Tartalomjegyzék

Bevezetés a Python nyelvbe

Az el?adáshoz els?számú olvasmány a Python tutorial.

A Python egy olyan általános körben használható magas szint? programozási nyelv, aminek az egyik alapelve az olvasható kód írása egy nagyon tiszta szintaxis használatával. 1991-ben alkotta meg Guido Van Rossum.

További jellemz?k

- objektum orientált, procedurális, funkcionális
- sok beépített modul a fejlesztés megkönnyítésére
- dinamikus típus kezelés
- automatikus memóriakezelés
- többféle megvalósítás (CPython, Jython, IronPython, PyPy, Python for S60)
- open-source a f?bb platformokra
- sokkal tömörebb sok más nyelvnél

Nevét a Monthy Python ihlette, nem az állat.

Filozófiája megkapható a 'this' modul betöltésével:

```
import this
```

Futtatása

Egyel?re csak interaktívan: egyszer?en a python parancs terminálból indítva.

```
$ python
Python 2.7.5+ (default, Sep 19 2013, 13:48:49)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Sokkal jobb környezetet biztosít az ipython:

```
$ ipython
Python 2.7.5+ (default, Feb 27 2014, 19:37:08)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Külön ablakot ad az IDLE (Interactive Development Environment):

```
idle
```

Kódolás stílusa

Stílus (code style) a PEP 8 alapján

- használj mindig 4 space-t minden egyes szinthez, de a folytatósort kezd még beljebb,
- a nagyobb kódrészeket tagold üres sorokkal (függvény, osztály, nagyobb kód blokk)
- használj space-t a vesz? után és a legmagasabb szinten lév? operátorok körül
- használj docstring-et és ahol lehet a megjegyzés a saját sorára vonatkozzon, vagy azonos mértékben behúzva arra a blokkódra
- ahol lehet használj ASCII karakterkódolást
- 79 karakternél ne legyen hosszabb egy sor
- CamelCase elnevezési konvenciót kövesse az osztályok neve és lower_case_with_underscores a függvények és változók nevei

Érdemes megnézni a Google python code style ajánlását is.

Objektumok

Objektumok: a nyelv alapelemei, ezekkel dolgozunk. Minden objektumnak típusa van. Négy felbonthatatlan skaláris alapobjektum:

- egész `int`: 2354, -12
- lebeg?pontos szám `float`: 1.0, -23.567, 2.3E4

- logikai `bool`: `True`, `False`
- semmi: `None`

M?veletek, kifejezések

Az objektumok m?veletekkel összekapcsolva kifejezéseket adnak, melyek kiértékelve valamilyen típusú objektumot adnak. Az egész és a lebeg?pontos m?veletek:

- `a + b` összeadás
- `a - b` kivonás
- `a * b` szorzás
- `a / b` osztás (Python 2.7-ben int/int = int, Python 3-tól float)
- `a // b` egész osztás
- `a % b` maradékképzés
- `a ** b` hatványozás
- `a == b`, `a < b`, `a > b`, `a <= b`, `a >= b`, `a != b`, `a <> b`

Logikai m?veletek bool-típusúak közt:

- `a and b`, ?és?
- `a or b`, ?megenged? vagy?
- `not a`, ?nem?

Azonosítók

Az adatokat többszöri felhasználásra *azonosítóval* (*névvel*) láthatjuk el.

- a név bet?vel vagy aláhúzással kezd?dhet: [_a-zA-Z]
- a név további karakterei az el?bbieken felül számok is lehetnek: [_a-zA-Z0-9]
- elméletileg bármilyen hosszú lehet a név
- név nem lehet foglalt szó
- nagybet?-kisbet? érzékeny, tehát a val1 név nem azonos a Val1 névvel

A foglalt szavak:

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

De ne használjuk a Python beépített neveinek, függvényeinek, kivételeinek neveit sem. Ezek megkaphatók a `dir(__builtins__)` paranccsal:

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', ......
```

Objektumhivatkozások

Az értékadás (=) esetén valójában **objektumhivatkozás** történik, azaz az egyenl?ség bal oldalán álló névhez egy hivatkozás kapcsolódik, mely az egyenl?ség jobb oldalán álló objektumra mutat. Ez érthet?vé teszi a következ? kódot:

```
>>> a = 1
>>> b = a
>>> a = 2
>>> a
2
>>> b
1
```

A Python **dinamikusan típusos** nyelv, azaz az objektum, amire egy objektumhivatkozás mutat, lecserélhet? egy más típusú objektumra (nem kell a változók típusát deklarálni).

```
>>> a = 1
>>> type(a)
<type 'int'>
>>> a = "b"
>>> type(a)
<type 'str'>
```

Lehetséges a többszörös értékadás:

```
>>> a, b = 1, 2
>>> a, b = b, a
>>> print a, b
2 1
```

Karakterláncok (str)

A karakterláncok megadása: `"http://wiki.math.bme.hu..."`http://wiki.math.bme.hu, `'...'` vagy `"http://wiki.math.bme.hu"`http://wiki.math.bme.hu`"http://wiki.math.bme.hu..."`http:/ módon történhet:

```
>>> 'alma'
'alma'
>>> """itt 'ez' meg "az" van"""
'itt \'ez\' meg "az" van'
>>> print _
itt 'ez' meg "az" van
>>> type("alma")
<type 'str'>

>>> c = 'aa\nbb'
>>> c
'aa\nbb'
>>> print c
aa
bb
```

Véd?kódok (eszkép karakterek, escape characters) a rep-karaterrel: \ (folytatás új sorban), \\ (\), \' ('),

\"http://wiki.math.bme.hu ("http://wiki.math.bme.hu), \n (új sor), \t (tab). (Ha a karakterlánc elé **r** bet?t írunk, a véd?kódok nem érvényesek.)

M?veletek karakterláncokkal, **indexelés** és **szeletelés**:

```
lánc[sorszám]
lánc[kezdet:vég]
lánc[kezdet:vég:lépés]
```

továbbá az **+** (összef?zés) és a ***** (többszörözés) m?veletek:

```
>>> a = "ho"
>>> b = "rgasz"
>>> 3*a + b
'hohohorgasz'

>>> c = _       # _ az el?z? eredmény
>>> c
'hohohorgasz'
>>> c[0]
'h'
>>> c[0:2]
'ho'
>>> c[:2]
'ho'
>>> c[-1]
'z'
>>> c[:2]+c[6:]
'horgasz'
>>> c[1:7:2]
'ooo'
>>> c[1:6:2]
'ooo'
>>> c[-1::-1]
'zsagrohohoh'
>>> c[-3:4:-1]
'agro'
```

Az indexekre kétféleképp gondolhatunk: 1. a második index már nincs (ennek pl. az az értelme, hogy egy intervallum végét, és a következ? elejét azonos érték jelzi, nem kell 1-et hozzáadni), 2. az indexeket az elemek közé képzeljük, vagyis az elemek határait indexeljük:

```
 +---+---+---+---+---+---+---+
 | h | o | r | g | a | s | z |
 +---+---+---+---+---+---+---+
 0   1   2   3   4   5   6   7
-5  -6  -5  -4  -3  -2  -1
```

A karakterláncok nem változtatható (immutable) objektumok, vagyis a m?veletek, tagfüggvények alkalmazása után új karakterlánc keletkezik:

```
>>> a = "aaaa"
>>> a[1] = b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Egyszer? programok

Szekvencia, bemenet, kimenet

```
>>> nev = raw_input('Mi a neved? ')
Mi a neved? Laci
>>> print 'Szia %s!' % nev
Szia Laci!
```

Újabb szintaxis a kimenetre:

```
>>> nev = raw_input('Mi a neved?\n')
Mi a neved?
Laura
>>> ora = int(raw_input('Kb. mikor találkozzunk?\n'))
Kb. mikor találkozzunk?
3
>>> print 'Szia %s! Találkozzunk %d körül!' % (nev, ora)
Szia Laura! Találkozzunk 3 körül!
>>> print 'Szia {0}! Találkozzunk {1} körül!'.format(nev, ora)
Szia Laura! Találkozzunk 3 körül!
```

Elágazás, ciklus

if

```
>>> x = int(raw_input("Adj meg egy egész számot: "))
Adj meg egy egész számot: 42
>>> if x < 0:
        print "ez negatív"
    elif x == 0:
        print "ez nulla"
    elif x == 1:
        print "ez egy"
    else:
        print "ez sok"

ez sok
```

Az elif-ek száma tetsz?leges, és ezzel elkerülhet? a sok behúzás.

for, break, else, continue

Döntsük el 9-ig minden egészr?l, hogy prím vagy összetett szám!

```
>>> for n in range(2, 10):
        for x in range(2, n):
            if n % x == 0:
                print n, '=', x, '*', n/x
                break
```

```
        else:
            # ha a ciklusban nem talált osztót (nem volt break)
            print n, 'prím'

2 prím
3 prím
4 = 2 * 2
5 prím
6 = 2 * 3
7 prím
8 = 2 * 4
9 = 3 * 3
```

Az **else** a **for**-ral is használható! Szerencsétlen a névhasználat (jobb lenne pl. finally), de a szerkezet praktikus lehet. Az else ágra a ciklus utolsó végrehajtása után kerül a vezérlés (így akkor is, ha a ciklus egyszer sem fut le). Elkerüli viszont az else ágat a vezérlés, ha a break utasítással hagyjuk el a ciklust.

Írjuk ki az 50 alatti páros számokat, de a 3-mal oszthatók helyett *-ot tegyünk!

```
>>> for n in range(2, 50, 2):
        if n % 3 == 0:
            print "*",
            continue
        print n,

2 4 * 8 10 * 14 16 * 20 22 * 26 28 * 32 34 * 38 40 * 44 46 *
```

while, break, else

Írjuk ki az 1000 alatti Fibonacci-számokat:

```
>>> n = 1000
>>> a, b = 0, 1
>>> while a < n:
        print a,  # a vessz? miatt egy sorba kerülnek
        a, b = b, a + b

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

A while-nak is lehet else-ága és használható a break.

A pythonban nincs hátul tesztel? ciklusutasítás, a következ?vel helyettesíthet?:

```
while True:
    utasítások
    if kilépési_feltétel:
        break
    (utasítások)
```

Függvény

Írjunk függvényt az n! kiszámítására!

Iteratív megoldás:

```
def faktorialis_i(n):
    """Feltesszük, hogy n >= 0. Visszaadja n! értékét!"""
    fakt = 1
    while n > 1:
        fakt = fakt * n
        n -= 1
    return fakt
```

Rekurzív megoldás:

```
def faktorialis_r(n):
    """Feltesszük, hogy n >= 0. Visszaadja n! értékét!"""
    if n <= 1:
        return 1
    else:
        return n * faktorialis_r(n - 1)
```

Hatókör (scope), névtér

Minden új függvény fölépíti saját névterét, egy szótár ('név':érték párok, formailag kapcsos zárójelek közt) formájában. Íme a globális és egy lokális névtér:

```
>>> x = 2
>>> print globals()
{'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', 'x': 2, '__doc__': Non
>>> def fn():
...     y = 3
...     print locals()
...
>>> fn()
{'y': 3}
```

Ha egy név lokálisan nem lett létrehozva, a tartalmazó névterekben keresi egyre kijjebb haladva (itt az x nem lett lokálisan létrehozva, de elérhet?):

```
>>> def fn():
...     y = 3
...     print "x, y:", x, y
...     print locals()
...
>>> fn()
x, y: 2 3
{'y': 3}
```

```
>>> def f(x):
def g():
            x = 42
    print 'g: x =', x
def h():
            y = x
    print 'h: x, y =', x, y
        x += 1
print 'f1: x =', x
 ()     g
 ()     h
print 'f2: x =', x
```

```
return h

>>> i = 2
>>> f(i)
f1: x = 3
g: x = 42
h: x, y = 3 3
f2: x = 3
<function h at 0x2c7ce60>
>>> a = f(5)
f1: x = 6
g: x = 42
h: x, y = 6 6
f2: x = 6
>>> a()
h: x, y = 6 6
```

A névtér törl?dik a függvény lefutása után, magasabb szinten az alacsonyabb szint? névterek nem érhet?k el:

```
>>> def fn():
...     z = 1
...
>>> print z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
```

***-os kérdés**: Mi történik az alábbi kód hatására? magyarázzuk meg, mi történik:

```
>>> def fn():
...     y = 3
...     print "x, y:", x, y
...     x = 5
...     print locals()
...
>>> fn()
```

És mi történik, ha a 3. és 4. sort fölcseréljük?

A **függvény lezárása** a Pythonnak azt a képességét jelenti, hogy a nem globális névtérben definiált függvény **emlékszik** a definiálás pillanatában érvényes bennfoglaló névterekre:

```
>>> def kulso(x):
...     def belso():
...         print x
...     return belso
...
>>> f1 = kulso(3)
>>> f2 = kulso(5)
>>>
>>> f1()
3
>>> f2()
5
```