Python spiccio

Ottavio Campana

11 settembre 2013

cosa e perché

Python è un linguaggio molto potente ma allo stesso tempo molto semplice da imparare.

Ci sono tuttavia alcuni punti meno intuitivi o meno documentati, che possono far perdere molto tempo.

Questo documento è una raccolta di soluzioni a problemi ricorrenti per chi si avvicina al python provenendo da altri linguaggi.

enum

python < 3.4

```
1  def enum (*sequential, **named) :
       enums = dict (zip (sequential, range (len (sequential))), **named)
3      return type ('Enum', (), enums)

5  Animals = enum (ANT=1, BEE=2, CAT=3, DOG=4)

7  print Animals.ANT

9  OtherAnimals = enum ('ANT', 'BEE', 'CAT', 'DOG')

11 print OtherAnimals.ANT
```

enum

python 3.4 ha `enum`

```
1 from enum import Enum

3 class Animals(Enum):
      ANT = 1
5     BEE = 2
      CAT = 3
7     DOG = 4
```

switch - case

python non ha lo `switch - case` però si può fare una cosa simile

```
1  def case_A ():
       print 'case_A'
3
   def case_B ():
5      print 'case_B'

7  def case_unknown ():
       print 'not a valid case'
9
   {
11     'A':  case_A,
       'B':  case_B,
13 }.get (choice, case_unknown) ()
```

prompt

realizzare rapidamente una shell per lanciare comandi

```
1  import cmd

3  class SwitchCase (cmd.Cmd):
       prompt = '=>'
5
       def do_A (self, arg):
7          print 'A ' + arg

9  SwitchCase ().cmdloop ()
```

ricercare un dizionario in una lista

l'equivalente di `select * from table where ...`

```
a = [ {'name':'pippo', 'age':'5'} ,
      {'name':'pluto', 'age':'7'} ]

item = [d for d in a if d['name'] == 'pluto']

print item
```

determinare dinamicamente quali parametri passare ad una funzione

non usare `eval` su comandi python contenuti in stringhe costruite dinamicamente, perché si rischia code injection, soprattutto in funzioni che hanno a che fare con il web.

```
def my_function (name=None, surname=None, age=None):
    print name
    print surname
    print age

params = {'surname': 'foobar' }

my_function (**params)
```

*args e **kwargs

*args viene usato per passare una tupla, **kwargs per passare un dizionario.

```
def test_args (first , *args):
    print 'first ' + first
    for arg in args:
        print 'arg ' + arg

def test_kwargs (first , **kwargs):
    print 'first ' + first
    for key in kwargs:
        print 'arg ' + kwargs[key]


test_args ('A', 'B', 'C')
test_args ('A', *('B', 'C'))

test_kwargs ('A', **{'one':'B', 'two':'C'})
```

Catturare l'output di un programma

Usare il modulo subprocess

```
1 import subprocess

3 p = subprocess.Popen (['ls', '-al'], stdout=subprocess.PIPE)

5 out, err = p.communicate ()
```

le variabili `out` ed `err` vengono popolate alla fine dell'esecuzione del comando. Una soluzione interattiva è proposta in 12.

Validare le varibili di una classe

I membri degli oggetti python sono tutti pubblici. È tuttavia possibile validare i dati, agganciando degli attributy. Da python 2.2 al posto di property è inoltre possibile usare dei decoratori.

```
class Mydata (object):
    def __init__ (self, value=0):
        self.value = value

    def get_value (self):
        return self.value

    def set_value (self, value):
        if value not in range (0, 10):
            raise ValueError ('value is not an integer in [0, 9]')
        self.__value = value

    value = property (get_value, set_value)

data = Mydata ()
data.value = 10 # Boom!
```

catturare l'output di un programma da WX widget 1/2

```
import wx
import functools
import threading
import subprocess
import time

class Frame (wx.Frame):
    def __init__ (self):
        super (Frame, self).__init__ (None, -1, 'Threading Example')
        # add some buttons and a text control
        panel = wx.Panel (self, -1)
        sizer = wx.BoxSizer (wx.VERTICAL)

        # add a button
        button = wx.Button (panel, -1, 'Start')
        func = functools.partial (self.on_button, button=button)
        button.Bind (wx.EVT_BUTTON, func)
        sizer.Add (button, 0, wx.ALL, 5)

        text = wx.TextCtrl (panel, -1, style=wx.TE_MULTILINE|wx.TE_READONLY)
        self.text = text
        sizer.Add (text, 1, wx.EXPAND|wx.ALL, 5)
        panel.SetSizer (sizer)

    def on_button (self, event, button):
        # create a new thread when a button is pressed
        thread = threading.Thread (target=self.run, args=(button,))
        thread.setDaemon (True)
        thread.start ()
```

catturare l'output di un programma da WX widget 2/2

```
    def on_text (self, text):
2       self.text.AppendText (text)

4   def run (self, button):
        cmd = [ 'bash', '-c', 'for i in {1..10} ; do echo $i ; sleep 1 ; done' ]
6       proc = subprocess.Popen (cmd, stdout=subprocess.PIPE, stderr=subprocess.
     STDOUT)

8       while True:
            out = proc.stdout.read (1)
10          if out == '' and proc.poll() != None:
                break
12          if out != '':
                wx.CallAfter (self.on_text, out)
14
if __name__ == '__main__':
16      app = wx.PySimpleApp()
        frame = Frame()
18      frame.Show()
        app.MainLoop()
```

indipendenza dal driver del db

Se si vuole poter scrivere un programma che carica il driver del database in base ad un valore salvato esternamente, per esempio in un file di configurazione

```
dbtype = 'psycopg2'# from config file
dbdriver = __import__ (dbtype, globals (), locals (), [], -1)

connection = dbdriver.connect (...)
```

passaggio parametri alle query sicuro

Il modo corretto di fare l'escaping dei parametri è di farlo fare al driver del database. I parametri possono essere passati come dizionario o come lista.

```
cursor.execute ('update customers set password = %(pw)s
                 where customer_id = %(id)d',
                {'id':37, 'pw':'secret'})
```

```
cursor.execute ('update customers set password = %s
                 where customer_id = %d',
                ['secret', 37])
```

passaggio parametri multiplo alle query sicuro

`cursor.executemany` può essere usato per richiamare più execute, passando i parametri in una lista.

```
cursor.executemany ('update customers set password = %s
                     where customer_id = %d',
                    [['secret', 37],
                     ['kjhksj', 93]])
```

usare listen - notify di postgresql

```
connection.set_isolation_level (psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)

cursor = connection.cursor ()
cursor.execute ("LISTEN output;")

while True:
    if select.select ([connection],[],[],2) == ([],[],[]):
        pass
    else:
        connection.poll ()
        while connection.notifies:
            notify = connection.notifies.pop ()
            % handle the notify
```

Su postgres

```
NOTIFY output;
```