PYTHON

Avagy hosszú az út a BioPythonig

Miért a Python?

- Mert ez áll a legközelebb az ember által beszélt nyelvhez.
- Mert ez a leggyorsabb az ismert script nyelvek közül
- Mert rengeteg modul érhető el hozzá
- Mert a rövid egyszerű feladatokat is könnyü megoldani, de támogatja az objektum orientált gondolkodásmódot is.
- Mert a PyMOL-hoz így tudunk majd plug-in-t írni.
- Mai óra anyaga nagyrészt itt meg van részletesebben: http://docs.python.org/release/2.5.2/tut/tut.html

Hogyan használjuk a Pythont?

- Parancssorból
  - Például: python -c "print 'Hello World'"
- Mint parancs értelmező

```
~$ python
Python 2.5.2 (r252:60911, Jan 20 2010, 21:48:48)
[GCC 4.2.4 (Ubuntu 4.2.4-1ubuntu3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
>>>
```

- Python scriptek
  - Python utasítások előre megírt sorozata, melyeket egy szövegfile tárol

Script futtatás

- A python parancs értelmező stdin-jére irányítjuk a forrás filet
  - Például: echo ”print 'hello world!'”|python
  - Vagy: cat hello.py|python
- De van egy jobb módja is:
  - Kellemes konvenció: Ha egy szövegfile *#!* karakter sorozattal kezdödik, akkor az elsö sor további része a filehoz tartozó interpretert jelöli
    - Pl: *#!/bin/bash* vagy *#!/usr/bin/perl*
    - Vagy mi esetünkben: *#!/usr/bin/python*
      - A *~$whereis python* utasítással lehet kideríteni, hogy mit kell a *#!* után írni

Script futtatás

- Ezek után csak annyi a dolgunk, hogy:
  - Megírjuk a script file-t, elején az interpreter megjelölésével
    - Például legyen ez a hello.py:

```
#!/usr/bin/python
print ”hello world”
```

  - Futtatási jogot adunk a script file-nak
    ~$ chmod 777 hello.py
  - *Futtatjuk a scriptet*
    ~$ ./hello.py

A Hello World-ön túl

- Megjegyzések: #
- Aritmetika: + - * /
- Complex számok: 1+1j
- Stringek (füzérek) : ” ” jelek közt vagy ' ' közt
  - Több soros string: ”””” és ”””” között
  - Összefüzés: + jellel

Az első ”értelmes” program

```
#!/usr/bin/python

# Fibonacci sorozat:
# Az utolsó két elem összege a következö elem.

a, b = 0, 1 # Csoportos értékadás
while b < 1000:
    print b
    a, b = b, a+b
```

A blokkokat a behúzások definiálják!!!

Elágazások

```
#!/usr/bin/python

x = int(raw_input("Please enter an integer: "))
if x < 0:
      x = 0
      print 'Negative changed to zero'
elif x == 0:
    print 'Zero'

elif x == 1:
    print 'Single'

else:
    print 'More'
```

Tömbök

- Def: a = ['spam', 'eggs', 100, 1234]
- Elem: a[0]
- Intervallum: a[1:2] vagy a[1:-1]
- Lista hossza: len(a)
- A stringek speciális tömbök, így a fentiek a stringekre is alkalmazhatóak

FOR ciklus

```
#!/usr/bin/python

a = ['cat', 'window', 'defenestrate']
for x in a:
      print x

#VAGY

for i in range(len(a)):
      print i, a[i]

#VAGY


for i, v in enumerate(a):
    print i, v
```

Break és Continue

```
#!/usr/bin/python

for n in range(2, 10):
        for x in range(2, n):
                if n % x == 0:
                        print n, 'equals', x, '*', n/x
                        break
        else:
                print n, 'is a prime number'
############

a=2*3*5*7*11*11*13*17

d=2

while a>1:
    if not a%d==0:
            d=d+1
            continue
    print d, a
    a=a/d
```

Még a tömbökről

- T.append(x) – a T tömböt végére beszúrja x-et
- T.extend(L) – A T és L tömböt egymás után fűzi
- T.insert(i, x) – Az i-edik helyre beszúrja x-et
- T.pop() – Törli az utolsó elemet, és vissza adja értékét
- T.pop(i) – Törli az i-edik elemet, és vissza adja értékét
- T.index(x) – Megmondja hol fordul elösször elő x
  - Hasonló: T.count(x), T.remove(x)
- T.sort() - Rendezi a tömböt
- T.reverse() - Megcseréli a sorrendet

Szótárak

- Olyan tömbök, melyeket egészek helyett szavakkal indexelhetünk
- tel = {'jack': 4098, 'sape': 4139}
  tel['guido'] = 4127
  print tel['jack']
  print tel
- For ciklus ilyenkor így néz ki:

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.iteritems():
    print k, v
```

Függvények

```
#!/usr/bin/python

def fib(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result


print fib(100)
```

Alapértelmezett értékek

```
#!/usr/bin/python

def bogus(a,b,c=1,d=1):
        return a*b+c*d


# Lehetséges hívások:

bogus(1,2)

bogus(1,2,3)

bogus(1,2,3,4)

bogus(1,2,d=6)

bogus(d=6, a=1, b=2)

# Ha nem akarjuk elöre megadni az argumentumok számát:

def bogus(*args):
     for i in args:
             print i,
```

Modulok

- Pythonban a függvények többsége valamilyen modulban található!
- Ezek lényegében fügvény könyvtárak, melyeket tipikusan C-ben vagy Pythonban írtak.
- C-ben írt modulok függvényei gyorsak
- Pythonban mi is könnyen írhatunk modul-t
- Ami minket a legjobban izgat majd: BioPython

Olvasás

```
#!/usr/bin/python

# Olvasás a stdin-röl:
import sys

print sys.stdin.readlines()
print sys.argv

# Olvasás fileból:
fpin=open('filename')
print fpin.readline()
for i in fpin.readlines():
        print i
```

Példa: PDB atom koordináták

```
#!/usr/bin/python

fpin=open('10gs.pdb')
atoms=[]
for i in fpin.readlines():
	if i[0:6]=='ATOM   ':
		atoms.append([float(i[31:38]), float(i[39:46]), float(i[47:54])])

print atoms
```

Casting

- A Python igen rugalmasan kezeli a változókat:
  - A lokális változókat nem kell elöre definiálni, csupán csak elkezdjük használni
    - Azért célszerü inicializálni!
  - Azt sem szükséges megadni mi lesz a változóban
    - Ami épp benne van olyan típusu lesz a változó
    - Változó név prefixekkel célszerü magunknak jelölni, hogy egy adott változóba milyen típusu értékeket szánunk
- Viszont idönként szükség van rá, hogy változó típusok között convertáljunk. Ilyenkor a tipus neve lesz a konvertáló függvény neve is
  - Leggyakoribb típusok:
    - int – egész
    - float – lebegö pontos érték (valós vagy complex)
    - str – karakter sorozat (string)

Hatáskör

- Függvényekben eddig mi lokális változókat használtunk
  - Ugyan az a változó név minden függvényben más változóra mutat.
- Mi van ha szeretnénk egy változót minden függvényböl egyszerre látni?
  - A global kulcsszóval jelezzük a python számára, hogy az adott változó névvel egy közösen használt értékre szeretnénk hivazkozni (példa mindjárt)
  - A függvényeken kívül használt változók autómatikusan globálisak
  - BEST PRACTICE: kerüljük a globális változókat
- Nincsenek statikus változók: minden függvény hívásnál nullázódik a függvény lokális változóinak értéke

Példa

```
#!/usr/bin/python

def bogus():
        global g
        l='local inside'
        g='global inside'
        return l, g

l='local outside'
g='global outside'

print l,g
print bogus()
print l, g
```