Informatika Saila

Python programazio-lengoaia

Iruñea, 2003ko uztailaren 15etik 17ra

Garikoitz Araolaza (CodeSyntax www.codesyntax.com)

Mark Pilgrim-en "Dive Into Python" liburutik eratorritako dokumentazioa.

1. Murgil zaitez Python-en!

1.1. Murgil gaitezen

1.2. Funtzioak deklaratu

```
def funtzioa( parametroa1, parametroa2, ...) :
```

- Ez da emaitzaren tipoa zehazten. Agian itzuliko du zerbait.

- Parametroen tipoa ez da zehazten.

1.3. Dokumentazioa

```
def buildConnectionString (params):
      """Build a connection string from a dictionary of
parameters.

      Returns string"""
```

- 3 komatxoak erabiltzen dira lerro anitzek osatutako kateak zehazteko.

- Funtzioaren `doc string` edo dokumentazio karaktere katea

1.4. Dena da objektua

- Funtzio bat, (Pythonen beste edozer bezala) objektu bat da.

```
import odbchelper

print odbchelper.buildConnectionString.__doc__
```

- Non bilatzen du Pythonek?

```
import sys

print sys.path

sys.path.append('/my/new/path')
```

- Badaude ere "built-in" moduluak, hots, Python-ekin barruan datozenak. (Ez daude Python-en programatuta)

- **EZ AHAZTU:** dena da objektua Python-en (karaktere kateak, zerrendak, funtzioak... moduluak ere objektuak dira!)

1.5. Indentazioa

- Blokeak indentazioaren bidez markatzen dira. Ez dago "begin/end" edo giltzik.

- Mesedez, erabili **4 hutsuneko indentazioa.** *(Python Style Guide)*

- Lasai, azkenean gustatuko zaizue. ;-)

1.6. Moduluak testeatzea

```
if __name__== "__main__":
```

- Zuzenean modulua exekutatzen badugu (`python odbchelper`),

`__name__== "__main__"` egia izango da.

1.7. Hiztegiak (dictionaries):

- Hiztegiek klabeen eta balioen arteko erlazioak definitzen dituzte.

```
d = {'server': 'mpilgrim', 'database': 'master'}
```

- Klabearekin -> Balioa lor daiteke. Alderantziz ez.

- Klabeak bakarrak dira.

- Klabe-Balio pareak edozein momentutan gehitu daitezke.

- Ez dago orden jakinik.

- Hiztegien balioak edozein motatakoak izan daitezke.

- Klabeak: testu kateak, zenbaki osoak eta tuplak (Geroago ikusiko ditugu)

- Elementuak ezabatzeko:

```
del d['database']
```

- Hiztegia hustutzeko:

```
d.clear() edo d={}
```

1.8. Zerrendak (lists):

```
i = ['a', 'b', 'mpilgrim', 'z', 'example']
```

- Elementuak jasotzeko

```
li[0] edo li[4]
```

- Indize negatiboak:

```
li[-1]  li[-3]
```

- Zerrenden zatiak:

```
li[1:3]  li[1:-1]
```

- Zatiak azkarrago:

```
li[:2]    li[:]
```

- Elementuak gehitu:

```
li.append('new')
```

- Elementuak txertatu

```
li.insert(2, 'new')
```

- Bi zerrenda bildu

```
li.extend(['two', 'elements'])
```

- Bilaketak:

```
li.index('example')
```

- Bilaketak (2):

```
'x' in li
```

- Zer da "egia" Python-en ? (`True` eta `False`)

- Elementuak ezabatu:

```
li.remove('new')
```

- Elementuak jaso:

```
li.pop()
```

- Batuketak zerrendekin:

```
li = li + ['example', 'new']
```

- Batuketak motzago:

```
li += ['two']
```

- Biderketak:

```
li = [1, 2] * 3
```

1.9. Tuplak (tuples):

- Zerrenda bat bezala definitzen da, baina parentesiekin

- Zerrendek daukaten indize sistema berbera daukate

- BAINA: tuplek ez dute metodorik

- `in` egin daiteke tuplekin

- Zertarako balio dute tuplek?

    > Zerrendak baino askoz azkarragoak dira

    > Hiztegietako klabe izan daitezke

    > Testu kateak formatatzeko

- `tuple()` eta `list()` metodoei esker zerrenda bat tupla bihur dezakegu (eta alderantziz)

1.10.Aldagaien definizioa:

- Aldagaiak ez dira deklaratzen. Balio bat ematearekin nahikoa da.

- Automatikoki garbitzen dira ingurunetik kanpo gelditzen direnean.

```
if __name__ == '__main__':
    myParams = {'server':'mpilgrim', \
                'database':'master', \
                'uid':'sa', \
                'pwd':'secret' \
                }
```

- Lerro luzeak \ sinboloarekin moztu daitezke.

- Parentesi eta antzekoak itxi arte lerrotan bana dezakegu.

- Aurretik definitu gabeko aldagaiak erabiltzean salbuespenak sortzen dira.

1.11.Aldagai bat baino gehiago batera definitu:

- Aldagai anitzak batera definitu daitezke:

```
(x, y, z) = ('a', 'b', 'e')
```

- Bai eta funtzioekin lotuta (`range()`)

1.12.Testu kateei formatua ematea:

- C-ko `sprintf`-ren sintaxia erabiltzen da.

```
"%s=%s" % (k, v)
```

- Ez da konkatenazioa bakarrik; aldagai motak ere moldatzen ditu.

1.13.Zerrenden mapaketa

- Zerrenden *ulermena (list comprehension)*

```
[elem*2 for elem in li]
```

- Gure adibidean:

```
["%s=%s" % (k, v) for k, v in params.items()]
```

- Hiztegien metodo bereziak:

```
keys()

values()

items()
```

1.14.Zerrendak elkartu eta karaktere kateak moztu

```
";".join(["%s=%s" % (k, v) for k, v in params.items()])
```

- `";"` katea ere objektu bat da, eta `join` bere metodoa da.

```
s = "server=mpilgrim;uid=sa;database=master;pwd=secret"
s.split(';', 1)
```

2. Introspekzioaren indarra

1.15.apihelper.py

1.16.Aukerazko parametroak eta izendatuak

```
def help(object, spacing=10, collapse=1):
```

- `spacing` eta `collapse` aukerazko parametroak dira.

- Dei posibleak:

```
help(odbchelper)
help(odbchelper, 12)
help(odbchelper, collapse=0)
help(spacing=15, object=odbchelper)
```

1.17.`type, str, dir` eta beste funtzio integratuak (__builtin__)

```
type(1), type("kaixo")

str(x)  # Edozein objekturi aplika diezaiokegu

dir(x)  # Edozein objekturi dagozkion metodoen zerrenda

callable(x) # atributu bat exekutagarria ote da?

__doc__   # Edozein metodoren azalpena

__builtin__ # Berez Python-en inkorporatuta datozen funtzio eta atributuak.

from apihelper import help
import __builtin__
help(__builtin__, 20)
```

- Python *self-documenting* edo berez dokumentatzen den lengoaia da!

1.18.Objektuen erreferentziak lortzea `getattr` erabiliz.

`li.pop` eta `getattr(li, 'pop')` gauza berdina dira

- `getattr` funtzioari esker, metodoak string bezala eman ditzazkegu

- Berdin gertatzen da modulu eta objektuekin:

```
getattr(object, method)
```

1.19.Zerrendak galbahetu

```
[ mapping-expr  for element in source-list if filter ]
```

Adibidez:

```
[ elem for elem in li if li.count(elem) == 1 ]
```

1.20.`and` eta `or` nola erabili

```
'a' and 'b'
'' and 'b'

'a' and 'b' and 'c'

'a' or 'b'

'' or 'b'

'' or [] or {}
```

- `and-or` tranpa

```
a = 'Lehena'

b = 'Bigarrena'

1 and a or b  # Kontuz, a ezin da False izan!

0 and a or b
```

1.21.Lambda funtzioak

- Lambda funtzioei esker, mini-funtzioak sor daitezke momentuan. (Lisp-etik lapurtua)

```
def f(x):
      return x*2

f(3)

g = lambda x: x*2

g(3)
```

- Zerbait konplexuagoa

```
processFunc = collapse and (lambda s: " ".join(s.split())) or \
      (lambda s: s)
```

1.22.Dena batera jar dezagun

- apihelper.py-ren funtsa:

```
print "\n".join(["%s %s" %  (method.ljust(spacing),
      processFunc(str(getattr(object, method).__doc__)))
      for method in methodList])
```

- Dokumentazio katea dinamikoki atera:

```
print getattr(object, method).__doc__
```

- `ljust` metodoa : kateak ezkerrera justifikatzeko.

- Zerrendak inprimatu:

```
print "\n".join(li)
```

3. Objektueri zuzendutako ingurunea

1.23.Murgil gaitezen: `fileinfo.py`

1.24.Moduluen importazioa

```
import types

from types import FunctionType
```

1.25.Klaseen definizioa

```
from UserDict import UserDict

class FileInfo(UserDict): # Parentesi artean herentzia
    "store file metadata" # Dokumentazioa
        def __init__(self, filename=None):
            UserDict.__init__(self)
            self['name'] = filename # Klasea jadanik hiztegia da.
```

- Herentzia anitza onartzen da: `class foo(classA, classB, classC):`

- `__init__` metodoa ez da *constructor* metodoa, baina antzerako lanak egiten ditu.

- Klaseen metodoen lehen argumentua, beti da objektua bera: `self`

- `__init__` metodoaren lehenengo lana, arbasoaren `__init__` metodoa exekutatzea.

- __init__ metodoa ez da beharrezkoa, baina jartzen bada, beti exekutatu klase arbasoaren __init__ metodoa.

- Klase kanpotik metodo klase bati deitzean, ez eman self-ik. Pythonek automatikoki egiten du. Ez da inkonsistentzia bat, badago arrazoi bat.

1.26.Klaseen instantziak sortuz

```
import fileinfo

f = fileinfo.FileInfo("/musika/kairo.mp3")
```

- __class__ __name__ eta __bases__ atributuak

- Instantziak desagertaraztea oso erraza da. Sortua izan den ingurunetik ateratzen garenean automatikoki gertatzen da.

1.27.Klaseen metodo bereziak

- Zenbait metodo berezi definitu daitezke klaseetan, Pythonek gure ordez exekutatzen dituenak zenbait egoeratan.

```
f.__getitem__('name')              f['name']

f.__setitem__('genre', 31)         f['genre'] = 31
```

- Zenbait kasutan gure klaseetan metodo hauek berridaztea komeni zaigu asaben metodoez gain beste ekintza batzuek egiteko. Edozein kasutan ez dugu asaben metodoa exekutatzea ahaztu behar.

1.28.Klaseen metodo berezi aurreratuak

`__repr__`

`__cmp__`

`__len__`

`__delitem__`

1.29.Klaseen atributuak

- Klaseen atributuak sortzeko, nahikoa da klasean bertan definitzea.

- Ez dira konstanteak, eta erreferentzia egin diezaiokegu: `self.__class__.count`

1.30.Funtzio pribatuak

- Funtzio pribatuak ezin dira modulo kanpotik deituak izan.

- Pribatuak: `__parse`

- Bereziak: `__setitem__`

1.31.Salbuespenak kudeatu

- Adibidez, existitzen ez den fitxategi bat zabaltzerakoan:

```
try:
    p=open('fitxategirik_ez.txt')
except IOError:
    print 'Fitxategi hori ez dugu aurkitzen'
print 'Hau inprimatuko da, fitxategia zabaldu ala ez'
```

1.32.Fitxategi objektuak

```
f = open('/musika/ikastaroa/bat.mp3', 'rb')

f.mode

f.name

f.tell()

f.seek(-128, 2)

f.read(128)

f.closed

f.close()
```

1.33.for begiztak

- for instrukzioak, zerrrenden inguruan sortzen ditu begiztak

```
li = ['a', 'b', 'd', 'e']
for s in li:
    print s

print "\n".join(li)
```

- Hiztegi baten inguruan begizta:

```
import os
for k, v in os.environ.items():
    print "%s=%s" % (k, v)
```

1.34.Moduluen inguruan gehiago

- Zein modulo kargatuta dauzkagun jakiteko:

```
import sys
print "\n".join(sys.modules.keys())
```

- Zein modulutan dago definituta klase jakin bat?

```
from fileinfo import MP3FileInfo
MP3FileInfo.__module__
```

1.35.os modulua

- os moduluak fitxategiak eta prozesuak kudeatzeko funtzio ugari dakartza

- os.path erabil dezakegu fitxategi eta karpeta izenekin eragiketak egiteko

```
os.path.join( )

os.path.expanduser('~')

os.path.split( )

os.path.splitext( )
```

- Beste funtzio batzuk

```
os.listdir('.')

os.path.isfile( )

os.path.isdir( )
```

Hiztegitxoa:

tuple: tuplo edo n-kote
string: katea
parameter: parametroa
variable/data type: datu mota
to map – mapping: mapaketa
list comprehension: ulermena
method: metodoa
optional: aukerazko parametroa
built-in: inkorporatuak
ancestor class: asaba klasea