

Pemrograman Modul Python

Dalam pemrograman, terkadang kita perlu menggunakan ulang kode-kode program yang pernah kita tulis sebelumnya. Hal tersebut tentu akan sangat merepotkan apabila setiap ingin menggunakan fungsi yang telah dibuat, Anda harus menulis ulang. Oleh karena itulah, diperlukan modul.

Bagi Anda yang bekerja di lingkungan *interpreter* Python, semua fungsi yang ditulis akan hilang begitu saja sewaktu Anda keluar dari *interpreter*. Untuk menghindarinya, sebaiknya Anda menyimpan fungsi-fungsi yang telah Anda buat dalam sebuah file modul.

Apa itu modul?

Di dalam Python, fungsi-fungsi *built-in* atau fungsi yang kita *import* disimpan dalam file eksternal yang disebut modul. Sebuah modul adalah sebuah file yang berisi kumpulan fungsi-fungsi dan instruksi-instruksi program Python. Modul ini biasa disimpan dalam extension *.py*. Selanjutnya, modul-modul ini dapat kita pakai berulang-ulang dalam pembuatan aplikasi dengan memanggil fungsi *import nama_modul*. Dengan demikian, Anda akan menghemat waktu dalam pembuatan aplikasi tanpa harus menulis ulang fungsi-fungsi yang telah Anda buat sebelumnya.

Pada saat Anda memanggil *interpreter* Python, sebenarnya Python sendiri telah meng-import modul *__builtin__*. Anda dapat melihatnya dengan perintah *dir()*. Perintah *dir(objek)* akan menampilkan semua atribut dan fungsi yang terdapat dalam sebuah objek (dalam hal ini objek modul).

```
>>> dir()
['_builtin__', '_doc_', '_name_']
>>> dir(_builtin__)
...
...
>>> import sys
>>> dir(sys)
...
...
>>>
```

Modul-modul Python selain dapat ditulis dengan bahasa Python sendiri, juga dapat ditulis memakai bahasa C/C++ ataupun

Java untuk Jython. Pembuatan modul dengan bahasa C/C++ maupun Java tidak akan dibahas mengingat materi ini termasuk rumit. Kita hanya akan membahas pembuatan dan pendistribusian modul Python murni.

Akan tetapi ada satu hal yang perlu Anda ingat sebelum membuat suatu modul, mungkin secara tidak sengaja, Anda sering membuat fungsi sendiri yang sebenarnya telah disediakan Python. Hal ini tentunya akan memakan waktu dan kemungkinan besar kode-kode yang kita buat tidak seoptimal yang dibuat oleh para pengembang Python. Oleh karenanya, gunakanlah sumber daya yang telah disediakan oleh Python. Jika modul yang akan diinginkan tidak tersedia, barulah Anda membuat modulnya.

Modul yang akan kita buat di bawah ini merupakan contoh sederhana saja untuk menunjukkan bagaimana membuat suatu modul, Anda bisa memodifikasinya lebih



lanjut. Berikut kita akan membuat sebuah modul untuk menghitung luas bangun sebagai berikut:

```
#Luas.py
"Modul 'tuk menghitung luas bangun"

print 'Loading module luas...'
#pendeklarasian variabel
PHI=3.14

def L_lingkaran(r):
    return PHI*(r*r)

def L_persegi(s):
    return s*s

def L_persegipanjang(p,l):
    return (p*l)
```

▲ Menampilkan isi modul dengan *dir()*.

```
def L_segitiga(a,t):
    return 0.5*(a*t)

def L_trapesium(a,b,t):
    return (a+b)/2*t

print 'Module loaded.'
```

Python selalu mencari file modul yang di-import berdasarkan isi variabel shell PYTHONPATH dan di direktori aktif. Misalkan modul Anda disimpan di direktori aktif dan Anda memanggil interpreter Python, Anda tidak perlu melakukan apa-apa. Akan tetapi jika modul Anda berada di direktori lain, Anda perlu mengubah isi variabel shell PYTHONPATH atau kita juga dapat memanggil modul yang kita buat dengan cara berikut. Misalkan modul yang penulis buat disimpan di /home/eric/modul maka hal yang pertama yang harus dilakukan adalah meng-import modul sys. Kemudian kita menambahkan path di mana modul yang kita buat berada dengan fungsi sys.path.append('/home/eric/modul'). Untuk lebih jelasnya, lihat contoh berikut:

```
$ pwd
/home/eric/modul
$ ls
luas.py
$ python
>>> import luas
Loading module luas...
Module loaded.
>>> dir(luas)
['L_lingkar', 'L_persegi',
'L_persegipanjang', 'L_segitiga',
'L_trapesium', 'PHI', '__builtins__',
'__doc__', '__file__', '__name__']
>>>
$ cd ..
$ pwd
/home/eric
$ python
>>> import luas
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named luas
>>> import sys
>>> sys.path.append('/home/eric')
>>> import luas
Loading module luas...
Module loaded.
```

```
>>> print luas.L_persegi(4)
16
```

Untuk selanjutnya, dianggap modul yang sementara kita buat ini disimpan di direktori aktif (dalam hal ini /home/eric/modul). Sebelum membahas lebih lanjut tentang pembuatan modul ini, kita akan membahas beberapa cara pemanggilan modul.

1. Bentuk import *nama_modul*

Melakukan import suatu modul sebagai satu kesatuan.

Bentuk ini memungkinkan kita menggunakan semua atribut maupun fungsi yang dideklarasikan dalam *nama_modul.py*, dengan syarat kita harus menyebutkan nama modulnya. Berikut contohnya:

```
>>> import luas
Loading module...
Module loaded.
>>> luas.L_persegi(5)
25
```

2. Bentuk from *nama_modul* import *

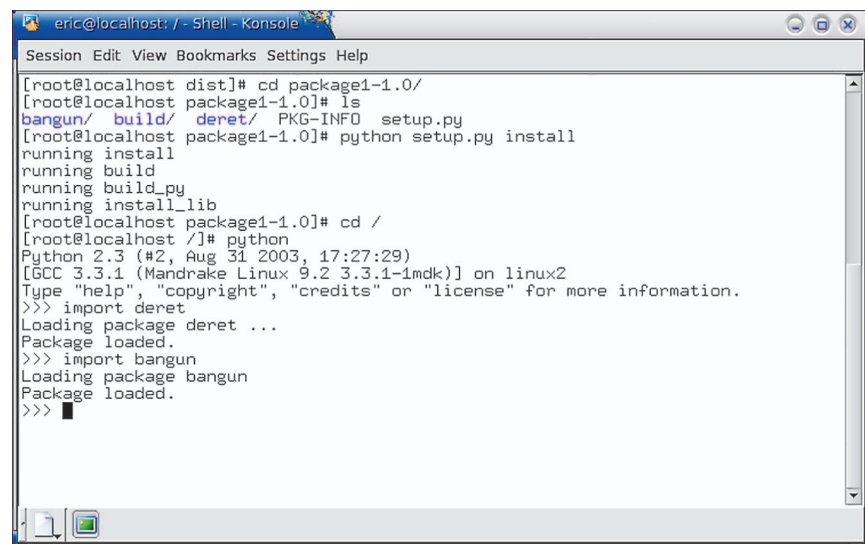
Melakukan impor semua fungsi maupun atribut dari modul.

Bentuk ini menjadikan fungsi dalam modul kita menjadi global, sehingga Anda dapat memanggil langsung fungsinya tanpa harus menyertakan nama modulnya. Untuk meng-import hanya satu atau beberapa fungsi, gantilah * dengan nama fungsi/atribut yang hendak diimpor. Berikut cara penggunaannya:

```
$ python
>>> from luas import L_segitiga
Loading module...
Module loaded.
>>> L_segitiga(4,5)
10.0
>>> L_persegi(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'L_persegi' is not defined
>>>
$ python
>>> from luas import *
Loading module luas ...
Module loaded.
>>> L_persegi(4)
16
>>> L_lingkar(10)
314.0
```

3. Bentuk reload(*nama_modul*)

Kita dapat melihat bahwa penggunaan modul hanya akan dilakukan sekali dalam setiap proses. Beberapa modul mungkin membutuhkan perubahan nilai atributnya, yang akan dipakai oleh proses selanjutnya. Sekali kita melakukan import, maka nilai yang didapat dari hasil import tersebutlah yang akan dipakai selanjutnya. Perubahan pada atribut modul ataupun melakukan import ulang tidak akan memberikan efek yang kita inginkan. Untuk menyesuaikan perubahan atribut modul, kita bisa menggunakan bentuk reload(*nama_modul*). Untuk jelasnya, lihat contoh berikut:



```
eric@localhost: / - Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@localhost dist]# cd package1-1.0/
[root@localhost package1-1.0]# ls
bangun/ build/ deret/ PKG-INFO setup.py
[root@localhost package1-1.0]# python setup.py install
running install
running build
running build_py
running install_lib
[root@localhost package1-1.0]# cd /
[root@localhost ~]# python
Python 2.3 (#2, Aug 31 2003, 17:27:29)
[GCC 3.3.1 (Mandrake Linux 9.2 3.3.1-1mdk)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import deret
Loading package deret ...
Package loaded.
>>> import bangun
Loading package bangun
Package loaded.
>>> █
```

▲ Contoh penggunaan modul.

```
$ cat > a.py
versi="versi pertama"

def cetak():
    print versi
$ python
>>> import a
>>> a.cetak()
versi pertama
```

Biarkan interpreter Python tetap aktif, sementara Anda membuka sebuah terminal baru dan edit file a.py dengan editor vi.

```
$ vi a.py
$ cat a.py
versi="versi pertama, dengan revisi."

def cetak():
    print versi
```

Sesudah itu, Anda kembali ke interpreter python yang masih aktif.

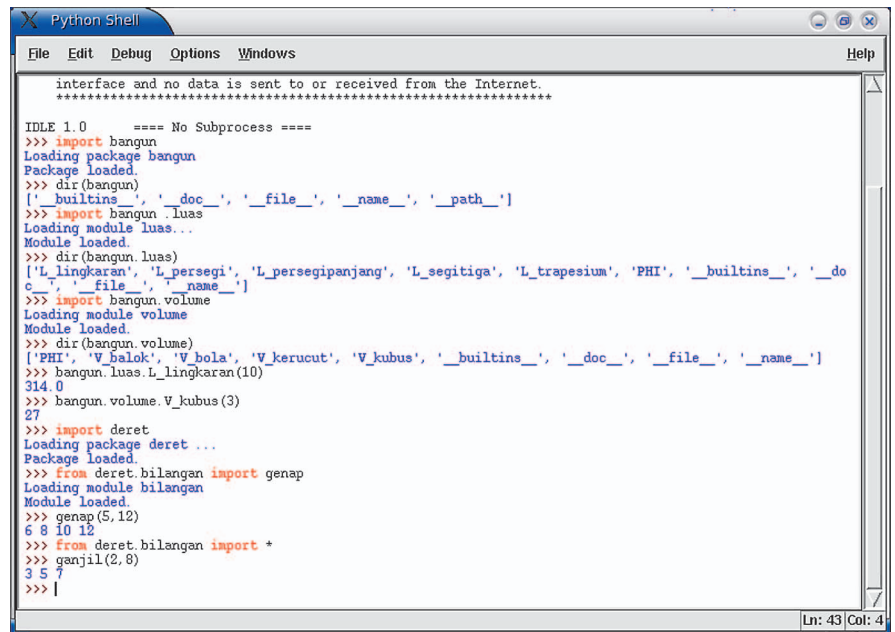
```
>>> a.cetak()
versi pertama
>>> import a
>>> a.cetak()
versi pertama
>>> reload(a)
>>> a.cetak()
versi pertama, dengan revisi.
```

Walaupun pengubahan atribut yang demikian tidak disarankan, mungkin ada kalanya Anda memerlukannya. Penggunaan bentuk ini berguna dalam sistem yang besar, di mana me-*restart* suatu aplikasi akan berakibat fatal. Misalnya suatu sistem yang terhubung ke server melalui jaringan. Bila kita menghentikan sistem tersebut, tentu akan berakibat fatal di mana semua koneksi dalam jaringan akan terputus. Jadi penggunaan `reload(nama_modul)` akan diperlukan dalam hal ini.

4. Bentuk `import nama_module as alias`

Nama modul yang kita impor dapat digantikan nama alias dengan perintah di atas.

```
>>> import luas as L
Loading module luas...
Module loaded.
>>> L.L_persegi(6)
36
```



▲ Penggunaan modul dengan IDLE.

5. Bentuk `import nama_paket.nama_modul` atau `from nama_paket.nama_modul import *`

Jumlah modul yang kita buat mungkin saja berjumlah banyak. Alangkah baiknya jika kita mengelompokkannya dalam sebuah direktori yang lazim disebut *package*. Secara umum, package dan modul di dalamnya diwakili dengan direktori(nama package) dan file-file di dalamnya (modul).

Berikut contohnya jika Anda pernah memakai wxPython.

```
>>> from wxPython.wx import *
>>> from wxPython.grid import *
>>> from distutils.core import setup
```

Untuk penjelasan lebih tentang package ini, kita akan membahas lebih lanjut dalam pembuatan package di bawah.

Setelah memahami cara pemanggilan modul di atas, kita kembali membahas tentang pembuatan modul. Modul yang telah Anda buat bisa kita distribusikan ke teman Anda ataupun lewat Internet. Akan tetapi, mungkin timbul berbagai kendala yang dihadapi seperti masalah perbedaan OS dan konfigurasi sistem ataupun path tiap user yang berbeda. Hal tersebut jelas menjadi kendala berarti yang harus diatasi. Tidak praktis dan merepotkan tepatnya jika setiap orang yang menggunakan modul Anda harus mengubah variabel shell

PYTHONPATH ataupun memanggil `sys.path.append()` setiap ingin menggunakan modul yang Anda buat.

Di sisi lain, kita tahu bahwa konfigurasi OS dan path tiap user berbeda. Hal tersebut tentu sangat merepotkan sehingga bisa-bisa aja user tidak ingin lagi menggunakan modul Anda. Mungkin Anda pernah memakai *third-party modules* dalam bentuk RPM ataupun *source codes*, di mana modul tersebut dapat diinstal di sistem Anda dalam sekejap dan dapat berjalan hampir di semua OS. Beberapa distribusi modul yang terkenal antara lain seperti NumPy, PyXML, PIL(Python Imaging Library), MySQLdb, dan lain-lain. Dalam hal ini, Python sendiri telah menyediakan suatu modul bernama *distutils* yang merupakan singkatan dari *Distribution Utils* yang berperan dalam membuat file distribusi. Pembuatan utilitas *distutils* ini sebenarnya diilhami oleh Perl's MakeMaker System.

Dengan menggunakan *distutils*, kini kendala di atas tidak berarti apapun lagi. Perbedaan OS bukanlah masalah lagi. Cara kerja *distutils* adalah dengan membuat sebuah file bernama `setup.py` yang akan dapat dijalankan secara *multi-platform*. Untuk menjalankannya, Anda hanya perlu mengetik perintah `python setup.py install`. Dengan demikian, modul-modul tersebut akan terinstal dalam direktori Python Anda dan modul tersebut siap pakai. Bagi Anda

yang pernah menggunakan Perl, distutils ini sama halnya dengan Perl's MakeMaker System (ExtUtils::MakeMaker adalah modul yang dipakai dalam Makefile.PL). File Makefile.PL pada Perl sama dengan file setup.py pada Python. Dengan adanya modul distutils pada Python, pendistribusian modul kini menjadi lebih praktis.

Sebenarnya pendistribusian modul terdiri atas dua macam, yakni pendistribusian per modul dan pendistribusian per package. Kita akan membahas keduanya di bawah ini.

Pendistribusian per modul

Berikut struktur direktori modul luas yang kita buat.

```
<modul>/
  setup.py
  luas.py

#setup.py
from distutils.core import setup

setup(name="luas", version="1.0",
      description="Modul penghitung luas",
      author="Eric",
      author_email="eric85id@plasa.com",
      url="http://www.geocities.com/eric/",
      py_modules=["luas"])
```

Selanjutnya adalah membuat file distribusinya sebagai berikut:

1. File distribusi bentuk *.tar.gz

Untuk membuat file distribusi ini, kita cukup memanggil perintah Python setup.py sdist. Berikut contohnya:

```
$ python setup.py sdist
running sdist
reading manifest file 'MANIFEST'
creating luas-1.0
making hard links in luas-1.0...
hard linking luas.py -> luas-1.0
hard linking setup.py -> luas-1.0
tar -cf dist/luas-1.0.tar luas-1.0
gzip -f9 dist/luas-1.0.tar
removing 'luas-1.0' (and everything
under it)
$ ls
dist/ luas.py* MANIFEST setup.py~
$ cd dist
$ ls
luas-1.0.tar.gz
```

Kini modul yang kita buat telah siap didistribusikan. Untuk menginstal modul yang kita buat, pertama kita mengekstrak file-nya, kemudian menjalankan perintah python setup.py install. Semuanya akan langsung terinstal dan dapat dijalankan dengan memanggil interpreter Python. Untuk menginstall modul ini di OS lain, caranya sama yakni dengan memanggil perintah python setup.py install.

```
$ tar -xvzf luas-1.0.tar.gz
$ cd luas-1.0/
$ ls
luas.py* PKG-INFO setup.py
$ su
# python setup.py install
running install
running build
running build_py
running install_lib
copying build/lib/luas.py -> /usr/lib/
python2.3/site-packages
byte-compiling /usr/lib/python2.3/site-
packages/luas.py to luas.pyc
# cd /
# python
>>> import luas
Loading module...
Module loaded.
>>>
```

2. File distribusi bentuk RPM

Untuk membuat file distribusi dalam bentuk RPM, perintah yang dibutuhkan adalah python setup.py bdist_rpm. Berikut contohnya:

```
$ python setup.py bdist_rpm
...
...
$ cd dist
$ ls
luas-1.0-1.noarch.rpm luas-1.0-
1.src.rpm
$ su
# rpm -ivh luas-1.0-1.noarch.rpm
Preparing... ##### [100%]
1:luas ##### [100%]
```

Pendistribusian per package

Modul yang Anda buat mungkin akan bertambah banyak. Sebaiknya modul-modul tersebut dikategorikan sendiri dalam bentuk package yang telah disinggung di atas.

IKLAN

Berikut langkah-langkah pembuatan package:

- Pertama-pertama kita harus membuat direktori-direktori yang akan dijadikan nama package.
- Setelah itu masuk ke direktori tersebut dan buatlah file `__init__.py`, di mana isinya dapat berupa keterangan ataupun variabel yang penting. Python memerlukan file `__init__.py` sebagai penanda sebuah package. Anda dapat aja membuat file `__init__.py` tanpa mepedulikan isinya.
- Buatlah modul-modul dalam direktori tersebut.

Struktur package yang akan kita buat akan tampak sebagai berikut:

```
<modul>/
  setup.py
  bangun/
    __init__.py
    luas.py
    volume.py
  deret/
    __init__.py
    bilangan.py
```

Di direktori modul kita buat direktori yang akan kita jadikan nama package, yakni direktori bangun dan direktori deret. Terus kita buat file `__init__.py` dan modul-modul di dalam direktori tersebut. File `setup.py` harus dibuat di dalam direktori modul atau di direktori atasnya direktori yang kita buat tadi.

Isi file `setup.py` tidak jauh berbeda dengan yang di atas, hanya perlu ditambahkan paramater packages pada fungsi `setup()`.

```
#setup.py
from distutils.core import setup

setup(name="package1", version="1.0",
      author="Eric",
      author_email="eric85id@plasa.com",
      url="http://www.geocities.com/eric/",
      packages=["bangun","deret"])

# __init__.py untuk paket bangun
'Paket bangun'
print 'Loading package bangun ...'
print 'Package loaded.'
```

Untuk file `luas.py`, sama dengan yang di atas.

```
#volume.py
"Modul 'tuk menghitung volume bangun'"

print 'Loading module volume...'
#Pendeklarasian variabel
PHI=3.14

def V_bola(r):
    return 4/3*PHI*(r*r*r)

def V_kubus(r):
    return (r*r*r)

def V_balok(p,l,t):
    return (p*l*t)

def V_kerucut(r,t):
    return (PHI*r*r*t)/3

print 'Module loaded.'

# __init__.py untuk paket deret
'Paket deret'
print 'Loading package deret ...'
print 'Package loaded.'


#bilangan.py
"Modul 'tuk mencetak deret bilangan'"
print 'Loading module bilangan'
def genap(a,b):
    if(a%2==1):
        a=a+1
    for i in range(a,b+1,2):
        print i,

def ganjil(a,b):
    if(a%2==0):
        a=a+1
    for i in range(a,b+1,2):
        print i,
print 'Module loaded.'
```

Setelah semua file dibuat, tiba saatnya untuk membuat file distribusinya. Pembuatan file untuk distribusi caranya sama dengan yang di atas.

```
$ python setup.py sdist
$ cd dist
$ tar -xvzf package1-1.0.tar.gz
$ cd package1-1.0/
$ su
# python setup.py install
# python
>>> import bangun
```

```
Loading package bangun
Package loaded.
>>> dir(bangun)
['_builtins_', '__doc__', '__file__',
 '__name__', '__path__']
>>> import bangun.luas
Loading module luas...
Module loaded.
>>> dir(bangun.luas)
['L_lingkaran', 'L_persegi',
 'L_persegipanjang', 'L_segitiga',
 'L_trapesium', 'PHI', '__builtins__',
 '__doc__', '__file__', '__name__']
>>> bangun.luas.L_lingkaran(10)
314.0
>>> import bangun.volume
Loading module volume
Module loaded.
>>> dir(bangun.volume)
['PHI', 'V_balok', 'V_bola', 'V_kerucut',
 'V_kubus', '__builtins__', '__doc__',
 '__file__', '__name__']
>>> bangun.volume.V_kubus(3)
27
>>> bangun.volume.V_balok(2,3,7)
42
>>> import deret
Loading package deret ...
Package loaded.
>>> from deret.bilangan import genap
Loading module bilangan
Module loaded.
>>> genap(3,10)
4 6 8 10
>>> from deret.bilangan import *
>>> ganjil(5,11)
5 7 9 11
```

Untuk file distribusi bentuk RPM, caranya sama dengan yang di atas. Kini untuk membuat modul dan mendistribusikannya tidak menjadi kendala lagi. Dengan `distutils`, semua terasa begitu sederhana. `Distutils` juga bisa menangani pendistribusian modul dalam bahasa C/C++ maupun Java (untuk Jython). Akan tetapi, kita takkan membahas materi ini sekarang disebabkan materi ini termasuk rumit. Dengan adanya modul, tentunya Anda bisa menghemat waktu dalam pekerjaan Anda. Ditambah lagi dengan `distutils`, kini modul Anda siap disebarluaskan ke seluruh dunia. Makin praktis, bukan? 

Eric (eric85id@plasa.com)