Scripting speci

Magyarfalvi Gábor

2007

Mi az a script

- Nagyon magas szintu nyelven írt, többnyire rövid program
- Magas szintu nyelvek: Unix shellek, Tcl, Perl, Python, Ruby, Scheme, Rexx, JavaScript, VisualBasic
- Itt a főleg a Pythonról, kicsit a Perlről, és talán a bash-ról lesz szó

Mire jó egy script?

- Más programok összekapcsolása
- Szöveges adatok feldolgozása
- File manipuláció
- Alkalmi programok
- Sok scriptből nagy programrendszer is lehet
- Könnyen GUI is csapható hozzá
- Hordozható Unix, Windows és Mac között
- Interpretált (nincs fordítás és linkelés

Miért nem jó a Java vagy a C/C++?

- A script rövidebb és tömörebb
- Sokkal gyorsabb és kényelmesebb a programozó dolga
- Jobban esik

Miért?

- Nem kell változókat definiálni
- Rengeteg standard könyvtár

Tömörebb

Valós számok összegzése egy fileból, ahol tetszés szerinti elrendezésben számok vannak.

```
F=file(filename,'r')
list=[float(x) for x in F.read().split()]
sum=sum(list)
```

Miért csinálom a specit?

Világéletemben tartózkodni akartam a programozástól.

- A számítógép buta, az eszközzel többet kell foglalkozni, mint a céllal.
- Soha nincs vége, karban kell tartani a programot.

A csapból is a scriptnyelvek folynak.

Ma már mindent Python-nal csinálnék - szinte mindent lehet is.

Élvezet vele dolgozni.

Természettudományos és kémiai informatika

- Nagyszabású, merev, cél-orientált programok (Gaussian, Mathematica, . . .)
- Gyorsan változó, épp fejlesztendő programok

A Python mindkét feladatra optimális.

Történet

1989 karácsonya, Guido van Rossum

Monty Python

Előzmények: ABC, Modula-3

Cél:

- olvasható és tömör
- gyorsan írható és javítható
- magas szintű adatstruktúrák, OOP
- Unix/C közeli
- nagyon könnyen bővíthető, meglevő eszközökkel is
- széles körben elérhető
- hatékony, jól skálázódó

Lassabb, mint a statikus nyelvek, de van megoldás.

Implementáció

- Fordítás bytekódra, majd interpretálás
- Automatikus memóriamenedzselés
- Dinamikus adattípusok, nem kell deklarálni
- Minden igazi objektum
- Introspekció, megváltoztatható szerkezetek
- Csoportosítás behúzással

Dokumentáció, letölthető programok:
‘`http://www.python.org`’
‘`http://www.enthought.com`’, Win32 + rengeteg tudományos könyvtár

Hozzáférés

Linux – gyakorlatilag mindig.

‘`python`’ (‘`/usr/bin/python`’)

Indítás

Magában – interaktív shell.

```
$python
Python 1.5.2 (#1, Sep 30 2000) on linux-i386
Copyright 1991-1995 Stichting Mathematisch Centru
>>> 2**16
65536
>>> a = 'Berkenye'
>>> print a[1:5]
erke
>>>
```

Nem csak intelligens számológép.

Minden elképzelés, kifejezés kipróbálható.

Kilépés: EOF(UNIX: Control-D, Win: Control-Z)

A viselkedése hasonlít a Unix shellekre:

```
python program.py arg1 arg2
```

```
python <program.py
```

```
python -c 'print "hello"'
```

```
Py_Initialize();
PyRun_SimpleString("x,y = x+y, x-y ");
```

Közvetlen futtatható programok

Unix: `chmod +x program.py`

```
1:#!/usr/bin/python
2:...
```

Windows: asszociáció

Példa

pH.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys,math
ph = sys.argv[1]
print 'A pH:',ph
print 'A H-konc: ', math.exp(-float(ph))
```

```
chmod +x pH.py
```

```
python pH.py 4.2
```

Megjegyzések

- `#!/usr/bin/env`

  Az interpretert adja meg közvetlen futáshoz

- `import sys`

  Könyvtári modulok betöltése

- `float(ph)`

  String és szám típus

```
#!/usr/bin/python

import random

a = random.randint(1,10)
b = random.randint(1,10)
print 'Mennyi ',a,'+',b,'?'
c = raw_input()
print c
if int(c) == a+b :
print 'OK'
```

Számok

Egész (C *long*)	`666, 01232, 0x29a`
Tetszőleges méretű egész	`2L*33, 666666666666L`
Lebegőpontos (C *double*)	`6.0e-23, 2.82`
Komplex (2 double)	`1j, 2.1+1.1j`

Operátorok

`+ - * / ( )` – értelem szerint

`%` – maradék

`**` – hatvány, `2L**32`

`&,|,` – bináris and, or, xor

`«,»` – bináris biteltolás, `2«2`

Vegyes operandusok

Az eredmény a legnagyobb rangú lesz.

```
>>> 7/2
3
>>> 7.0/2
3.5
```

Konverziók

`int(x)`

`long(x)`

`float(x)`

`complex(real,[imag])`

Egyéb beépített függvények

`abs(x)` – abszolút érték (komplex - nagyság)

`hex(x)`

`oct(x)`

`pow(x,y)` – x^y

`round(x,[n])`

A többi függvény a *math* modulban.

Attribútumok

```
>>> a = complex(1,1)
>>> a.real, a.imag
(1.0, 1.0)
```

Stringek

Nem megváltoztatható karaktersorok

```
'python', "Python", "7 o' clock"
'"Állj!" - mondta.'

>>> print '1.sor \
... 2.sor'
1.sor 2.sor
>>> print '''1. sor
... 2. sor'''
1. sor
2. sor
```

Escape kódok

\n – új sor

\t – tabulátor

\',\" – ', "

\\ – \

\033, \x41 – bármi

Műveletek

```
>>> a = 'a''b''c'
>>> b = 'Python ' + '2.0'
>>> a*3
'abcabcabc'
>>> b[2]
't'
```

Szeletelés

```
>>> b[:6]
'Python'
>>> b[7:]
'2.0'
>>> b[-3:]
'2.0'
>>> b[:]
'Python 2.0'
```

```
+---+---+---+---+---+---+---+---+---+---+
| P | y | t | h | o | n |   | 2 | . | 0 |
+---+---+---+---+---+---+---+---+---+---+
0   1   2   3   4   5   6   7   8   9   10
-10 -9 -8  -7  -6  -5  -4  -3  -2  -1
```

```
>>> b[2]='t'
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> len(b)
10
```

Nyers stringek: `r'\n'`

String formázás:

```
>>> 'Python %2.1f' % 2.0
'Python 2.0'
```

```
%s, %c, %d, %i, %u, %e, %E, %f, %g, %G
```

Listák

Tetszőleges objektumok sorozatai

Üres lista: `[]`

Hasonló operációk: szeletelés, +, *, len()

Megváltoztatható, egymásba ágyazható

```
>>> l = [0, 1.41]
>>> m = 2 * l
>>> m
[0, 1.41, 0, 1.41]
>>> m[3] = ['csipke','bodza'] #szeletre is
>>> m
[0, 1.41, 0, ['csipke', 'bodza']]
>>> m[2][1]
>>> m[3][:]
['csipke', 'bodza']
```

`len(m), min(m), max(m), del m[x:y]`

`x in m`

Attribútumok:

```
>>> m.append('a')
>>> m
[0, 1.41, 0, ['csipke', 'bodza'], 'a']
>>> m.remove(1.41) # ua. del m[m.index(1.41)]
>>> m[2].sort()    #helyben
>>> m
[0, 0, ['bodza', 'csipke'], 'a']
```

`m.index('a')` – az első előfordulás

`m.count('a')` – előfordulások száma

`m.insert(hova,mit)`

`m.reverse()`

`m.pop([honnan])`

```
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
>>> range(2,8)
[2, 3, 4, 5, 6, 7]
>>> range(2,8,2)
[2, 4, 6]
```

Szeresek – Tuple-ok

Meg nem változtatható sorozatok

Üres: `()`

Egytagú: `(1,)`

Többtagú: `(1, 2, 3)` zárójel opció

```
x, y = y, x
```

Szótárak (dictionary)

Asszociatív tömb, üres: {}

```
>>> szin = {'kökény': 'kék', 'csipke': 'piros'}
>>> szin['csipke']
'piros'
>>> szin['bodza'] = 'fekete'   # bővít
>>> szin['kökény'] = 'fekete'  # felülír
>>> del szin['kökény']
```

Nem rendezett, gyors indexelés

Kulcsok csak nem változtathatóak lehetnek.

```
>>> szin.keys()
['csipke', 'bodza']
>>> szin.values()
['piros', 'fekete']
>>> szin.items()
[('csipke', 'piros'), ('bodza', 'fekete')]
>>> szin.get('berkenye','Passz')
'Passz'
```

Minden objektum, szerepelhet listán, szótárban.
(függvény, modul, osztály, lefordított kód, ...)

Értékadás, összehasonlítás

Deklarálni nem kell, csak inicializálni.

A hozzárendelés csak referencia!!!

`==, <=, >=, <, >, !=, is, is not`

2.0: `+=, -=, *=, /=, **=, ...`

```
>>> x = x + 1
Traceback (innermost last):
  File "<stdin>", line 1, in ?
NameError: x
>>> x = y = 2
>>> 1 < x < 3
1
>>> l = [1,2]
>>> m = l
>>> l[0] = 2
>>> m
[2, 2]
```

Utasítások

Általában 1 sor 1 utasítás.

A sor végére nem kell pontosvessző.

Ha nagyon kell, több utasítást pontosvessző választhat el.

```
>>> a = 1; b = 2       # 2 egy sorban
```

1 sornál hosszabb utasítások:

1. Maguktól értetődnek (folytatódó struktúrák)
2. Nagyon ritkán \

Megjegyzések: # (egész vagy részleges sorok)

```
>>> #1 két sorban
>>> c = {'kökény' : 'kék',  # köztes duma
...      'csipke' : 'piros'}
>>> print 2+\
... 2
4
```

Kifejezések

`abs(1.2+j)`: függvény

`[].append('pagony')`: objektum módszere

Kiíratás

```
>>> c   # interaktív
{'kökény': 'kék', 'csipke': 'piros'}
>>> print 'kökény','csipke' # szóközök
kökény csipke
>>> print 'kökény'+'csipke'
kökénycsipke
>>> print 'kökény'+'csipke', #uj sor nelkul
kökénycsipke
>>> print >>file, 'aaa' # 2.0-tol
```

```
c = b = a = None
```

```
del a
```

Kisbetű és nagybetű nem mindegy!

If szerkezetek

```
if 'posix' in _names:
    name = 'posix'
elif 'nt' in _names:
    name = 'nt'
elif 'mac' in _names:
    name = 'mac'
else:
    raise ImportError
```

Ciklusok

```
while 1:
    pass  #megszakításra vár

>>> for i in range(10):
...     print i,
...
0 1 2 3 4 5 6 7 8 9
```

```
x = y / 2
while x > 1:
    if y % x == 0:
        print x,'osztja.'
        break
    x = x-1
else:
    print y, 'prím.'

for x in range(2,y/2):
    if y % x == 0:
        print x,'osztja.'
        break
else:
    print y, 'prím.'
```

Ciklusoknál is van `else:`.

`break` kiugrik a legbelső ciklusból (else sem fut!).

`continue` újrakezdi a ciklust.

`pass` nem csinál semmit.

A `for` egy listán lépdel végig.

A ciklusváltozót bánthatjuk, a listát jobb hagyni:

```
>>> a = list('berkenye')
>>> for i in a[:]:  #string is jöhetne
...     if i<'k': a.remove(i) #de fix
...
>>> a
['r', 'k', 'n', 'y']
```

Az `xrange()` nem generálja előre a listát.

Lehet, hogy egy kis memóriát megtakarít.

Házi feladat

- Tesztértékelés

  Input:

  ```
  ./teszt.py n helyes_megoldas
  ```

  ```
  Pl.:
  ```

  ```
  ./teszt.py 20 bbaebceaacdcdbcaecbd
  ```

  A program olvassa be a választ, értékelje, és kérjen új vizsgázót

- Római - arab számok A program kér egy római számot 1-2000 között és arab számként kiírja.

```
jo = raw_input()
felsz = len(jo) #feladatok száma
stat = {} #eredmények
while 1:
    line = raw_input()
    if not line: break
    pont = 0
    for fel in range(felsz):
        a = stat.get((fel,line[fel]),0)
        stat[(fel,line[fel])] = a + 1
        if line[fel] == jo [fel]:
            pont = pont + 1
    print '%3u pont' % pont
for fel in range(felsz):
    print ("%-3u" % (fel+1)),
    for kod in ['a','b','c','d','e']:
        elof = stat.get((fel,kod),0)
        if kod == jo [fel] :
            print "|%1s| %2i " % (kod,elof),
        else:
            print " %1s  %2i " % (kod,elof),
    print
```

Függvények

```
def cmp2(one, other):
    "String összehasonlítás" #docstring
    one = string.lower(one)
    other = string.lower(other)
    return cmp(one,other) #beépített
```

A függvények objektumként kezelhetőek:

```
>>> a = ['este','Esti']
>>> b = cmp2
>>> a.sort(b)
>>> a
['este', 'Esti']
>>> a.sort()
>>> a
['Esti', 'este']
```

```
def fact(n):
    if n == 1: return 1
    else:
        return n*fact(n-1)

def g(x):
    def fact(n):
        if n == 1: return 1
        else:
            return n*fact(n-1)
    print fact(x)
```

A második verzió 2.1 óta megy csak. Miért?

A dolgokat három helyen kereste:

1. Helyben
2. A beágyazás szintjein
3. Globálisok közt (a pr. legfelsőbb szintje)
4. Beépítettek közt

A 2.1 verziótól fokozatosan lépdel kifele. (nested

namespace)

A hozzárendelés lokális. Hacsaknem `global a`.

Argumentumok

Objektum referenciát adunk tovább.

Alapértelmezett argumentum:

```
def exit(uzenet='Viszlát'):
    print uzenet
```

Tetszés szerinti argumentumlista:

```
def szerkkesz(self,valt,*args):
# args egy tuple, az értékekkel
```

Hívás kulcsszavakkal:

```
def exit(uzenet,ertek,ido):
    ...

exit(ertek=1,uzenet='vege',ido='most')
```

Keverhetőek, de:

1. A rendes argumentumok
2. A kulcsszavas argumentumok
3. Az alapértelmezettek
4. A maradék megy *arg és **arg-ba

File objektumok

```
input = open("b.dat",'r')
```

Olvasás ('r','a')

`file.read([x])` – x byte, vagy az egész olvasása

`file.readline()` – egy sor, sorvéggel együtt

`file.readlines()` – az összes sor egy listába

`file.write(string)` – string írása

`file.writelines(list)` – stringek listájának írása

`file.flush()` – pufferek ürítése

`file.close()` – bezárás (nem feltétlen kell)

Funkcionális eszközök

`lambda` – Névtelen függvény

```
>>> def kakukk():
...     return lambda x: x[:-2]
...
>>> visszh = kakukk()
>>> visszh('haho')
'ha'
```

`apply(func,args)` – Hívás

`map(func,list)` – Implicit átalakító ciklus

`filter(func,list)` – Implicit válogatás

`reduce(func,list)` – Implicit "összegző"

`zip(seq1,seq2)` – `map(None,seq1,seq2)`

2.0:

```
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
```

Modulok

A programszervezés legfelsőbb szintje:

- Python nyelvű file-ok (file.py)
- más nyelvű könyvtárak (C)
- Rendszermodulok

Használat: import

```
import os
print os.getcwd()
```

Előre is lehet hivatkozni, kivéve a modul legfelső szintje.

Gyakran kerül a modul végére ilyesmi:

```
#ha mi vagyunk a főprogram
if __name__ == '__main__': teszt()
```

Import menete:

1. megkeresi a modult
2. új modul obj. létrehozása
3. a modul utasításainak lefuttatása és az új objektumok létrehozása a modulon belül

Direkt importálás:

```
from os import getcwd
getcwd()
```

A kapott objektum lokális.

Többszöri importtal ugyanazt osztjuk meg.

`import sys; rendszer = sys; del sys`

2.0: `import sys as rendszer`

Menet közben is lehet frissíteni: `reload modul`

A modulok belül szótárak.

Speciális attribútumok:
`__dict__, __doc__, __file__, __name__`

```
dir([modul]), vars([modul])
```

Rendszermodulok

Rengeteg van: rendszerkönyvtárak, hálózat, gyakori programkonstrukciók, tömörítő, kép-, hangfeldolgozó könyvtárak, GUI, ...

sys

A python rendszer fontosabb részei:

- `sys.argv` – argumentumok listája
- `sys.exit()` – kilépés
- `sys.ps1, sys.ps2` – interaktív promptok
- `sys.stdin, .stdout, .stderr` – file objektumok
- `sys.modules` – a betöltött modulok

```
sys.modules['sys'].__dict__['exit']()
```

os

Az oprendszer szolgáltatásai:

- `os.system('parancs')` – futtatja
- `os.getcwd()` – jelenlegi directory
- `os.getpid()` – process id
- `os.listdir(path)` – ls
- `os.mkdir(path)`,`os.rmdir(path)`
- `os.remove(path)`
- `os.rename(mit,mire)`
- `os.popen('parancs')` – fileobjektum a parancshoz kötve
- `os.linesep` – sorvégkarakter(ek)
- `os.path.exists(path)` – létezik-e?
- `os.path.isfile(path)` – file-e?
- `os.path.splitdrive(path)`, `splitext()`
- `os.path.basename()`, `dirname()`
- `os.path.split(path)`

glob

Unix shell filenév expanzió

```
>>> from glob import glob
>>> glob('*.py')
['a.py', 'roma.py', 'Script.py', 'Script1.py']
```

shutil

```
shutil.copy(mit,hova)
```

math

Matematikai rutinok

```
log(), log10(), exp(), cos(), ...
```

```
math.pi, math.e
```

random

- `random.random()` – [0.0 ... 1.0)
- `random.choice(seq)` – véletlen elem
- `random.randrange(arg)` – choice(range(arg))
- `random.uniform(a,b)` – [a ... b)

string

Gyors, C műveletek.

2.0-tól jórészt elérhetőek a string típus módszereiként.

- `string.digits, .letters, .punctuation, ...`
- `string.upper(), .lower()`
- `string.swapcase(), .capitalize()`
- `string.find(string,mit)` – hol van benne (-1 nincs)
- `string.rfind(string,mit)` – jobbról kezdi
- `string.index(), .rindex` – hiba, ha nincs

- `string.count(string,mit)` – hányszor van meg
- `string.strip(string)` – whitespace eltávolítása
- `string.rstrip(string), .lstrip(string)` – ua.
- `string.center(string,hossz)` – kétoldalról space feltöltés
- `string.rjust(), .rjust()` – jobbról, balról
- `string.expandtabs(string,[tabhossz])` – tabból space
- `string.split(string,[szeparátor])` – szétszedi
- `string.join(lista,[szeparátor])` – összerakja

Hibakezelés

A futás közben jelentkező hibákat kezelhetjük.

Szokatlan, de nagyon hasznos vezérlési módszer.

```
def oszto(dict):
    "dict normalasa dict['egy']-gyel"
    ered={}
    try:          #figyeli a hibákat
        for i in dict.keys():
            ered[i]=dict[i]/dict['egy']
    except ZeroDivisionError, mess:
        print 'nullával osztás', mess
    except KeyError, key:
        print 'Nincs egységelem', key
    else:         #ha nem volt hiba
        return ered
```

Az `except` tartalmazhat hibalistát vagy állhat üresen is.

Üresen bármilyen hibát elcsíp. Veszélyes, mert azt is kezeli, amire nem számítunk.

A try szerkezetek egymásba ágyazhatóak.

Mi is generálhatunk hibát: `raise SystemExit`

Fontosabb hibák:

- AttributeError
- EOFError (file.readline nem kelti!)
- IOError
- IndexError
- KeyError
- KeyboardInterrupt

Definiálhatunk új hibákat is.

Alternatív szerkezet:

```
try:
    fontosfile.adatait.modositja()
finally:  # mindenképpen végrehajtásra kerül
    fontosfile.write(puffer)
    fontosfile.close()
```

Házi feladat

- Lottó

  Input:

  ```
  ./lotto.py
  ```

  A program 5 darab lottótippet generáljon egy file-ba.

- Izotóptömegek

  Az atomok izotóptömeg-eloszlása file-ban adott, formátuma:

  ```
  Atom Izotóp Gyakoriság Izotóp Gyakoriság .....
  H   1  0.98        2  0.0095     3  0.0005
  ```

  A program ezt a következo struktúrába olvassa be:

  ```
  osszatomeloszl={"C":{12:0.99,13:0.01},
                  "H":{1:0.98,2:0.0095,3:0.0005}
                  "O":{16:1.0}}
  ```