<-- vissza

Tartalomjegyzék

Python-ról általában

A Python egy olyan általános körben használható magas szint? programozási nyelv, aminek az egyik alap elve 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. A tavalyi el?adást mint ismétlés ajánlom átnézni. (Tananyag része!)

Python szintaktika és az értelmezése

- változók definiálása és egyben deklarálása az **=** operátorral típus definiálása nélkül
- minden változó egy objektumként jelenik meg a háttérben aminek a típusát a **type(obj)** függvénnyel kérhetjük el.
- azonos kódblokkokat azonos behúzással jelöljük
- kódblokk kezdetét a **:** jelzi

- megjegyzést a **#** karakterrel tudsz beírni, ami azt jelenti, hogy a sorban utána lév? karaktereket már nem veszi figyelembe a fordító (sok helyen láthatsz olyat, hogy hivatkozás nélküli string-kel csinálják)
- lista másolása a `MyLista[:]` kifelyezéssel tehetjük meg. Fontos mivel több referencia használatával a lista módosításával esetleg olyan eredményt kaphatunk amire nem számítunk.
- egyszerre több elem inicializálása `a=b=c=1`
- a sorozat elemei kiszedhet?ek `a, b = [4, 5]`. Fontos hogy az összes elemre kell változót írnunk!
- az el?bbi struktúrát átültethetjük több változó több változóval való értékadására
- használható a `5 <= summa < 10` kifejezés is, ami olyan mintha kiírnánk a `summa >= 5 and summa < 10`

Code style PEP 8 alapján

- használj mindig 4 space-t minden egyes szinthez
- 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
- 80 karakternél ne legyen hosszabb egy sor
- CamelCase elnevezési konvenciót kövesse az osztályok neve és lower_case_with_underscores a függvények nevei

Google python code style

Docstring

Ahogy már említettük, a hivatkozás nélküli string elemet szokás használni megjegyzések írására és dokumentálásra.

```
"http://wiki.math.bme.hu"http://wiki.math.bme.hu"http://wiki.math.bme.huThis is a class of example.
```

TODO: needs implementation. "http://wiki.math.bme.hu"http://wiki.math.bme.hu"http://wiki.math.bme.hu

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

Adattípusok

- **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:

- a név betüvel vagy aláhúzással kezd?dhet: [_a-zA-Z] - ez egy reguláris kifejezés ami megegyezik a leírt feltétellel
- a név további karakterei az el?bbin felül lehet szám is: [_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 leírva:

identifier ::= (letter|"http://wiki.math.bme.hu_"http://wiki.math.bme.hu) (letter | digit | "http://wiki.math.bme.hu_"http://wiki.math.bme.hu)*

letter ::= lowercase | uppercase

lowercase ::= "http://wiki.math.bme.hua"http://wiki.math.bme.hu..."http://wiki.math.bme.huz"http://wiki.math.bme.hu

uppercase ::= "http://wiki.math.bme.huA"http://wiki.math.bme.hu..."http://wiki.math.bme.huZ"http://wiki.math.bme.hu

digit ::= "http://wiki.math.bme.hu0"http://wiki.math.bme.hu..."http://wiki.math.bme.hu9"http://wiki.math.bme.hu

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

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
```

Megjegyzések

- 2011-es év el?adás anyaga, érdemes átnézni, hátha letisztáz pár kérdést