Tartalomjegyzék

Python

Bevezetés

A Pythont ismerjük a Sage-b?l. Különbségek:

- ^ helyett ??, / osztást, // egészosztást jelöl.
- [a..b] helyett range(a,b), vagy xrange(a,b)
- A megszokott, beépített matematikai függvények hiánya (find_root, plot, is_prime).
- Nincsenek szimbolikus változók

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 (imperatív, 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

Python kód futtatása

A kód futtatható interpreter konzolon belül és küls? fájlban tárolva.

- Nyiss egy új file-t pl. a *gedit*-ben, mentsd el "http://wiki.math.bme.hukerdez.py"http://wiki.math.bme.hu néven egy könyvtáradba!
- Írd bele a következ? python kódot (ne használj ékezeteket):

```
s = input("Mondj egy szamot: ")
print "Ennel eggyel kisebbet mondtal: ", str(s + 1)
```

- Mentsd el, és futtasd a scriptedet! (*python kerdez.py*)
- Most kicsit kiegészítjük a scriptet, hogy tartalmazhasson ékezetes bet?ket, és hogy kényelmesebben futtatható legyen (a *python* parancs begépelése nélkül is):

```
#!/usr/bin/python
#coding=UTF-8
s = input("Mondj egy számot: ")
print "Ennél eggyel kisebbet mondtál: ", str(s + 1)
```

- Mentsd el, és adj rá futtatási jogot csak magadnak (`chmod +x kerdez.py`)!
- Futtasd így: `kerdez.py` vagy így: `./kerdez.py`
- A kód második sora lehet ez is:

```
# -*- coding: utf-8 -*-
```

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 az operátorok mellett
- használj docstring-et és ahol lehet a megjegyzés a saját sorára vonatkozzon
- ahol lehet használj ASCII karakterkódolást
- 79 karakternél ne legyen hosszabb egy sor (elvben 80)
- 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.

Docstring

A hivatkozás nélküli string elemet szokás használni megjegyzések írására és dokumentálásra.

```
"""This is a class of example.

TODO: needs implementation.
"""
```

Els? sort nagybet?vel kezdjük és pontal zárjuk. Egy összefoglaló mondat legyen. Majd egy üres sort hagyva részletesen leírhatunk minden funkciót amit az osztály vagy függvény tartalmaz.

Hasznos linkek:

- hogyan érdemes
- unittest-elés docstring segítségével
- Sphinx

A Python eljárásközpontú (procedurális) programozásának elemei

Adattípusok

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

Azonosítók

- 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
- nagybetü kisbetü érzékeny, tehát a val1 név nem azonos a Val1 névvel

Másképp, a Backus?Naur formában leírva:

```
identifier ::=  (letter|"_") (letter | digit | "_")*
letter     ::=  lowercase | uppercase
lowercase  ::=  "a"..."z"
uppercase  ::=  "A"..."Z"
digit      ::=  "0"..."9"
```

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', ......
```

Egészszer? adattípusok: egész, logikai

int (egész) és hosszú egész (long)

```
>>> a = 12
>>> b = 0b011101
>>> c = 0o3701
>>> d = 0xff0c
>>> a, b, c, d
(12, 29, 1985, 65292)
>>> e = 22222222222222222222222222222222222222
>>> e
22222222222222222222222222222222222222L
>>> type(a), type(b), type(e)
(<type 'int'>, <type 'int'>, <type 'long'>)
```

M?veletek egészekkel: +, -, *, / (float eredményt ad), //, % (maradék), **, abs, pow, round, | (OR butenként),

^ (XOR bitenként), & (AND bitenként), <<, >> (eltolás bitenként), ~ (bitenkénti NOT)

Logikai (bool)

Logikai értékek: False (0), True (nem 0)

Logikai m?veletek: and, or, not

```
>>> True and False
False
>>> 1 and 0
0
>>> 3 and 0
0
>>> 3 or 1
3
>>> 1 or 3
1
>>> not 67
False
>>> not 0
True
```

Beépített lebeg?pontos típusok (float, complex)

M?veletek eredmény NaN és infinity is lehet!

Lebeg?pontos szám megadása:

```
>>> 2.3, -1.2e3, -1.2e-2
(2.2999999999999998, -1200.0, -0.012)
```

Matematikai függvények:

```
import math
```

után. Ld. [1]

Komplex szám imaginárius része után **j** bet?, de * nincs! Komplex **jellemz?i (attribute)** a real és az imag, **tagfüggvénye (method)** conjugate():

```
>>> z = 3.1 + 2.2j
>>> z
(3.1000000000000001+2.2000000000000002j)
>>> z.real
3.1000000000000001
>>> z.imag
2.2000000000000002
>>> z.conjugate()
(3.1000000000000001-2.2000000000000002j)
```

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://wiki.math.bme.hu"http://wiki.math.bm módon történhet:

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

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

>>> d = r'aa\nbb'  # az r bet? után minden karakter magát jelenti
>>> d
'aa\\nbb'
>>> print d
aa\nbb
```

Véd?kódok (escape characters): \ (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: Szeletelés:

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

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

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

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

Tagfüggvények: részletesen lásd [2]. Folytatva az el?z? példát:

```
>>> c.capitalize()
'Hohohorgasz'
>>> c.upper()
'HOHOHORGASZ'
>>> c.index('o')
1
>>> c.count('o')
```

```
3
```

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
```

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'>


=== Gy?jteményes adattípusok ===

==== Sorozatszer? típusok: str, list, tuple ====

Tuladonságaik: bejárhatók, in, len(), []

'''Sorozat (tuple)''' változtathatatlan (immutable), mint a karakterlánc, m?veletei:
szeletelés [], összef?zés (+), szorzás (*), összehasonlítás, tagsági viszony (in, not in)

'''lista''' - változtatható (mutable)
http://docs.python.org/release/2.5.2/lib/typesseq-mutable.html

halmaztípusok: halmaz
http://docs.python.org/release/2.5.2/lib/types-set.html
```

2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1.

- **None** - a semmi programbeli megvalósulása
- **numerikus**
  - egész
  - lebeg?pontos
  - (complex)
  - long
  - boolean
- **sorozatok** - elemeik egész számmal indexelhet?ek
  - módosíthatatlanok
    - string
    - tuple

```
>>> t = ()
>>> t
()
>>> t = tuple()
>>> t
()
>>> t = (1,2,'')
>>> t
(1, 2, '')
>>> t = 3,4,5,''
>>> t
(3, 4, 5, '')
>>> t[2]
5
>>> t.count(5)
1
>>> t.index(5)
2
```

  - módosíthatóak
    - lista

```
>>> l = []
>>> l
[]
>>> l = list()
>>> l
[]
```

```
>>> l = [1, "p", ['k'], 2.5]
>>> l
[1, 'p', ['k'], 2.5]
>>> l += [1]
>>> l
[1, 'p', ['k'], 2.5, 1]
>>> l.pop()
1
>>> l
[1, 'p', ['k'], 2.5]
>>> l.reverse()
>>> l
[2.5, ['k'], 'p', 1]
>>> l.count(1)
1
>>> l.insert(1,1)
>>> l.count(1)
2
>>> l
[2.5, 1, ['k'], 'p', 1]
>>> l.remove(1)
>>> l
[2.5, ['k'], 'p', 1]
```

- **halmazok**
  - ♦ halmaz (set)

```
>>> s = set([5,6,7,8,6])
>>> s
set([8, 5, 6, 7])
>>> s2 = set()
>>> s2
set([])
>>> s2 = s.copy()
>>> s2
set([8, 5, 6, 7])
>>> s2.add(6)
>>> s2.add(11)
>>> s2
set([8, 11, 5, 6, 7])
>>> s2.difference(s)
set([11])
>>> s2.discard(11)
>>> s2.difference(s)
set([])
>>> s2.intersection(s)
set([8, 5, 6, 7])
>>> s2.update([5,4,3])
>>> s2
set([3, 4, 5, 6, 7, 8])
>>> s2.discard(2)
>>> s2
set([3, 4, 5, 6, 7, 8])
>>> s2.remove(3)
>>> s2
set([4, 5, 6, 7, 8])
>>> s2.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
>>> s
set([8, 5, 6, 7])
>>> s.union(set([56,5]))
```

```
set([56, 5, 6, 7, 8])
>>> s
set([8, 5, 6, 7])
>>> s2
set([4, 5, 6, 7, 8])
>>> s.clear()
>>> s
set([])
```

- **térképezés**
  - ♦ szótár (dict)

```
>>> d = dict()
>>> d
{}
>>> d = {}
>>> d
{}
>>> d = {'fn':'Marci', 'ln':'Pala', 2:89}
>>> d
{'ln': 'Pala', 2: 89, 'fn': 'Marci'}
>>> d[2]
89
>>> d['ln']
'Pala'
>>> d2 = d.copy()
>>> d2
{'ln': 'Pala', 2: 89, 'fn': 'Marci'}
>>> d2[2] = 8
>>> d2
{'ln': 'Pala', 2: 8, 'fn': 'Marci'}
>>> d
{'ln': 'Pala', 2: 89, 'fn': 'Marci'}
>>> d.update([("t",3),('b',6)])
>>> d
{'ln': 'Pala', 2: 89, 'b': 6, 't': 3, 'fn': 'Marci'}
>>> d2
{'ln': 'Pala', 2: 8, 'fn': 'Marci'}
>>> d.values()
['Pala', 89, 6, 3, 'Marci']
>>> d.pop('b')
6
>>> d
{'ln': 'Pala', 2: 89, 't': 3, 'fn': 'Marci'}
>>> d.keys()
['ln', 2, 't', 'fn']
>>> d.items()
[('ln', 'Pala'), (2, 89), ('t', 3), ('fn', 'Marci')]
>>> d.iteritems()
<dictionary-itemiterator object at 0x00B73C90>
>>> d.get('teki')
>>> d.get('teki', t)
(3, 4, 5, '')
>>> d
{'ln': 'Pala', 2: 89, 't': 3, 'fn': 'Marci'}
>>> d['teki']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'teki'
>>> len(d)
4
>>> d.clear()
>>> d
```

```
{}
>>> len(d)
0
>>> d2
{'ln': 'Pala', 2: 8, 'fn': 'Marci'}
```

Fontos tudni, hogy a numerikus és a módosíthatatlan sorozatok úgymond "http://wiki.math.bme.huváltoztathatatlanok"http://wiki.math.bme.hu ami annyit tesz, hogy ha egy elemre több hivatkozásunk van, akkor ha az egyiket megváltoztatom akkor az új értékkel keletkezik egy új elem a memóriában és a hivatkozás már erre fog mutatni. (Ez a felhasználó számára láthatatlan!)

Ezek az alapvet?en használt elemek, ezen felül van meg sok egyéb amelyek közül a legfontosabb az osztályok amiket kés?bb fogunk venni. Ha többet akartok tudni a python adatmodelljér?l akkor itt találtok leírást róla.

változók névadásánál a következ? szabályok vonatkoznak:

Operátorok

Operátorok precedenciája

Az operátorok között, mint a matematikában itt is, van precedenciai sorrend:

Operator	Description
lambda	Lambda expression
if ? else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, identity test, including membership test
`|`	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
`+, -`	Addition and subtraction
*, //, /, %	Multiplication, division, remainder
`+x, -x, ~x`	Positive, negative, bitwise not
**	Exponentiation
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key:datum...}, `expressions...`	Binding or tuple display, list display, dictionary display, string conversion

A sage, python a kifejezéseket balról jobbra értékeli ki, kivéve az értékadásnál, amikor el?bb a jobb oldalt értékeli ki, majd a ball oldalt.

pl.:

logikai kifejezés elemeit ha mar felesleges nem értékeli ki

Vezérlési elemek

A vezérlési elemeket az el?z? félévben a SAGE-el kapcsolatban már átnéztük, ezeket kell tudni. Info1 kapcsolódó diái

Összefoglalva:

Elágazás

- **if** (elif, else)

Ciklusok

- **while** (else)
- **for** (else)
- **break**, **continue**

Hasznos segít?eszközök

Ciklusok során a következ? nyelvi elemeket találhatjuk hasznosnak:

- `range(x,z,y), xrange(x,y,z)`

  - ezekkel számlistákat tudunk el?állítani. els? paraméter a mett?l, a második a meddig és a harmadik a lépésköz. Csak a meddig paraméter a kötelez?, a kezdet alapból 0 és a lépésköz pedig 1. Nagy listák esetén az xrange optimálisabb kódot eredményez
- `enumerate(x)`

  - ennek segítségével az elemekkel megkapjuk a listában betöltött helyüket is

```
>>> for i,v in enumerate(l):
...   print 'index:',i,'and value:',v
...
index: 0 and value: 2.5
index: 1 and value: ['k']
index: 2 and value: p
index: 3 and value: 1
```

- `d.iteritems()`

  - segítségével a szótár egy elemének kulcsával és értékével tér vissza

```
>>> for k, v in d2.iteritems():
...   print 'key:',k,'and value:',v
...
key: ln and value: Pala
key: 2 and value: 8
key: fn and value: Marci
```

- `zip(list, ...)`

  - segítségével a paraméterként kapott sorozatok elemeit csoportosítja elhelyezkedésük szerint.

```
>>> import copy
>>> l2 = copy(l)
>>> l2 = copy.copy(l)
```

```
>>> l2
[2.5, ['k'], 'p', 1]
>>> l
[2.5, ['k'], 'p', 1]
>>> l2[1] = 0
>>> l
[2.5, ['k'], 'p', 1]
>>> l2
[2.5, 0, 'p', 1]
>>> zip(l, l2)
[(2.5, 2.5), (['k'], 0), ('p', 'p'), (1, 1)]
```

- `reversed(list)`

  - segítségével a paraméterként kapott felsorolás elemeit megfordító objektummá alakítja

```
>>> l
[8, 7, 6, 5]
>>> reversed(l)
<listreverseiterator object at 0xb722628c>
>>> l
[8, 7, 6, 5]
```

- `sorted(sorozat)`

  - segítségével a paraméterként kapott felsorolás elemeit rendezett listában visszaadja

```
>>> l = [8,7,6,5]
>>> sorted(l)
[5, 6, 7, 8]
>>> l
[8, 7, 6, 5]
```

- `len(sorozat)`

  - segítségével a sorozat hosszát kaphatjuk meg

```
>>> len(l)
4
>>> len(s)
0
>>> len(s2)
5
```