítette).

```
C:\munka> python C:\OsGeo4W\apps\Python27\Lib\idlelib\idle.bat
```

Linux- on dőljünk hátra és gépeljük be az `idle` parancsot :)
Készítsük el első programunkat, mely az első 100 egész szám összegét számítja ki (for ciklus). Nyissunk egy fájlt a kedvenc szövegszerkesztőnkben (pl. vim, nedit (Linux), Notepad++, Jegyzettömb (Windows)) vagy az ***idle*** környezetben és vigyük be a következő kódot:

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
""" első Python programom
    1 - 100 közötti egész számok összege
"""
s = 0
for i in range(1,101):              # a kettőspont jelzi a blokk elejét
    s += i
print s                             # ez a ciklus vége
```

Az első két sorban két speciális megjegyzés található. Az első sor a Linux buroknak szól, ha futtathatóvá tesszük a fájlt (`chmod +x sum.py`), akkor elegendő a fájl nevét megadni a parancssorban (pl. ha az aktuális könyvtárban található a programunk `./sum.py`, lásd a következőkben). A második sor a fájlban használt karakterkódolás adja meg.
Többsoros megjegyzéseket is elhelyezhetünk, ezek három dupla vagy szimpla aposztróffal kezdődnek és végződnek. Az ilyen megjegyzésekből automatizáltan generálhatunk fejlesztői dokumentációt. A programot *sum.py* névvel mentsük ki a háttértárolóra. A programunkat (modulunkat) futtathatjuk a Python értelmezőből:

```
[siki@ale qgis]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sum
5050
>>> help(sum)
Help on module sum:

NAME
    sum

FILE
    /home/siki/qgis/sum.py
...
```

A parancssorból közvetlenül is futtathatjuk a programokat, általában ez a szokásos.

```
[siki@ale ~]$ python sum.py
5050
[siki@ale ~]$
```

Ez utóbbi változat lehetővé teszi, hogy más szkriptekbe beépítsük a programunkat.
Vagy futtathatóvá téve a program fájlt, Linuxon így is működhet.

```
[siki@ale ~]$ chmod +x sum.py             # ez csak egyszer kell!
[siki@ale ~]$ ./sum.py
```

Módosítsuk a programunkat, hogy a parancssorban megadhassuk az összegzés felső határát. Ehhez a `sys` modul `argv` listáját használhatjuk.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
""" első Python programom
    1 - argv[0] közötti egész számok összege
"""
from sys import argv
if len(argv) < 2:
    print "usage: sum.py number"
....exit(1)
s = 0
for i in range(1,int(argv[1])+1):  # a kettőspont jelzi a blokk elejét
    s += i
print s                            # ez a ciklus vége
```

```
[siki@ale ~]$ ./sum.py 50
1275
```

Windows operációs rendszeren a `python sum.py 50` paranccsal futtathatjuk a programunkat, ha a *sum.py* fájl az aktuális könyvtárban található.
Az első n természetes szám összegzésére az alábbi, a Python logikához közelebb álló megoldás is adható:

```
>>> numbers = range(1000)
>>> sum([n**2 for n in numbers])  # lista generálás és összegzés sum fv.
332833500
>>> sum(n**2 for n in numbers)     # változó hosszúságú paraméterlista
332833500
```

További algoritmus szerkezeteket is biztosít a Python. A programozás során a ***for*** ciklus mellet ***while*** ciklust is használhatunk, a ciklusok végrehajtását a ***continue*** és ***break*** utasításokkal befolyásolhatjuk. Egy speciális utasítás a ***pass***, mely nem csinál semmit, helykitöltésre használható. A döntéseknél az ***if elif else*** utasítást használhatjuk. Lásd a további példákban.

Szövegfájlok kezelését is egyszerűen megoldhatjuk Pythonban. Mintaként készítsünk egy programot, mely egy szövegfájlból kiválasztja a *hilti* szót tartalmazó sorokat.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

try:
    f = open('minta.txt', 'r')        # fájl nyitás hibakezeléssel
except:
    print "error"
    exit(1)
for line in f:                        # soronként olvasás
    if line.find('hilti') >= 0:
        print line
f.close()
```

A fenti programot futtatva egyrészt azt vehetjük észre, hogy minden kiírt sor után van egy üres sor, másrészt azt, hogy a pontszám oszlopban szereplő *hilti* szöveget is megtalálta a programunk, amit nem biztos, hogy szeretnénk. Az első problémát az okozza, hogy az újsor karaktert is tartalmazza a beolvasott sor. Ezt egyszerűen megoldhatjuk a `strip()` függvény használatával, a `print` sorban: `print line.strip()`
A második probléma megoldására a szabályos kifejezéseket használhatjuk, melyek a re standard modulban találhatók.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import re

try:
    f = open('minta.txt', 'r')
except:
    print "error"
    exit(1)
for line in f:
    line = line.strip()
    if re.search('hilti$', line) >= 0:   # sorvégi egyezés
        print line
f.close()
```

Saját függvényeket definiálhatunk a ***def*** kulcsszóval. A függvényeknek nem kötelező értéket visszaadniuk. A visszaadott érték összetett adatszerkezet is lehet, például lista vagy szótár is.
Néhány egyszerű függvény (a *func.py* fájlba írjuk be a következőket):

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
from math import sqrt

def celsius(fahrenheit=0):
    "celsius to fahrenheit conversion"      # doc comment
    return (fahrenheit - 32) * 5.0 / 9.0
```

```
def root(a, b, c):
    "roots of quadratic equation"
    d = b**2 - 4 * a * c
    if d == 0:
        return -b / 2.0 / a
    elif d > 0:
        return ((-b + sqrt(d)) / 2.0 / a, (-b - sqrt(d)) / 2.0 / a)
    else:
        pass    # complex roots solved later
    return None

def f(n):
    "factorial calculation"
    f = 1
    while n > 0:
        f *= n
        n -= 1
    return f

def fact(n):
    "factorial calculation"
    f = 1
    for i in range(1, n+1):
        f *= i
    return f

# recursive function
def factorial(n):
    "faktorial calculation"
    if n <= 1:
        return 1
    return n * factorial(n-1)
```

A függvény paraméterekhez adhatunk alapértelmezett értéket, lásd `fahrenheit` paraméter a `celsius` függvényben. A függvények hívásánál a paraméterek értékét a definíció sorrendjében adhatjuk meg vagy a paraméter nevének megadásával tetszőleges sorrendben.

```
[siki@ale python_samples]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from func import *
>>> celsius()                       # 0 fahrenheit
-17.77777777777778
>>> celsius(-20)
-28.88888888888889
>>> root(c=5, b=-2, a=2)            # nincs valós gyök
>>> root(c=-5, b=-2, a=2)
(2.1583123951777, -1.1583123951777)
>>> celsius('alma')                 # paraméternek nincs típusa!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "func.py", line 7, in celsius
    return (fahrenheit - 32) * 5.0 / 9.0
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Feladatok

Mi történne, ha a `celsius` függvénybe a `5.0 / 9.0` helyett `5 / 9`-et írnánk?
Próbáljuk meg a $0\ x^2 - 3\ x + 5 = 0$ egyenlet gyökeit kiszámítani (root(0, -3. 5))! Írjuk át a függvényt a `try` utasítás használatával!
Próbáljuk kiszámítani a 1000! mindhárom függvényünkkel (f, fact, factorial)! Írassuk ki a függvények „doc sztringjét” (factorial.__doc__).

Mi történik, ha negatív paramétert adunk meg a faktoriális függvénynek?
Készítsünk függvényt a irányszög számításra (math.atan2)!
Készítsünk függvényt a DMS szögek radiánba átváltására!

Objektumok

A Python objektum orientált programozást a 2D-s pontok osztályának elkészítésén keresztül mutatjuk be. A Python nyelvben minden osztály (például a lista vagy a szótár is, de a függvények is).
Egy *point2d.py* nevű fájlban kezdjük el az osztály kódjának elkészítését.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Point2D(object):                # object a bázis osztály
    """ kettő dimenziós pontok osztálya
    """
    def __init__(self, east = 0, north = 0):  # __init__ a konstruktor
        """ Initialize point

            :param east: első koordináta
            :param north: második koordináta
        """
        self.east = east              # tagváltozó létrehozása
        self.north = north

    def abs(self):
        """ calculate length of vector (absolute value)
            :returns: absolute value
        """
        return (self.east**2 + self.north**2)**0.5
```

A self változó az osztály minden (nem statikus vagy osztály) metódusának az első paramétere és az objektum példányt jelenti, ezen keresztül hivatkozhatunk az objektum elemeire pl. `self.east`. A többsoros megjegyzés a sphinx programnak megfelelően készült, hogy automatikusan lehessen dokumentációt generálni a kódból.
Próbáljuk ki a fenti kódunkat.

```
[siki@ale ~]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from point2d import Point2D
>>> p = Point2D()                 # példányosítás, 0, 0 pont
>>> p1 = Point2D(5)               # 5, 0 pont
>>> p2 = Point2D(2, -2)           # 2 – 2 pont
>>> print p2.east                 # tagváltozó értéke
2
>>> print p2.north
-2
>>> p.abs()
0.0
>>> p2.abs()
2.8284271247461903
>>> print p2
<point2d.Point2D object at 0x2708b50>      ???????????
>>> print p2.__dict__
{'east': 2, 'north': -2}
>>> print c2.__doc__
 kettő dimenziós pontok osztálya

>>> print p2.abs.__doc__
 calculate length of vector (absolute value)
```

```
            :returns: absolute value
>>> help(Point2D)
Help on class Point2D in module point2d:

class Point2D(__builtin__.object)
 |  kettő dimenziós pontok osztálya
 |
 |  Methods defined here:
...
```

A `print p2` utasítás nem azt az eredményt adja amit szeretnénk. Az egyes osztályokhoz speciális függvényeket definiálhatunk (mint pl. az `__init__`), ezek jellemzője, hogy két aláhúzás karakterrel kezdődik és végződik a nevük. Az `__init__` függvény az úgynevezett konstruktor, ez fut le a példányok létrehozásakor, itt biztosíthatjuk, hogy ne legyen inicializálatlan tagváltozónk. A destruktort a __del__ függvény implementálásával valósíthatjuk meg. A dinamikus tárfoglalás hiányában a Pythonban erre ritkábban van szükség.
A `print` utasítás az osztály `__str__` metódusát hívja meg. A fenti példában ennek alapértelmezett változatának eredményét láthattuk az ***object*** osztályból.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Point2D(object):              # object a bázis osztály
    """ kettő dimenziós pontok osztálya
    """
...
    def __str__(self):
        """ String representation of the 2D point
            :returns: point coordinates as string (e.g. 5; 3)
        """
        return "%.3f; %.3f" % (self.east, self.north)
```

A % operátor a formatizált kiíratást teszi lehetővé. Ez egy kicsit elavult, helyette a szövegláncok `format` metódusát javasolják pl. `"{0:.3f}; {1:.3f}".format(self.east, self.north)`.
Próbáljuk ki az osztályunkat!

```
[siki@ale ~]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from point2d import Point2D

>>> p2 = Point2D(2, -2)           # 2; -2 pont
>>> print p2
2.000 ;-2.000
```

A Python nyelvben az osztályokhoz további speciális függvényekkel műveleteket is definiálhatunk. A pontokat mint helyvektorokat is használhatjuk, készítsük el a helyvektorok összeadását.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Point2D(object):              # object a bázis osztály
...
    def __add__(self, p):
        """ Add two point
            :param p: point to add
            :returns: the sum of the two point (vector)
         """
        return Point2D(self.east + p.east, self.north + p.north)
```

```
[siki@ale ~]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from point2d import Point2D

>>> p1 = Point2D(1, -2)
>>> p2 = Point2D(4, 1)
>>> print p1 + p2
5.000; -1.000
```

Megjegyzés: a Pythonban dinamikusak az osztályok, az osztály példányait is bővíthetjük tagváltozókkal és metódusokkal, de ezeket az osztály többi példányából nem tudjuk használni.

Elzártság

Az elkészített Point2D osztályunk az objektum orientált programozás egyik fontos kívánalmának, az elzártságnak nem tesz eleget. A pont osztály egy példányának a koordinátáit közvetlenül megváltoztathatjuk, ez azzal a következménnyel járhat, hogy:

1. a példány tagváltozóit az osztály felhasználója egy programhibából következően is átírhatja,
2. a példány tagváltozóit ellenőrzés nélkül is felül lehet írni.

```
[siki@ale ~]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from point2d import Point2D

>>> p1 = Point2D(1, -2)
>>> p1.east = 5
>>> p1.north = 10
>>> print p1
5.000; -1.000
```

Ennek kivédésére privát tagváltozókat („_” karakterrel kezdődő név) és getter/setter metódusokat használhatunk. Írjuk át az osztályunkat, hogy egy-egy metóduson keresztül lehessen megváltoztatni a tagváltozókat. A Python osztályok privát tagváltozóinak neve aláhúzás karakterrel kezdődik. A két aláhúzással kezdődő és végződő nevek a Python nyelv elemei.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Point2D(object):                # object a bázis osztály
    """ kettő dimenziós pontok osztálya, getter/setter változat
    """
    def __init__(self, east = 0, north = 0):  # __init__ a konstruktor
        """ Initialize point

            :param east: első koordináta
            :param north: második koordináta
        """
        self.setEast(east)
        self.setNorth(north)

    def setEast(self, east):
        self._east = east

    def getEast(self):
        return self._east

    def setNorth(self, north):
        self._north = north

    def getNorth(self):
        return self._north
```

```
    def abs(self):
        """ calculate length of vector (absolute value)

            :returns: absolute value
        """
        return (self._east**2 + self._north**2)**0.5

   def __str__(self):
        """ String representation of the 2D point

            :returns: point coordinates as string (e.g. 5; 3)
        """
        return "%.3f; %.3f" % (self._east, self._north)

    def __add__(self, p):
        """ Add two point
            :param p: point to add
            :returns: the sum of the two point (vector)
         """
        return Point2D(self._east + p._east, self._north + p._north)
```

Ebben a változatban már könnyen implementálhatjuk például azt, hogy az EOV koordináták esetén az *east* (y) koordinátának 400000-nél nagyobbnak, a *north* (x) koordinátának 400000-nél kisebbnek kell lennie:

```
    def setEast(self, east):
        if east > 400000:
            self._east = east
        else:
            raise ValueError('east must be greater than 400000')

    def setNorth(self, north):
        if north > 0 and north < 400000:
            self._north = north
        else:
            raise ValueError('north must be less than 400000')
```

```
[siki@ale python_samples]$ python
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from point2d_1 import Point2D
>>> p1 = Point2D(10, 600000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "point2d_1.py", line 13, in __init__
    self.setEast(east)
  File "point2d_1.py", line 20, in setEast
    raise ValueError('east must be greater than 400000')
ValueError: east must be greater than 400000
>>>
```

Ebben a változatban viszont a getter/setter metódusokat kell meghívni, ahelyett, hogy csak a tagváltozó nevét adnánk meg. Emiatt többet kell gépelnünk és talán kevésbé olvasható a kód.

Megjegyzés: a Pythonban a p1._east = 1 utasítással kikerülhetjük setter függvényt. Az „_” egy konvenció szerint jelzi a tagváltozó védett (protected) voltát, de nem kényszeríti ki!

Végül nézzük meg azt a megoldást, mely lehetővé teszi a tagváltozóra közvetlen hivatkozást és a getter/setter metódusok közvetett hívását. A Pythonban javasolt megoldás a *@property* illetve a *@név.setter* dekorátor használata.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Point2D(object):                # object a bázis osztály
```

```
    """ kettő dimenziós pontok osztálya, getter/setter változat
    """
    def __init__(self, east = 0, north = 0):  # __init__ a konstruktor
        """ Initialize point

            :param east: első koordináta
            :param north: második koordináta
        """
        self.east = east              # setter implicit hívása
        self.north = north

    @property
    def east(self):
        return self._east

    @east.setter
    def east(self, east):
        if east > 400000:
            self._east = east
        else:
            raise ValueError('east must be greater than 400000')

    @property
    def north(self):
        return self._north

    @north.setter
    def north(self, north):
        if north > 0 and north < 400000:
            self._north = north
        else:
            raise ValueError('north must be less than 400000')

    def abs(self):
        """ calculate length of vector (absolute value)

            :returns: absolute value
        """
        return (self.east**2 + self.north**2)**0.5

    def __str__(self):
        """ String representation of the 2D point

            :returns: point coordinates as string (e.g. 5; 3)
        """
        return "%.3f; %.3f" % (self.east, self.north)

    def __add__(self, p):
        """ Add two point
            :param p: point to add
            :returns: the sum of the two point (vector)
         """
        return Point2D(self.east + p.east, self.north + p.north)

if __name__ == "__main__":
    p1 = Point2D(654123.12, 243127.56)
    p2 = Point2D(655231.69, 243431.27)
    print p1
    p1.east += 2
    print p1
```

A fenti kód végére az osztály tesztelésére szolgáló kódot is elhelyeztünk. Az *if __name__* utáni kód csak akkor fut le, ha nem egy másik programba importáljuk a modulunkat.

Megjegyzés: a Pythonban osztály tagváltozókat is létrehozhatunk, melyen minden példány osztozkodik. Ilyet használhatunk például a példányok számának nyilvántartására. Lehetőségünk van statikus illetve osztály tagfüggvények létrehozására.

Öröklődés, többértelműség

Az öröklődés segítségével már létező osztályok funkcionalitását bővíthetjük ki, anélkül, hogy a szülő osztály kódján változtatnánk. Az öröklődés bemutatására készítsünk egy Point3D osztályt. A származtatott osztályban a szülő osztály metódusait felülbírálhatjuk szükség esetén, illetve újabb metódusokat hozhatunk létre. A származtatott osztály örökli a szülő osztály tagváltozóit és újabb tagváltozókat definiálhatunk. Az objektumorientált programozásban a többértelműség azt jelenti, hogy több ugyanolyan nevű függvényt hozhatunk létre eltérő paraméterlistával (a paraméterek típusa vagy száma eltérő). Ezt a Pythonban csak korlátozottan alkalmazhatjuk, mivel a függvényparaméterekhez nem adhatunk meg változó típust. Az alábbi példában az `__abs__` függvény egy példa. Ezt akkor hívja meg a Python értelmező, ha az `abs` függvény argumentuma egy Point3D típusú változó, de emellett egész vagy valós értékekkel is használhatjuk.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
from point2d_2 import Point2D

class Point3D(Point2D):
    """ 3D points

            :param east: első koordináta
            :param north: második koordináta
            :param elev: magasság
    """

    def __init__(self, east = 0, north = 0, elev = 0):
        super(Point3D, self).__init__(east, north)
        self.elev = elev

    @property
    def elev(self):
        return self._elev

    @elev.setter
    def elev(self, elev):
        if elev < 10000:
            self._elev = elev
        else:
            raise ValueError('elev must be less than 10000')

    def abs(self):
        """ calculate length of vector (absolute value)

            :returns: absolute value
        """
        return (self.east**2 + self.north**2 + self.elev**2)**0.5
   def __abs__(self):
        return self.abs()

    def __str__(self):
        """ String representation of the 2D point

            :returns: point coordinates as string (e.g. 5; 3)
        """
        return super(Point3D, self).__str__() + "; %.3f" % self.elev

if __name__ == "__main__":
    p1 = Point3D(600000,200000,100)
    print p1
    print p1.abs()
    print abs(p1)
    print abs(-1)
    print abs('abc')
```

Feladatok

Számítsuk ki a pont (helyvektor) irányszögét az origótól (atan2 függvény)!

Készítsük el a pontok (helyvektorok) különbségét (__sub__)!
Készítsük el a += operátort a pontokra (__iadd__)!
Származtassunk a 2D-s pont osztályunkból egy kör osztályt!

Python (térinformatikai) csomagok használata

Mielőtt egy programozási feladatot meg szeretnénk oldani érdemes körülnézni az interneten, hátha valaki már már készített egy Python modult, mely segít a megoldásban. Az alábbiakban néhány térinformatikai példát mutatok be nyílt forráskódú Python modulok felhasználásával.

Többszálú programozás

Lehet hatékonyabb megoldást találni az egész számok összegére? Amennyiben a számítógépünkben több processzor vagy a processzorban több mag található, akkor igen. Bontsuk a magok számának megfelelő párhuzamos szálra (thread) a feladatot, melyek párhuzamosan futhatnak.

```
import multiprocessing
import threading
from sys import argv

class SumThread(threading.Thread):
    """ thread for partial sum
    """
    def __init__(self, low, high):
        super(SumThread, self).__init__()
        self.low = low
        self.high = high
        self.total = 0

    def run(self):
        for i in range(self.low, self.high):
            self.total += i
            print "*"+str(i)+"*"

if len(argv) < 2:
    print "usage: {0} number".format(argv[0])
else:
    abshigh = int(argv[1])
    n = multiprocessing.cpu_count()
    part = abshigh / n
    threads = []
    low = 1
    for i in range(n):
        if i == n-1:
            high = abshigh + 1
        else:
            high = low + part
        threads.append(SumThread(low, high))
        threads[i].start()
        print "thread {0} started {1} - {2}".format(i, low, high)
        low += part
    for i in range(n):
        threads[i].join()
        print "thread {0} stopped subtotal {1}".format(i, threads[i].total)
    total = 0
    for i in range(n):
        total += threads[i].total
    print total
```

A `SumThread` osztály `run` metódusát hajtja végre a Python a szál indításakor, `start` metódus. Nézze meg a program üzeneteit. Milyen sorrendben jelennek meg az összegzett számok?

OGR, ezdxf

Nézzünk egy egyszerű példát. Oldjuk meg azt a feladatot, hogy egy shape fájl egyik attribútumából készítsünk feliratot egy AutoCAD DXF fájlban, felirat beszúrási pontja legyen a geometriai elem súlypontja. Számos olyan könyvtárat találhatunk az interneten, melyek DXF illetve shape fájl írását és olvasását oldják meg. Ezek közül a legkomplexebb szolgáltatást nyújtó legszélesebb körben alkalmazott az OGR könyvtár. A C++ nyelven írt OGR könyvtárhoz úgynevezett Python kötéseket készítettek, hogy Python programokból elérhetők legyenek az OGR funkciók. Az ogr könyvtár a térinformatikai logikát követi, ezért szöveget nem tud a DXF fájlba írni. Erre a célra az ezdxf Python modult fogjuk használni. Az ezdxf modult külön telepítenünk kell a gépünkre. Töltsük le a zip vagy tar.gz fájl a https://pypi.python.org/pypi/ezdxf/0.6.2 oldalról és tömörítsük ki egy könyvtárba. Lépjen be a *ezdxf-0.6.2* alkönyvtárba és adja ki az alábbi parancsot a modul telepítéséhez:

```
[siki@ale ~/ezdxf-0.6.2]$ python setup.py install
```

A telepítéshez olyan jogosultság kell, mellyel a Python rendszer könyvtárakba írhatunk.

Ezután már elkészíthetjük a programunkat a *pnt2txt.py* fájlba.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
""" shp attribute to DXF text
"""
from osgeo import ogr
import ezdxf
from os import path, unlink
import sys

if len(sys.argv) < 4:
   print "usage: %s <input_shp> <output_dxf> <txt_column> \
       [rotation_column] [text_height]" % sys.argv[0]
   exit(1)

# input layer
shpfile = sys.argv[1]
if not path.exists(shpfile):
   print "Shape file not found"
   exit(2)

# open input shape
inDriver = ogr.GetDriverByName("ESRI Shapefile")
inDataSource = inDriver.Open(shpfile, 0)
inLayer = inDataSource.GetLayer()

# output dxf file
dxffile = sys.argv[2]
dxf = ezdxf.new(dxfversion='AC1015') # AutoCAD R2000
if path.exists(dxffile):
    unlink(dxffile)
modelSpace = dxf.modelspace()
col = sys.argv[3]        # column name for label

rot = 0                  # default rotation 0 degree
if len(sys.argv) > 4:
    rotcol = sys.argv[4]

h = 1                    # default text height
if len(sys.argv) > 5:
    h = float(sys.argv[5])

# process each feature
for feature in inLayer:
    geom = feature.GetGeometryRef()
    if geom.GetGeometryName() == "POLYGON":
```

```
        p = geom.Centroid()    # insert text at centroid
    else:
        p = geom.GetPoint(0)   # insert text at first point
    label = unicode(feature.GetField(col), 'cp1250', 'ignore')
    rot = feature.GetField(rotcol)
    modelSpace.add_text(label, \
        dxfattribs={'height': h, 'rotation': rot}).set_pos(p)

dxf.saveas(dxffile)
```

Próbáljuk ki a programunkat:

```
[siki@ale ~]$ python pnt2txt varos.shp varos.dxf NEV1 FORGATAS 1000
```

Ellenőrizzük az eredményt (varos.dxf) QCAD-ben vagy AutoCAD-ben!

GDAL

Magyarországot területét tartalmazó domborzatmodellből készítsünk egy kimutatást egy megadott magasság alatti terület nagyságáról. A program elkészítéséhez a GDAL könyvtárat használjuk.

```
from osgeo import gdal
from gdalconst import *
from math import *
import struct

data = gdal.Open("mo.tif", GA_ReadOnly) # open dtm image
if data is None:
    print "Open error"
    exit(1)

geotr = data.GetGeoTransform()
band = data.GetRasterBand(1)     # get the only band
fmt = "<" + ("f" * band.XSize)   # float32 data
pixel_area = abs(geotr[1] * geotr[5])
area = 0.0                       # variable for area sum

for y in range(band.YSize):
    scanline = band.ReadRaster(0, y, band.XSize, 1, band.XSize, 1,
band.DataType)
    values = struct.unpack(fmt, scanline)
    for value in values:
        if value < 300 and value > 0:
            area += pixel_area
print area
```

Shapely

A Shapely modul segítségével a GEOS könyvtár funkcionalitását érhetjük el. A GEOS C++ nyelven írt, geometriai számítások végrehajtására készített programkönyvtár. A PostGIS is a GEOS-t használja. Az alábbi példában a varosok köré 30 km-es övezetet készítünk egy új rétegbe.

```
#! /usr/bin/python

# -*- coding: UTF-8 -*-

from osgeo import ogr
import shapely.wkt
import os,os.path,shutil
import os,os.path,shutil

shapefile = ogr.Open("varos.shp")       # input point shape
layer = shapefile.GetLayer(0)

driver = ogr.GetDriverByName("ESRI Shapefile")
```

```
outshp = "varosb.shp"
if os.path.exists(outshp):              # remove ouput shape
    driver.DeleteDataSource(outshp)
dstFile = driver.CreateDataSource("varosb.shp") # create output shape
dstLayer = dstFile.CreateLayer("layer", geom_type=ogr.wkbPolygon)
field = ogr.FieldDefn("id", ogr.OFTInteger)     # create output field
dstLayer.CreateField(field)

for i in range(layer.GetFeatureCount()):
    feature = layer.GetFeature(i)       # get point ffom input
    geometry = feature.GetGeometryRef()
    wkt = geometry.ExportToWkt()        # change to wkt format
    p = shapely.wkt.loads(wkt)          # convert to shapely geom
    pb = p.buffer(30000)                # 30 km buffer
    wktb = shapely.wkt.dumps(pb)        # export to wkt
    feature = ogr.Feature(dstLayer.GetLayerDefn())
    feature.SetGeometry(ogr.CreateGeometryFromWkt(wktb))
    feature.SetField("id", i)           # set id
    dstLayer.CreateFeature(feature)

dstFile.Destroy()                       # close output
```

urllib, urllib2

A Pythonból WEB szolgáltatásokat is elérhetünk az *urllib* illetve *urllib2* modulokkal. Az alábbi példában egy HTTP GET kérés kezelését mutatom be a tanszéki honlap szolgáltatásához.

```
>>> import urllib
>>> req = urllib.urlopen(
        """http://www.agt.bme.hu/on_line/etrs2eov/etrs2eov.php?
           e=650000&n=240000&sfradio=single&format=TXT""").read()
>>> print req
1 19.0474474 47.5039331
```

A következő példa egy HTTP POST kérés kezelését mutatja be.

```
>>> import urllib
>>> import urllib2
>>> url = 'http://www.agt.bme.hu/on_line/etrs2eov/etrs2eov.php'
>>> val = { 'e' : 650000, 'n' : 240000, 'sfradio' : 'single', \
    'format' : 'TXT' }
>>> data = urllib.urlencode(val)
>>> req = urllib2.Request(url, data)
>>> res = urllib2.urlopen(req)
>>> print res.read()
1 19.0474474 47.5039331
```

Feladatok

Írja át a 300 méter alatti terület kiszámítását egy függvénnyé, melynek paramétere a magasság!
Írja át a 300 méter alatti terület kiszámítását, hogy egy raszterbe ábrázolja a területet!
Írja át a HTTP GET illetve POST kéréseket, hogy a koordinátákat a parancssorból beolvasó program legyen!

Kapcsolódás relációs adatbázisokhoz

Python nyelvből számos relációs adatbáziskezelőhöz kapcsolódhatunk. A Python DB API segítségével olyan kódot készíthetünk, mely minimális mértékben függ csak a használt adatbáziskezelőtől. A DB API moduljai objektumokat kínálnak a programozóknak és nevük általában a „db” betűkkel végződik, pl. MySQLdb (MySQL), de psycop2 (PostgreSQL). A megfelelő adatbáziskezelőhöz tartozó DB API modult telepítenünk kell a számítógépünkre, ezután a Python programunkba importálhatjuk a modult.

A megfelelő adatbáziskezelőhöz tartozó DB API modult telepítenünk kell a számítógépünkre, ezután a Python programunkba importálhatjuk a modult.

```
    import psycopg2 as db    # PostgreSQL meghajtó betöltése
```

A Phyton programból egy adatbázis kapcsolat objektumot kell létrehozni, majd ezen a kapcsolaton keresztül SQL utasításokat hajthatunk végre az adatbázison. Lekérdezésekhez úgynevezett cursor-t használunk, melynek segítségével akár egyesével kaphatjuk meg és dolgozhatjuk fel a lekérdezés eredményét.
A kapcsolat objektum létrehozásakor több paramétert adhatunk meg, melyek többségére alapértelmezett érték is rendelkezésre áll.

```
con = db.connect(host="számítógép", port="port",
    database="adatbázis_név", user="felhasználó", password="jelszó")
```

Alapértelmezett értékek		
host	localhost	A saját számítógépünk, ezen a gépen fut az adatbáziskezelő
port	5432	Adatbáziskezelő függő, pl. PostgreSQL: 5432, MySQL 3306
user		Aktuálisan bejelentkezett felhasználó (*ident* azonosítás esetén)

Az alábbi program egy adatbázis tábla tartalmát listázza ki a képernyőre:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import psycopg2 as db  # PostgreSQL meghajtó betöltése
import sys

con = None             # kapcsolat változó inicializálása

try:
    con = db.connect(database='postgis') # aktuális felhasználó
    cur = con.cursor()                   # cursor a lekérdezéshez
    cur.execute('SELECT * FROM varos')   # adatok lekérdezése
    rec = cur.fetchone()                 # következő sor a lekérdezésből
    while rec:
        print rec
        rec = cur.fetchone()
except db.DatabaseError, e:
    print 'Hiba: %s' % e                 # hibaüzenet kiírása
    sys.exit(1)
finally:
    if con:                              # kapcsolat lezárása
        con.close()
```

Felhasznált, ajánlott irodalom

Magyar nyelven

Gyűjtő oldal: http://pythontutorial.pergamen.hu/
Python oktató: http://pythontutorial.pergamen.hu/downloads/tut.pdf
Tanuljunk meg programozni Python nyelven: http://mek.oszk.hu/08400/08435/08435.pdf
Python minireferencia: http://www.blender.hu/index.php?page=tut&l=sdoc/pythonref

Angolul

Dive into Python: http://www.diveintopython.net/ illetve http://www.diveintopython3.net/
Python course: http://www.python-course.eu/
Python docs: https://docs.python.org/2/ illetve https://docs.python.org/3/
Beginners guide: https://wiki.python.org/moin/BeginnersGuide
Python tutorial: https://docs.python.org/2/tutorial/
GDAL/OGR: http://gdal.org/python/
Shapely: https://pypi.python.org/pypi/Shapely
PostGIS: https://wiki.python.org/moin/UsingDbApiWithPostgres
Standard modulok: https://docs.python.org/2/library/index.html
Modul index: https://pypi.python.org/pypi
Writing Idiomatic Python (Jeff Knupp):
https://www.jeffknupp.com/writing-idiomatic-python-ebook/

Fejlesztőkörnyezetek

IDLE a Python része
PyCharm http://www.jetbrains.com/pycharm/
PyScripter https://code.google.com/p/pyscripter/
Eclipse/PyDev http://pydev.org/
NetBeans http://wiki.netbeans.org/Python

Szoftver fejlesztési ciklusok:

1. A programozó elkészíti az által hibátlannak vélt programot.
2. A tesztelés során 20 hibát találnak.
3. A programozó kijavít 10 hibát és elmagyarázza a tesztelőknek, hogy a további 10 igazából nem is hiba.
4. A tesztelők szerint 5 hibajavítás nem megfelelő és találnak 15 új hibát.
5. A harmadik és negyedik lépés ismétlődik háromszor.
6. A piaci nyomás miatt és a túl korai bejelentés miatt tarthatatlanul optimista programfejlesztési periódus ellenére piacra dobják a terméket.
7. A felhasználók 137 új hibát találnak.
8. Az eredeti programozó már sehol sem található, miután felvette a járandóságát.
9. Az új programozó csapat kijavítja majdnem az összes 137 hibát,de elkövetnek újabb 456 hibát.
10. Az eredeti programozó képeslapot küld az alulfizetett tesztelőknek a Fiji szigetekről. A tesztelő csapat felmond.
11. A cég az utolsó változattal szerzett profitból felvásárolja a konkurens cég szoftverét, melyben 783 hiba van.
12. Az igazgatótanács új vezérigazgatót nevez ki, aki felvesz egy új programozót, hogy teljesen újraírja a programot.
13. A programozó elkészíti az által hibátlannak vélt programot...

Budapest, 2015. január 15.

Pythonic kód készítése

A programozási nyelvek többségében alkalmazott program struktúráknál a Python sokszor hatékonyabb, olvashatóbb, egyszerűbb megoldásokat kínál. Az alábbi táblázatban néhány ilyet foglalunk össze.

Általános megoldás	Pythonic megoldás
Két változó tartalmának felcserélése	

`tmp = a`
`a = b`
`b = temp`
	`a, b = b, a`

Többszörös értékadás	

`A = 0`
`B = 0`
`C = 0`
	`A = B = C = 0`

Érték egy intervallumba esik-e?	
`if a < b and b < c:` `  print ('közötte')`	`if a < b < c:` `  print('közötte')`
Egy érték azonos-e több érték közül eggyel?	
`if = a == 'alma' or a == 'körte' or a == 'szilva':` `  print ('szeretem')`	`if a in ('alma', 'körte', 'szilva'):` `  print ('szeretem')`
Rövid feltételes kiértékelés	

`if nagy:`
`  a = 2`
`else:`
`  a = 1`
	`a = 2 if nagy else 1`

Ciklusok	

`i = 0`
`while i < 10:`
`  print i`
`  i += 1`
	`for i in range(10):`
	`  print i`

`lista = ['Peti', 'Franciska', 'Regő']`
`index = 0`
`for elem in lista:`
`  print('{} {}'.format(index, elem))`
`  index += 1`
	`lista = ['Peti', 'Franciska', 'Regő']`
	`for index, elem in enumerate(ista):`
	`  print('{} {}'.format(index, elem))`

`lista = ['Peti', 'Franciska', 'Regő']`
`index = 0`
`while index < len(lista):`
`  print(lista[index])`
`  index += 1`
	`lista = ['Peti', 'Franciska', 'Regő']`
	`for elem in lista:`
	`  print(elem)`

`hiba = False`
`for cim in minden_cim():`
`  if hibas_cim(cim):`
`  hiba = True`
`  print('Hibás cím')`
`  break`
`if not hiba:`
`  print('Minden cím jó')`
	`for cim in minden_cim():`
	`  if hibas_cim(cim):`
	`  print('Hibás cím')`
	`  break`
	`else:`
	`  print('minden cím jó')`
	`# az else a for-hoz tartozik és akkor fut`
	`# le, ha nem break-kel léptünk ki a`
	`# ciklusból`

Függvény hívások láncolása	

Általános megoldás	Pythonic megoldás

`konyv_info = '  A zabhegyező: Salinger  '`
`konyv = konyv_info.strip()`
`konyv = konyv.upper()`
`konyv = konyv.replace(':', ' by')`
	`konyv_info = '  A zabhegyező: Salinger  '`
	`konyv =`
	`konyv_info.strip().upper().replace(':', ' by')`

Szöveglánc vagy lista fordított sorrendbe állítása	

`s = 'python'`
`w = ''`
`for i in range(len(s)-1, -1, -1):`
`  w += s[i]`
	`s = 'python'`
	`w = s[::-1]`

Lista összefűzése szövegláncca	

`l = ['egy', kettő', 'sok']`
`s = ''`
`for i in l:`
`  s += i + ','`
	`l = ['egy', 'kettő', 'sok']`
	`s = ','.join(l)`

Két szöveglánc váltakozó összefűzése ('abcd', '1234' → 'a1b2c3d4')	

`a = 'abcd'`
`b = '1234'`
`s = ''`
`for i in range(len(a)):`
`  s += a[i] + b[i]`
	`a = 'abcd'`
	`b = '1234'`
	`s = ''.join(''.join(i) for i in zip(a, b)]`

Van hamis (nulla/False) elem a listában?	

`l = [2, 4, 0, 2, 5, 6]`
`for i in l:`
`  if not l[i]:`
`  print False`
`  break`
`else:`
`  print True`
	`l = [2, 4, 0, 2, 5, 6]`
	`all(l)  # False`
	`all(l[3:])  # True`

Skaláris szorzat	

`a = [1, 4, -2]`
`b = [-2, 3, -4]`
`s = 0`
`for i in range(len(a)):`
`  s += a[i] * b[i]`
	`a = [1, 4, -2]`
	`b = [-2, 3. -4]`
	`s = sum([x * y for (x,y) in zip(a,b)])`

Hosszú számsorokra használjuk az xrange iterátor függvényt a range függvény helyett, az xrange függvény nem állítja elő a teljes listát, gyorsabb és memória kímélőbb lehet.	

`for i in range(1,1000000):`
`  if i % 631 == 0 and i % 35 == 0:`
`  print i`
`  break`
	`for i in xrange(1, 1000000):`
	`  if i % 631 == 0 and i % 35 == 0:`
	`  print i`
	`  break`

Listák egyesítése szótárrá	

`l1 = ['Csabi', 'Tomi', 'Piri']`
`l2 = [2, 0, 3]`
`s = {}`
`for i in range(len(l1)):`
`  if l2[i]:`
`  s[l1[i]] = l2[i]`
	`l1 = ['Csabi', 'Tomi', 'Piri']`
	`l2 = [2, 0, 3]`
	`s = { l1[i]: l2[i] for i in range(len(l1))`
	`  if l2[i] }`

Ismétlődő elemek megszüntetése a listában	

`l =[1, 2, 1, 3, 2, 4, 3]`
`l1 = []`
`for i in l:`
`  if not i in l1:`
`  l1.append(i)`
	`l =[1, 2, 1, 3, 2, 4, 3]`
	`l1 = list(set(l))`

Általános megoldás	Pythonic megoldás
Prim számok	

prims = [1]
for i in range(2, 1000):
    prim = True
    for j in range(2, int(math.sqrt(i))
+1):
        if i % j == 0:
            prim = False
    if prim:
        prims.append(i)
	numbers = range(1000)
	for j in numbers[2:]:
	    numbers[j+j::j] = [0 for k in
	numbers[j+j::j]]
	prims = sorted(list(set(numbers) -
	set([0])))