Tartalomjegyzék

Bevezetés a Python nyelvbe

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) programozása

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, 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 (int, long), logikai (bool)

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 butenként), & (AND butenként), <<, >> (eltolás butenként), ~ (butenkénti NOT)

Logikai (bool) típus:

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)

A m?veletek eredménye NaN (not a number) 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: 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[: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)

```
>>> t = ()
>>> t
()
>>> t = 1, 5, 5, 5, 3   # itt elhagyható a zárójel
>>> t
(1, 5, 5, 5, 3)
>>> t[1]
5
>>> t.index(3)
4
>>> t.count(5)
3
```

lista - változtatható (mutable) [3]

```
>>> l[1:4:2] = [7, 8]
>>> l
[1, 7, 3, 8, 5]
>>> l.remove(7); l
[1, 3, 8, 5]
>>> l.reverse(); l
[5, 8, 3, 1]
>>> l.append(99); l
[5, 8, 3, 1, 99]
>>> l += [55, 66]
>>> l
[5, 8, 3, 1, 99, 55, 66]
>>> l.pop()
66
>>> l
[5, 8, 3, 1, 99, 55]
```

Halmaztípusok: halmaz (set)

Lásd [4]

```
>>> 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([])
```

Megfeleltetési típusok: szótár (dict)

Lásd [5]

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

Listaértelmezések (list comprehension)

Rövid listák létrehozásának egyszer? módja. Általános alakja:

```
[expression for expr in sequence1
            if condition1
            for expr2 in sequence2
            if condition2
            ...
            for exprN in sequenceN
            if conditionN]
```

Kis programrészek helyettesíthet?k vele. Például soroljuk fel a szök?éveket 1890 és 1915 között.

```
szoko = []
for ev in range(1890, 1922):
    if (ev%4 == 0 and ev%100 != 0) or (ev%400 == 0):
        szoko.append(ev)
print szoko
[1892, 1896, 1904, 1908, 1912, 1916, 1920]
```

Lépésenként egyre összetettebb listaértelmezéssel állítsuk el? ugyanezt:

```
>>> szoko = [ev for ev in range(1890, 1915)]
>>> szoko    # ez még az összes év
[1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1
>>> szoko = [ev for ev in range(1890, 1915) if ev%4 == 0]
>>> szoko    # ez a 4-gyel osztható évek listája
[1892, 1896, 1900, 1904, 1908, 1912]
>>> szoko = [ev for ev in range(1890, 1915)
               if (ev%4 == 0 and ev%100 != 0) or ev%400 == 0]
>>> szoko
[1892, 1896, 1904, 1908, 1912]
```

Egy hasonló feladat: *Hányféleképp lehet nyolc királyn?t föltenni egy sakktáblára, hogy semelyik kett? ne üsse egymást?*

Az ilyen bástyaelhelyezések száma 8! = 40320. A királyn?k azonban átlósan is üthetik egymást. Az i-edik és j-edik sorban lév? királyn? pontosan akkor üti egymást, ha abs(i - j) != abs(x[i] - x[j]). A permutációk listázásához töltsük be az itertools csomag permutations függvényét, mely a for ciklus minden ciklusában ad egy övetkez? permutációt, amíg a végére nem ér. Így a megoldás:

```
from itertools import permutations
```

```
for x in permutations(range(8)):
    if all([i == j or abs(i - j) != abs(x[i] - x[j])
                for i in range(8) for j in range(8)]):
        print x
```

Gy?jtemények másolása (sekély és mély másolás)

Az értékadás (=) csak objektumhivatkozással jár, nem történik adatmásolás. Ennek következtében egy `y = x` parancs után, ahol x gy?jteményes adattípus, az y ugyanarra az objektumra fog mutatni. Így ha megváltoztatjuk x-et vagy y-t, változik a másik is. Az objektumhivatkozások megtörténhetnek egy szinttel mélyebben is, a (`z = x[:]` kód esetén az x elemeire mutató hivatkozások másolódnak, de ekkor sem jön létre a teljes objektumról másolat. Ezt hívjuk **sekély másolásnak (shallow copy)**.

```
>>> x = [1, 2, ["a","b"]]
>>> y = x
>>> z = x[:]
>>> y[0] = 0
>>> x, y, z
([0, 2, ['a', 'b']], [0, 2, ['a', 'b']], [1, 2, ['a', 'b']])
>>> z[0] = 5
>>> x, y, z
([0, 2, ['a', 'b']], [0, 2, ['a', 'b']], [5, 2, ['a', 'b']])
>>> x[2][0] = "X"
>>> x, y, z
([0, 2, ['X', 'b']], [0, 2, ['X', 'b']], [5, 2, ['X', 'b']])
```

Mély másolás (deep copy), amikor valóban új példány keletkezik az objektumból:

```
import copy
w = copy.deepcopy(x)
```

M?veletek, operátorok

Logikai értéket visszaadó m?veletek

Azonossági m?velet

Az alábbi példában a és b még azonos, mert ugyanarra az objektumra hivatkoznak, de a c és d már **nem azonosak**, bár egyenl?k:

```
>>> a = 2
>>> b = 2
>>> a is b
True
>>> c = [1, 2, 3]
>>> d = [1, 2, 3]
>>> c is d
False
>>> c == d
True
```

Összehasonlító m?veletek

Nem csak a számok, de karakterláncok, sorozatok, listák... is összehasonlíthatók (==, !=, <, >, <=, >=):

```
>>> a = 23
>>> b = 56
```

```
>>> a == b
False
>>> a != b
True
>>> a <= b
True
>>> c = "abc"
>>> d = "abcd"
>>> c == d[:3]
True
>>> c < d
True
>>> (2, 1, 3) <= (2, 1, 4)
True
```

Tagsági m?veletek

in, not in:

```
>>> l = [1, 2, "cica"]
>>> 2 in l
True
>>> "macska" not in l
True
>>> "c" in l[2]
True
>>> "ci" in l[2]
True
>>> "ci" in l
False
```

Logikai m?veletek

and, or, not (ld. fönt)

Operátorok precedenciája

Az operátorok között, mint a matematikában itt is, van precedencia 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 bal oldalt. Pl. a logikai kifejezés elemeit ha mar felesleges nem értékeli ki.

Vezérlési szerkezetek

- Elágazás
  - if (elif, else)
- Ciklusok
  - while (else)
  - for (else)
  - break, continue

- Lényeges, hogy a while és for ciklusokhoz itt hozzárendelhet? else ág. Ezek használatát mutatja be az alábbi két példa:

```
while condition:
    handle_true()
else:
    # condition is false now, handle and go on with the rest of the program
    handle_false()

for value in values:
    if value == 5:
        print "Found it!"
        break
else:
    print "Nowhere to be found. :-("
```

Függvények

- Python ban is van lehet?ség függvények definiálására és hívására.
- Függvényeknek itt is lehetnek paraméterei
- Ha rögzítjük a paraméterszámot, akkor ennek megfelel?en kell hívnunk a függvényt.
- Lehetséges el?re beállított paraméterekkel való hívás is. Ebben az esetben, az egyértelm?ség miatt a definiáláskor hátul adunk meg minen el?re is definiált paramétert.
- Példa

```
def a_d_test(fv, vert = 1):
  return fv * vert

a_d_test(5,2)
10
a_d_test(5)
5
```

- Pararméterek nevének különákülön való megadásával is átadhatjuk azok értékeit híváskor.

```
a_d_test(vert=2, fv=2)
```

- Speciálisan megadhatunk tetsz?leges paraméterszámú függvényeket.

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

as_many('trali', 'fari')
trali fari
```

Láthatóságról

- A kód itt is blokkokba rendez?dik és egy változó befelé látszik, kifelé nem.
- A bels? deklaráció "http://wiki.math.bme.huleárnyékol"http://wiki.math.bme.hu.
- Ez a küd tehát hibás, mert a b=a utasítás blokkján belül az a=6 utasítással létrehozzuk az a lokális változót. Emiatt leárnyékoljuk a küls? a változót, és a b=a utasításnál így hibát kapunk.

```
a=7
def sz3():
  b=a
  a=6
  return b

Traceback (click to the left of this block for traceback)
...
UnboundLocalError: local variable 'a' referenced before assignment
```

Kivételkezelés

- Lehet?ségünk van kivételek, hibák kezelésére.
- **try** és **except** blokkok használatával kezelhetjük a hibát adható részeit a kódnak.
- **A try blokkon belül megpróbáljuk végrehajtani az ott felsorolt utasításokat.**
- **Ha ez nem sikerül, akkor a felmerül? hibát az except blokkal "http://wiki.math.bme.hukapjuk el"http://wiki.math.bme.hu.**
- Példa, alább magyarázattal:

```
while True:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops!  That was no valid number.  Try again..."
```

- A while True egy végtelen ciklus, mellyel break utasítással léphetünk ki. A break utasítás csak akkor fut le, ha el?tte a felhasználótól hibátlanul bekértünk egy számot. Ha ez nem sikerült, akkor a try blokkból kilépünk a break utasíts el?tt és ugrunk az except részre. Mivel mindez egy végtelen while ciklusban van, ezért ezt addig filytatjuk, amíg helyes bemenetet nem kapunk.
- **Tehát megpróbáljuk végrehajtani a try blokkon belüli utasításokat, de amint hibát észlelünk, kilépünk a try blokkból.**
- A hibát az except blokkal kezeljük.
- Az except után megadhatjuk a hiba típusát, így a különböz? hibákat különböz?féleképp kezelhetjük:

```
try:
  z = x/y
except ZeroDivisionError, e:
  z = e # representation: "<exceptions.ZeroDivisionError instance at 0x817426c>"
```

```
print z
```

Bemenet, kimenet

Lsd. gyakorlaton

A Python objektumorientált programozása

- Ez a wiki szintén jól használható: http://en.wikibooks.org/wiki/Python_Programming/Classes

Osztályok, objektumok

- Az osztályok egységbe zárnak és a elrejtik a felesleges információt.
- Egy lehetséges néz?pont: a kód egy részét (mind tárolókat, mind a függvényeket nézve) összefogjuk, és a kifelé felesleges m?veleteket és változókat elrejtjük.
- Egy objektum egy osztály konkrét megvalósítása. Egy osztályban definiáljuk az objektum jellemz?it.
- Egy osztály metódusokból és változókból áll.
- Egy másik néz?pont: Az osztályon belül változókkal hozunk létre adattárolókat. Metódusokkal ezeket az adattárolókat érjük el kívülr?l.
- Példa osztály definiálásra pythonban:

```
class Osztaly:
    "Ez az uj csodafegyver"
    pass
```

A feni osztály még csak egy üres osztály. Ennek az egyszer? osztálynak egy objektum megvalósítása:

```
>>> o=Osztaly()
```

- Másszóval az o objektum az Osztaly osztály egy **példánya**.

A következ? kódrészlet metódus definiálására példa:

```
class Osztaly:
...   def make_sandwich(self):
...       self.sandwich=6
>>> x=Osztaly()
>>> x.make_sandwich()
```

- Ebben a példában az Osztaly osztályban definiáljuk a make_sandwitch() metódust. Az x objektum példányoztatása után az x objektumhoz tartozó make_sandwich() metódust hívjuk meg az utolsó sorban.
- Így kap értelmet a 'self' kifejezés. A metódusoknál az osztálydefinícióban jeleznünk kell, hogy az adott objektumon dolgozunk, és annak változóival hajtunk végre m?veletet.
- Emiatt minden metódus els? paramétere a 'self', ami mindig az osztály egy példányára (megvalósított objektum) hivatkozik.
- A self.sandwich emiatt az adott objektum változója lesz.
- Ha tehát meghívjuk a make_sandwich() metódust, akkor az az adott objektumon belül létrehozza a sandwich változót kezdeti 6 értékkel.

Láthatóság

- Említettük, hogy az osztályok egységbe zárnak és elrejtik a felesleges információt. Az "http://wiki.math.bme.huelrejtés"http://wiki.math.bme.hu azt jelenti, hogy valamilyen módon korlátoznunk kell az osztály változóira és metódusaira (továbbiakban elemek) vonatkozó hozzáférést.
- Az osztály egy eleme lehet: public, protected, private:
  - public: publikus elem hozzáférhet? az objektumon kívülr?l is.
  - private: privát elem csak az osztályon (objektumon) belül érhet? el.
  - protected: alosztályok (lsd. kés?bb) és az eredeti osztály számára elérhet? elem.
- Pythonban **minden változó és metódus publikus**
- A fenti szabály ellenére érdemes bizonyos dolgokat privát módon kezelni a kódban (annak ellenére, hogy lehet?ségünk lenne ?ket kívülr?l elérni). Emiatt a változók és metódusok elnevezésében a következ? konvenciót használjuk:
  - _ha_muszaj_elerem
  - __soha_nem_irom_felul__
- Változók:
  - osztály szint?: ha megváltoztatjuk az értékét az osztályon belül, akkor minden egyes objektumban megváltozik az értéke.
  - objektum szint?: adott objektumhoz tartozó változó
- Példa változókra:

```
class A:
...   u=8
...   def make_v(self):
...       self.v=6
>>> x=A()
>>> y=A()
>>> y.make_v()
```

- A fenti példában az x és y objektumok az A osztály példányai. Mindkett?ben megjelenik az osztály szint? u változó. Az y objektum make_v() metódusánk meghívása után (csak!) az y objektumban megjelenik a v változó.

Konstruktor és destruktor

- Azt az adott osztályhoz tartozó szubrutint, melynek feladata az osztály egy konkrét objektumának megvalósítása, konstruktornak hívjuk.
- Ehhez hasonló az objektum törlésekor hívódó destruktor.
- Pythonban külön nincs lehet?ség ezek felüldefiniálására. A Python szokásos értelemben nem használ konstruktorokat és destruktorokat.
- Azonban létezik olyan speciális metódus, mely az objektum létrehozása után közvetlenül lefut, ez az __init__():
- Az inithez hasonló a __del__().
- Példa az __init__()-re:

```
>>> class Test:
...     def __init__(self):
...
...         print "Mostmar elhiszed?"
...
>>> x=Test()
Mostmar elhiszed?
```

- A Test osztály megvalósítása az x objektum. Az x objektum létrejöttekor az __init__() azonnal lefut,

emiatt jelenik meg rögtön a kimeneten a "http://wiki.math.bme.huMostmar elhiszed?"http://wiki.math.bme.hu felirat.

Speciális metódusok pythonban

- az __init__() és __del__() metódusok mellett további speciális metódusok is elemei a nyelvnek.
- Ilyen például a __repr__() és az __str__(), melyek az adott osztály (objektum) leírását adják.
- Alapelv, hogy az __str__() legyen egyszer?, míg a __repr__() legyen egyértelm?.
- Példa:

```
>>> class A:
...   x=5
...
>>> a=A()
>>> a
<__main__.A instance at 0x7f1563927b90>
>>> class B:
...   x=5
...   def __repr__(self):
...     return "ez egy okos osztaly"
...
>>> b=B()
>>> b
ez egy okos osztaly
```

- A fenti példákon túl még több speciális metódus létezik. Lényegesek azon metódusok, melyekkel objektumokhoz tartozó operátorokat definiálhatunk.
- Mindezekr?l részletesen a gyakorlaton esett szó.
- Egy példa az el?adás anyagához:

```
class Word(str):
    '''Class for words, defining comparison based on word length.'''
    def __new__(cls, word):
        return str.__new__(cls, word)
    def __gt__(self, other):
        return len(self) > len(other)
    def __lt__(self, other):
        return len(self) < len(other)
    def __ge__(self, other):
        return len(self) >= len(other)
    def __le__(self, other):
        return len(self) <= len(other)
```

- A fenti példával stringeket hasonlíthatunk össze hosszuk alapján.

Örökl?dés, leszármaztatás

- Örökl?dés (leszármaztatás) esetén egy osztályból származtatunk le egy másik osztályt. Az ?sosztály leszármazottja (alosztály) örökli az ?sosztály tulajdonságait.
- Itt kap értelmet a "http://wiki.math.bme.huprivate"http://wiki.math.bme.hu és "http://wiki.math.bme.hupublic"http://wiki.math.bme.hu típusok mellett a "http://wiki.math.bme.huprotected"http://wiki.math.bme.hu típus. Minden ami publikus vagy protected az ?sosztályban, az szintén eleme lesz a származtatott osztálynak. Ne feledjük, hogy pythonban minden változó és metódus publikus.
- A fentiek már majdnem egyértelm? szabályokat adnak. Kérdés még, hogy abban az esetben, ha azonos metódusokat definiálunk az ?s és az alosztályban, akkor melyik lesz érvényes az

alosztályban?

- Pythonban minden metódus virtuális. Ez azt jelenti, hogy az el?z? kett?s definiálás esetén az alosztály definíciója lesz érvényes, tehát **felüldefiniáljuk** az eredeti metódust.

Példák:

```
>>> class A:
...   x=8
...   def __init__(self):
...       self.u=6
>>> class B(A):
...   y=5
...   def __init__(self):
...       self.v=7
>>> a=A()
>>> b=B()
```

- A fenti példában az a onjektum x és u, mg a b objektum x,y és v változókat fog magába foglalni.
- El?fordulhat, hogy az ?sosztály egy metódusára szeretnénk hivatkozni az alosztály egy metódusában. Ezt a következ?képp tehetjük meg:

```
class A(object):
    def proba(self):
      print "A osztaly"

class B(A):
    def proba2(self):
      super(B, self).proba()
      print "B osztaly"
```

- A fenti példában a proba2() metódus meghívja az A osztályban definiált proba() metódust.