



Bevezetés a Python programozási nyelvbe

Szathmáry László
Debreceni Egyetem
Informatikai Kar

2012-2013, 1. félév



7. Gyakorlat

- modulok / csomagok
- random számok
- haladó rendezés

Modulok

Amint a programunk egyre hosszabb lesz, felmerül az igény, hogy jó lenne szétvágni több részre.

Egyrészt könnyebb lenne a karbantartás, átláthatóbb lenne a projekt.

Másrészt egy-egy hasznos függvényt szeretnénk esetleg több programban is felhasználni anélkül, hogy a függvényt be kellene másolni minden egyes programba.

Feladat:

Írjunk két programot:

1.írassuk ki a 100-nál kisebb prímszámokat (`ex1.py`)

2.írassuk ki a 200-nál kisebb prímszámok összegét (`ex2.py`)

Link: <http://www.inf.unideb.hu/~szathml/pmwiki/index.php?n=Py.20121110a>

ex1.py

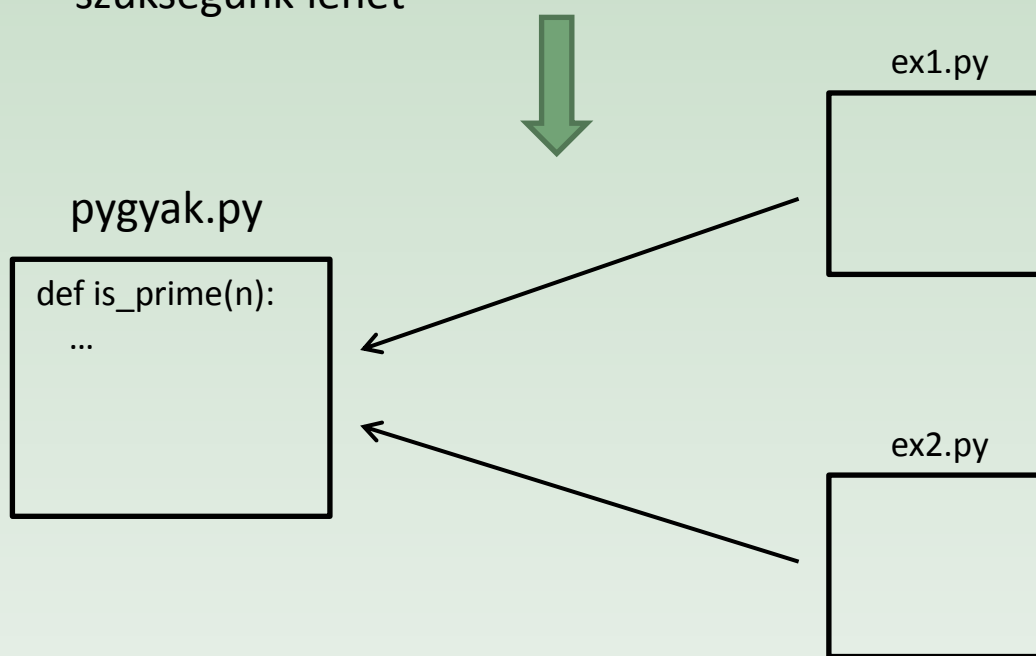
```
def is_prime(n):  
    ...
```

ex2.py

```
def is_prime(n):  
    ...
```

Miután készen vagyunk, nézzük meg, hogy mi a közös a két szkriptben. A közös rész az `is_prime()` függvény.

1. ugyanazt a függvényt ismétljük több helyen
2. az `is_prime()` olyan műveletet hajt végre, amire esetleg még máshol is szükségünk lehet



A közös függvényt tegyük ki egy modulba (pl. `pygyak.py`), majd mindkét szkriptben hivatkozzunk erre a modulra (importálás).

pygyak.py

```
1  def is_prime(n):
2      """
3      Decide whether a number is prime or not.
4      """
5      if n < 2:
6          return False
7      if n == 2:
8          return True
9      if n % 2 == 0:
10         return False
11
12     i = 3
13     maxi = n**0.5 + 1
14     while i <= maxi:
15         if n % i == 0:
16             return False
17         i += 2
18
19     return True
20
21
22  def hello():
23      return "Hello, World!"
```

Használata (ex3.py):

```
3  import pygyak
4
5
6  def main():
7      print pygyak.is_prime(5)
8
9      print pygyak.hello()
10
```



import pygyak # .py nélkül

Variációk

```
import pygyak
```

Ez a pygyak-ban definiált függvényeket nem teszi be az aktuális szimbólumtáblázatba. Csupán a modul neve, a pygyak kerül be a sz.-t.-ba. Ezért a modulon belüli függvényekre a modul nevén keresztül tudunk hivatkozni:

```
pygyak.is_prime(...)
```

```
import pygyak as pgy
```

Ha a modul neve hosszú és/vagy sokszor hivatkozunk rá, akkor átnevezhetjük úgy, hogy teszünk rá egy alias-t.

Innentől: `pgy.is_prime(...)`

Variációk (folyt.)

```
from pygyak import is_prime
```

Jelentése: a pygyak modulból hozzuk be az is_prime függvény nevét a sz.-t.-ba.

!!! Ez a modul nevét NEM hozza be a szimbólumtáblázatba !!!

`Pl.print pygyak.hello()` # hiba, a „pygyak” szimbólum ismeretlen

Megoldás:

1. `from pygyak import is_prime, hello`



2. `from pygyak import is_prime`
`import pygyak`



3. `from pygyak import *`

Nem ajánlott, átláthatatlan lesz hogy mit honnan importáltunk.



Modulok tesztelése

Előtte:

```
1 def is_prime(n):
2     """
3     Decide whether a number
4     is prime or not.
5     """
6     if n < 2:
7         return False
8     if n == 2:
9         return True
10    if n % 2 == 0:
11        return False
12
13    i = 3
14    maxi = n**0.5 + 1
15    while i <= maxi:
16        if n % i == 0:
17            return False
18        i += 2
19
20    return True
21
22
23 def hello():
24     """
25     Classic example.
26     """
27     return "Hello, World!"
```

```
1 #!/usr/bin/env python
2
3 """
4 pygyak modul
5 =====
6
7 A Python gyakorlathoz
8 készített modul.
9 A gyakran használt fv.-eket
10 itt gyűjtjük össze.
11 """
12
13 def is_prime(n):
14     """
15     Decide whether a number
16     is prime or not.
17     """
18     if n < 2:
19         return False
20     if n == 2:
21         return True
22     if n % 2 == 0:
23         return False
24
25     i = 3
26     maxi = n**0.5 + 1
27     while i <= maxi:
28         if n % i == 0:
29             return False
30         i += 2
31
32     return True
33
34
35 def hello():
36     """
37     Classic example.
38     """
39     return "Hello, World!"
40
41 #####
42
43 if __name__ == "__main__":
44     for n in range(2, 20):
45         if is_prime(n):
46             print '{n} primszam'.format(n=n)
```

Utána:

docstring

teszt



Modulok tesztelése (folyt.)

```
40
41 #####
42
43 if __name__ == "__main__":
44     for n in range(2, 20):
45         if is_prime(n):
46             print '{n} primszam'.format(n=n)
```

tetszőleges
tesztek

A feltétel (lásd 43. sor) csak akkor igaz, ha a modult közvetlenül futtatjuk a parancssorból (vagyis `./pygyak.py`). Ekkor lefut a teszt.

Ha modulként importáljuk, akkor a feltétel hamis, a teszt nem fut le.

Vagyis: ha megírunk egy modult, akkor azt nem csak modulként lehet használni, hanem önálló szkriptként is.

Feladat: Módosítsuk úgy a `pygyak.py` modult, hogy önálló indításkor kérjen be a felhasználótól egy egész számot, majd írja ki, hogy az prím-e vagy sem. Ezután futtassuk le az `ex1.py` szkriptet ismét, amiben importáljuk ezt a módosított `pygyak.py` modult. Mit tapasztalunk?

Modulok tesztelése (folyt.)

pygyak.py

```
1  #!/usr/bin/env python
2
3
4  def hello():
5      return "Hello, World!"
6
7  #####
8
9  if __name__ == "__main__":
10     print __name__
```

ex1.py

```
1  #!/usr/bin/env python
2
3  import pygyak
4
5
6  def main():
7      print pygyak.hello()
8      print pygyak.__name__
9
10     #####
11
12     if __name__ == "__main__":
13         main()
```



```
$ ./pygyak.py
__main__
```

```
$ ./ex1.py
Hello, World!
pygyak
```



Modulok tesztelése (folyt.)

```
40
41 #####
42
43 if __name__ == "__main__":
44     for n in range(2, 20):
45         if is_prime(n):
46             print '{n} primszam'.format(n=n)
```

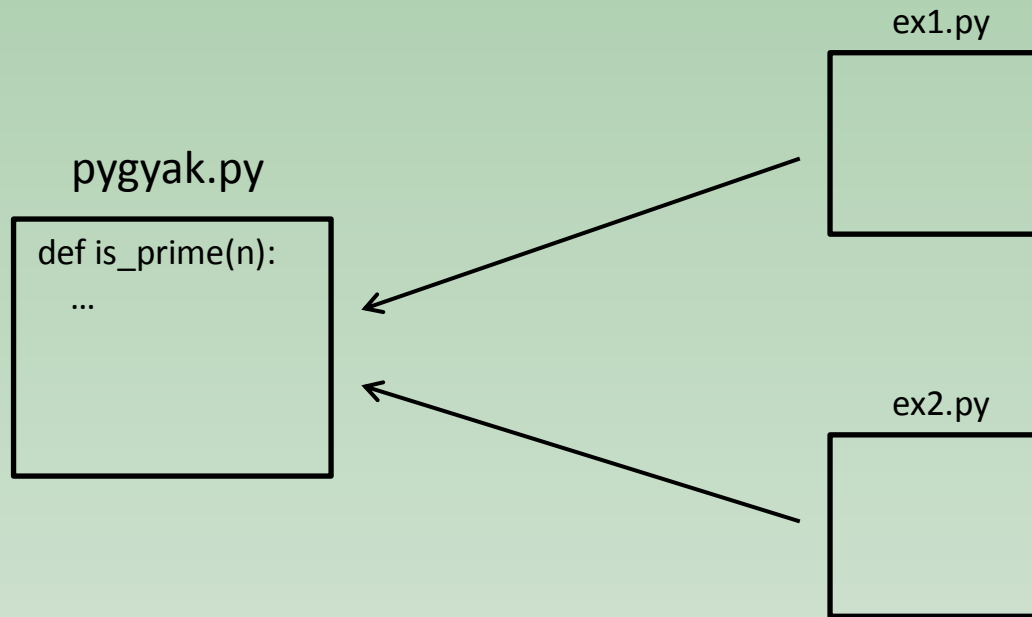


További előny:

Ha a modulunk végére teszünk teszteket, akkor azzal a modult használó munkáját is segítjük, mivel konkrét példákat adunk a modul használatára.

„Alice: - Írtam egy király modult, elküldtem emailben. Megkaptad?
Bob: - Igen, de... Most akkor ezeket a függvényeket hogy kell használni?
Alice: - Menj le a forrás aljára, ott találsz pár példát.
Bob: - Ja, OK, látom már. Ehh, zsír. Köszí.”

Modulok



Modulok használatának további előnye

Ha egy függvényt módosítunk, akkor azt csak egyetlen helyen kell megtenni.

Feladat: A `pygyak.py` modulban cseréljük le az `is_prime()` függvény implementációját a jelenleginél jóval hatékonyabb Miller-Rabin tesztre. Az MR teszt forráskódját a gyakorlati anyagok között találják.

Utána futtassuk le az `ex1.py` és `ex2.py` szkripteket. Mit tapasztalunk?

Modul importálásának a menete

```
import spam
```

Ezt szeretnénk importálni. Hogyan találja ezt meg az interpreter? Honnan fogja importálni?

1) Az interpreter megnézi, hogy van-e ilyen nevű beépített modul.

Beépített modul: bele van fordítva az interpreterbe.

```
>>> import sys
>>> sys.builtin_module_names
```

2) Ha nem találta meg, akkor egy `spam.py` nevű fájlt fog keresni könyvtárak egy listájában, mely lista a `sys.path`-ban található.

```
>>> import sys
>>> sys.path
```

Ez a `sys.path` lista a következőképpen inicializálódik:

- a szkriptet tartalmazó könyvtár
- `PYTHONPATH` környezeti változó. Ez a `PATH` -hoz hasonló: ugyanúgy épül fel és könyvtárak listáját tartalmazza.
- az adott telepítésre jellemző alapértelmezett könyvtárak

Modul importálása (folyt.)

```
>>> import sys  
>>> sys.path
```

A `sys.path` egy közönséges lista => az inicializáció után a saját szkriptünkből is módosíthatjuk.

!!! A futtatott szkriptet tartalmazó könyvtár a `sys.path`-ban *előre* kerül, a standard library mappái *elé*. Ez szokatlan hibákat okozhat, ha a könyvtár olyan fájlokat is tartalmaz, melyeknek a neve megegyezik valamelyik standard library-ben lévő modul nevével. !!!

„Lefordított” Python fájlok

Látható, hogy a `pygyak.py` mellett megjelent egy `pygyak.pyc` nevű file is. Ezt az interpreter automatikusan állította elő, ezzel nekünk nem nagyon kell foglalkozni.

Amikor egy modult importálunk, akkor az interpreter megpróbálja azt lefordítani, ezek lesznek a `.pyc` fájlok. Ez egy platformfüggetlen bináris kód.

Amikor importálunk egy modult és van belőle `.pyc` verzió, akkor az interpreter azt fogja betölteni. Az eredeti `.py` file módosítási ideje bele van fordítva a `.pyc` fájlba, vagyis ha a `.py` módosul, akkor az interpreter figyelmen kívül hagyja a `.pyc` fájlt (a `.py` fájlt fogja beolvasni és újrafordítja).

!!! Ha az interpreter a `.pyc` fájlt tölti be, attól még a programunk nem fog gyorsabban futni. **!!!** Csupán a modul betöltési ideje lesz gyorsabb.

Ha egy programot közvetlenül a parancssorból futtatunk, azt soha nem fogja az interpreter lefordítani. Csak az importált modulokat kísérli meg lefordítani.

Olyan is lehet, hogy csak a `spam.pyc` file van jelen a forrás nélkül. Ekkor az interpreter ezt fogja használni. Vagyis egy programot így is lehet terjeszteni, ha a forrást nem akarjuk kiadni. Ezt aránylag nehéz visszafejteni (reverse engineering).

Mi van egy modulban?

```
>>> import pygyak
>>> dir(pygyak)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'hello', 'is_prime']
>>>
>>> pygyak.__doc__
'\npygyak modul\n=====\n\nA Python gyakorlathoz \nkeszitett modul.\nA gyakran has
>>> print pygyak.__doc__

pygyak modul
=====

A Python gyakorlathoz
keszitett modul.
A gyakran hasznalt fv.-eket
itt gyujtjuk össze.
```

a modul docstring -je

Random számok

<http://docs.python.org/library/random.html>

```
1  >>> import random
2  >>>
3  >>> random.random()
4  0.12214691572646652
5  >>> random.randint(1, 10)
6  9
7  >>> li = [3, 8, 2, 8, 4, 2, 1, 9]
8  >>> li
9  [3, 8, 2, 8, 4, 2, 1, 9]
18 >>> random.shuffle(li)
19 >>> li
20 [9, 8, 1, 4, 8, 2, 3, 2]
21 >>>
22 >>> random.choice(li)
23 8
```

[0.0, 1.0)

lower <= N <= upper

összekeveri az elemeket
helyben
(random permutáció)

random elem kiválasztása
a listából

Feladat: a `shuffle` eljárás *helyben* módosítja a listát. Írjunk egy olyan `my_shuffle` nevű fv.-t, amely visszaadja az összekevert listát, lehetővé téve pl. a következőt: `n = my_shuffle(li)[-1]`.

Feladat

Bullshit Generátor

Link: <http://www.inf.unideb.hu/~szathml/pmwiki/index.php?n=Py.20121030a>

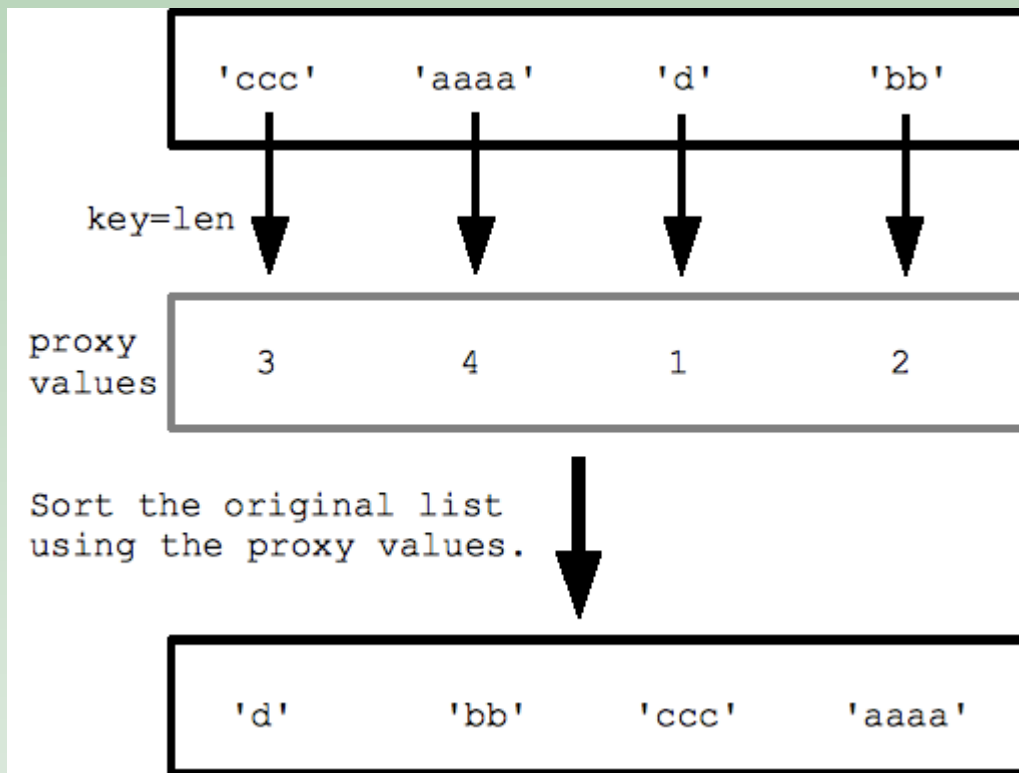


Haladó rendezés

Összetettebb rendezések esetén a `sorted()` függvénynek megadható egy „key” paraméter, melynek értékül egy függvényt lehet megadni. Ez a függvény a rendezés előtt az elemeket áttanszformálja. A függvény vesz egy értéket s egy új értéket ad vissza. A rendezés a függvény által visszaadott érték alapján történik.

```
1 >>> words = ['ccc', 'aaaa', 'd', 'bb']
2 >>> sorted(words)
3 ['aaaa', 'bb', 'ccc', 'd']
4 >>>
5 >>> sorted(words, key=len)
6 ['d', 'bb', 'ccc', 'aaaa']
```

A „key” értéke most a beépített „len” függvény. A szavakat a szavak hossza szerint szeretnénk rendezni. A rendezés során a `len` meghívódik a lista minden egyes elemére, s a rendezés a `len` által visszaadott érték szerint történik.



a „len” által visszaadott értékekből készít egy „árnyéklistát” s ezt rendezi:
[1, 2, 3, 4]

majd visszahelyettesít:

1 -> 'd'
2 -> 'bb'
3 -> 'ccc'
4 -> 'aaaa'

```
1 >>> words = ['Cc', 'BB', 'aa', 'zz']
2 >>> sorted(words)
3 ['BB', 'Cc', 'aa', 'zz']
4 >>>
5 >>> sorted(words, key=str.lower)
6 ['aa', 'BB', 'Cc', 'zz']
```

rendezzük úgy, hogy a kis- és nagybetűket *nem* különböztetjük meg

```
1 >>> words = ['xc', 'zb', 'yd', 'wa']
2 >>> sorted(words)
3 ['wa', 'xc', 'yd', 'zb']
4 >>>
5 >>> def my_func(s):
6 ...     return s[-1]
7 ...
8 >>> sorted(words, key=my_func)
9 ['wa', 'zb', 'xc', 'yd']
```

rendezzük az elemeket az utolsó karakter szerint

saját függvény is megadható

Feladat

Oldjunk meg néhány haladó rendezéssel kapcsolatos feladatot.

Link: <http://www.inf.unideb.hu/~szathml/pmwiki/index.php?n=Py.20121006e>