Python (un poco más) avanzado

Jesús Espino García
jespinog@gmail.com
@jespinog

Gul UC3M Nov 2011

Kaleidos
beautiful code

7 de Noviembre de 2011

Indice

. . .

- http://docs.python.org/library/index.html
- http://pypi.python.org

Indice

Comprehensions

```
>>> [ x*10 for x in range(1,10) if x%2==0 ]
[20, 40, 60, 80]

>>> [ [y for y in range(1,x)] for x in range(1,10) ]
[[],
 [1],
 [1, 2],
 [1, 2, 3],
 [1, 2, 3, 4],
 [1, 2, 3, 4, 5],
 [1, 2, 3, 4, 5, 6],
 [1, 2, 3, 4, 5, 6, 7],
 [1, 2, 3, 4, 5, 6, 7, 8]]
```

Comprehensions

```
# A partir de 2.7
>>> { x:x*10 for x in range(1,10) }
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 7: 70, 8: 80, 9: 90}
```

Funciones lambda

```
def pow(x):
    return x*x
```

Es equivalente a:

```
pow = lambda x: x*x
```

Closures

Funciones que se aplican a una lista de elementos:

```
>>> filter(lambda x: x>5, range(1,10))
[6, 7, 8, 9]

>>> map(lambda x: x*x, range(1,10))
[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> reduce(lambda x,y: x+y, range(1,10))
45

>>> sorted([7,4,5,9,1,3,2,6,8], lambda x,y: cmp(y,x))
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Herencia multiple

```
# Estilo nuevo (de manera directa o indirecta
# hereda de object)
class ClaseDerivada(Base1, Base2, Base3):
    def metodo(self):
        super(ClaseDerivada, self).metodo()

# Estilo antiguo (no hereda de object)
class ClaseDerivada(Base1, Base2, Base3):
    def metodo(self):
        Base1.metodo()
```

Context managers

```
from contextlib import closing

with closing(file('fichero.txt','r')) as fichero:
    for line in fichero:
        print line
```

Indice

Context managers

```
class MyManager(object):
    def __enter__(self):
        print "Entrando en el contexto"
    def __exit__(self, *args):
        print "Saliendo del contexto"

with MyManager():
    print "Dentro del contexto"
```

Generadores e iteradores

```
def generador():
    for x in range(1,10):
        yield x
class Iterador(object):
    def __iter__(self):
        for x in range(1,10):
            yield x
```

Generadores e iteradores

```
class Iterador2(object):
    def __init__(self):
        self.max = 10
        self.counter = 0

    def __iter__(self):
        return self

    def next(self):
        self.counter += 1
        if self.counter<self.max:
            return self.counter
        else:
            raise StopIteration
```

Metaclases

Sirven para definir nuestros propios tipos

```
class Meta(type):
   def __getattribute__(*args):
      print "Metaclass getattribute invoked"
      return type.__getattribute__(*args)

class C(object):
    __metaclass__ = Meta
    def __len__(self):
        return 10
    def __getattribute__(*args):
        print "Class getattribute invoked"
        return object.__getattribute__(*args)
```

Metaclases

```
>>> c = C()
>>> c.__len__()                 # Explicit lookup via instance
Class getattribute invoked
10
>>> type(c).__len__(c)          # Explicit lookup via type
Metaclass getattribute invoked
10
>>> len(c)                      # Implicit lookup
10
```

Indice

Gestionar software

```
$ pip search django
$ pip install Django
$ pip uninstall Django
$ pip install Django==1.3
```

Entornos virtuales

```
$ mkvirtualenv mi_directorio
$ cd mi_directorio
$ source bin/activate
$ pip install Django==1.3
$ cd ..
$ mkvirtualenv otro_directorio
$ cd otro_directorio
$ source bin/activate
$ pip install Django==1.2
$ deactivate
```

Empaquetar bibliotecas

El sistema de empaquetado de python se basa en un fichero setup.py

```
#!/usr/bin/env python

from distutils.core import setup

setup(name='MiBiblioteca',
      version='1.0',
      description='Da charlas para que yo me pueda quedar tomando ur
      author='Jesús Espino',
      author_email='jesus.espino@kaleidos.net',
      url='http://www.kaleidos.net',
      packages=['mibiblioteca'],
     )
```

http://docs.python.org/distutils/setupscript.html

Debugging (PDB)

```
import pdb; pdb.set_trace()
```

IPython y IPDB

Interpretes ricos de python y pdb.

Profiling

Desde el codigo:

```
import cProfile
cProfile.run('foo()')
```

Desde el interprete:

```
$ python -m cProfile myscript.py
```

Cython

```
from libc.math cimport sin

cdef double f(double x):
    return sin(x*x)

def hellocython(x):
    return "Hola cython %f" % (f(x))

def hellocython_direct(x):
    return "Hola cython %f" % (sin(x*x))
```

Indice

Referencias

- Proyecto python: http://www.python.org.
- Documentación de python: http://docs.python.org.
- Python-ES: python-es@python.org

Dudas

. . .