Python Modules

Modul memungkinkan Anda mengatur kode Python secara logis. Mengelompokkan kode terkait ke dalam modul membuat kode lebih mudah dipahami dan digunakan. Modul adalah objek Python dengan atribut yang diberi nama sewenang-wenang sehingga Anda bisa mengikat dan memberi referensi.

Cukup, modul adalah file yang terdiri dari kode Python. Modul dapat mendefinisikan fungsi, kelas dan variabel. Modul juga bisa menyertakan kode runnable.

Example

Kode Python untuk sebuah modul bernama aname biasanya berada pada sebuah file bernama aname.py. Berikut adalah contoh modul sederhana, support.py

```
def print _func( par ):
   print "Hello : ", par
   return
```

TheimportStatement

Anda dapat menggunakan file sumber Python apapun sebagai modul dengan mengeksekusi pernyataan impor di file sumber Python lainnya. Impor memiliki sintaks berikut:

```
import module1[, module2[,... moduleN]
```

Ketika penafsir menemukan sebuah pernyataan impor, ia mengimpor modul jika modul tersebut ada di jalur pencarian. Jalur pencarian adalah daftar direktori yang ditafsirkan juru bahasa sebelum mengimpor modul. Misalnya, untuk mengimpor modul support.py, Anda perlu meletakkan perintah berikut di bagian atas skrip -

```
#!/usr/bin/python

# Import module support
import support

# Now you can call defined function that module as follows
support.print _func("Zara")
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Hello : Zara
```

Modul hanya dimuat satu kali, berapa pun jumlahnya diimpor. Hal ini mencegah eksekusi modul terjadi berulang-ulang jika terjadi beberapa impor.

Thefrom...importStatement

Python dari pernyataan memungkinkan Anda mengimpor atribut tertentu dari modul ke dalam namespace saat ini. Dari ... impor memiliki sintaks berikut -

```
from modname import name1[, name2[, ... nameN]]
```

Misalnya, untuk mengimpor fungsi fibonacci dari modul fib, gunakan pernyataan berikut -

```
from fib import fibonacci
```

Pernyataan ini tidak mengimpor seluruh modul fib ke dalam namespace saat ini; Itu hanya memperkenalkan item fibonacci dari modul fib ke dalam tabel simbol global modul pengimpor.

Thefrom...import *Statement:

Hal ini juga memungkinkan untuk mengimpor semua nama dari modul ke dalam namespace saat ini dengan menggunakan pernyataan impor berikut -

```
from modname import *
```

Ini menyediakan cara mudah untuk mengimpor semua item dari modul ke dalam namespace saat ini; Namun, pernyataan ini harus digunakan dengan hemat.

Locating Modules

Saat mengimpor modul, juru bahasa Python mencari modul dalam urutan berikut -

- **Direktori saat ini.**
- **Jika modul tidak ditemukan, Python kemudian mencari setiap direktori di variabel shell PYTHONPATH.**
- **Jika semuanya gagal, Python memeriksa jalur default. Di UNIX, jalur default ini biasanya / usr / local / lib / python /.**

Jalur pencarian modul disimpan dalam sistem modul sys sebagai sys.pathvariable. Variabel sys.path berisi direktori saat ini, PYTHONPATH, dan default pengaturan instalasi.

ThePYTHONPATHVariable:

The PYTHONPATH adalah variabel lingkungan, yang terdiri dari daftar direktori. Sintaksis PYTHONPATH sama dengan variabel shell PATH.

Berikut adalah PYTHONPATH khas dari sistem Windows:

```
set PYTHONPATH=c: \python20 \lib;
```

Dan berikut ini adalah PYTHONPATH khas dari sistem UNIX:

```
set PYTHONPATH=/usr/local/lib/python
```

Namespaces and Scoping

Variabel adalah nama (identifiers) yang memetakan ke objek. Namespace adalah kamus dari nama variabel (kunci) dan objek yang sesuai (nilai).

Pernyataan Python dapat mengakses variabel dalam namespace lokal dan dalam namespace global. Jika variabel lokal dan global memiliki nama yang sama, variabel lokal membayangi variabel global. Setiap fungsi memiliki namespace lokal sendiri. Metode kelas mengikuti aturan pelingkupan yang sama dengan fungsi biasa.

Python membuat tebakan terdidik tentang apakah variabel bersifat lokal atau global. Ini mengasumsikan bahwa setiap variabel yang diberi nilai dalam suatu fungsi adalah lokal.

Oleh karena itu, untuk menetapkan nilai pada variabel global dalam suatu fungsi, Anda harus terlebih dahulu menggunakan pernyataan global.

Pernyataan global VarName memberitahu Python bahwa VarName adalah variabel global. Python berhenti mencari namespace lokal untuk variabel tersebut.

Sebagai contoh, kita mendefinisikan sebuah variabel Money in the global namespace. Dalam fungsi Money, kita menetapkan Money a value, oleh karena itu Python mengasumsikan variabel lokal Moneyas. Namun, kami mengakses nilai variabel lokal Moneybefore yang mengaturnya, jadi UnboundLocalError adalah hasilnya. Uncommenting pernyataan global memperbaiki masalahnya.

```
#!/usr/bin/python

Money = 2000
def AddMoney():
  # Uncomment the following line to fix the code:
  # global Money
  Money = Money + 1

print Money
AddMoney()
print Money
```

The dir( ) Function

Fungsi dir () built-in mengembalikan daftar string yang diurutkan yang berisi nama yang ditentukan oleh sebuah modul.

Daftar berisi nama semua modul, variabel dan fungsi yang didefinisikan dalam modul. Berikut adalah contoh sederhana -

```
#!/usr/bin/python

# Import built-in module math
import math

content = dir(math)

print content
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
[' _ _doc _ _', ' _ _file _ _', ' _ _name _ _', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

Di sini, variabel string khusus $_{n}ame_{a}dalahnamamodul,dan_{f}ile_{a}dalahnamafiletempatmoduldimuat.$

Theglobals()andlocals()Functions

Fungsi global () dan penduduk lokal () dapat digunakan untuk mengembalikan nama di ruang nama global dan lokal tergantung pada lokasi dari tempat mereka dipanggil.

Jika penduduk setempat () dipanggil dari dalam sebuah fungsi, maka semua nama yang dapat diakses secara lokal berasal dari fungsi itu.

Jika globals () dipanggil dari dalam sebuah fungsi, maka akan mengembalikan semua nama yang dapat diakses secara global dari fungsi itu.

Jenis kembalian dari kedua fungsi ini adalah kamus. Oleh karena itu, nama dapat diekstrak dengan menggunakan tombol () fungsi.

Thereload()Function

Saat modul diimpor ke dalam skrip, kode di bagian tingkat atas dari modul hanya akan dijalankan satu kali.

Oleh karena itu, jika Anda ingin mengulang ulang kode tingkat atas dalam modul, Anda dapat menggunakan fungsi reload (). Fungsi reload () mengimpor modul yang sebelumnya diimpor lagi. Sintaks fungsi reload () adalah ini -

```
reload(module _name)
```

Di sini, module _name adalah nama modul yang ingin Anda muat ulang dan bukan string yang berisi nama modul. Misalnya, untuk me-reload modul hello, lakukan hal berikut -

```
reload(hello)
```

Packages in Python

Paket adalah struktur direktori file hirarkis yang mendefinisikan satu lingkungan aplikasi Python yang terdiri dari modul dan subpackages dan sub-subpackages, dan seterusnya.

Pertimbangkan sebuah file Pots.py yang tersedia di direktori Phone. File ini memiliki baris kode sumber berikut -

```
#!/usr/bin/python

def Pots():
   print "I'm Pots Phone"
```

Cara yang sama, kita memiliki dua file lain yang memiliki fungsi berbeda dengan nama yang sama seperti di atas -

- *Phone/Isdn.py* file having function Isdn()

Phone/G3.py file having function G3()

Now, create one more file _ _init _ _.py in *Phone* directory −

- Phone/ _ _init _ _.py

Untuk membuat semua fungsi Anda tersedia saat mengimpor Telepon, Anda harus memasukkan pernyataan impor secara eksplisit di _ _init _ _.py sebagai berikut -

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

Setelah Anda menambahkan baris ini ke _ _init _ _.py, Anda memiliki semua kelas ini yang tersedia saat Anda mengimpor paket Telepon.

```
#!/usr/bin/python

# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

Pada contoh di atas, kita telah mengambil contoh fungsi tunggal di setiap file, namun Anda dapat menyimpan beberapa fungsi dalam file Anda. Anda juga dapat menentukan kelas Python yang berbeda dalam file tersebut dan kemudian Anda dapat membuat paket Anda dari kelas tersebut.

Modules

Jika Anda berhenti dari juru bahasa Python dan memasukkannya lagi, definisi yang telah Anda buat (fungsi dan variabel) hilang. Oleh karena itu, jika Anda ingin menulis program yang agak lama, sebaiknya Anda menggunakan editor teks untuk menyiapkan masukan bagi penerjemah dan menjalankannya dengan file itu sebagai masukan. Ini dikenal dengan membuat skrip. Seiring program Anda semakin lama, Anda mungkin ingin membaginya menjadi beberapa file untuk memudahkan perawatan. Anda mungkin juga ingin menggunakan fungsi praktis yang telah Anda tulis di beberapa program tanpa menyalin definisinya ke dalam setiap program.

Untuk mendukung ini, Python memiliki cara untuk menempatkan definisi dalam file dan menggunakannya dalam naskah atau dalam contoh juru bahasa interaktif. File seperti itu disebut modul; definisi dari modul dapat diimpor ke modul lain atau masuk ke modul utama (kumpulan variabel yang Anda akses ke dalam naskah yang dieksekusi di tingkat atas dan dalam mode kalkulator).

Modul adalah file yang berisi definisi dan pernyataan Python. Nama file adalah nama modul dengan akhiran .py ditambahkan. Dalam sebuah modul, nama modul (sebagai string) tersedia sebagai nilai dari variabel global _ _name _ _. Misalnya, gunakan editor teks favorit Anda untuk membuat file bernama fibo.py di direktori saat ini dengan konten berikut.

6.1. More on Modules

Modul dapat berisi pernyataan eksekusi serta definisi fungsi. Pernyataan ini dimaksudkan untuk menginisialisasi modul. Mereka dieksekusi hanya untuk pertama kalinya nama modul ditemukan dalam sebuah pernyataan impor. [1] (Mereka juga dijalankan jika file dijalankan sebagai skrip.)
Setiap modul memiliki tabel simbol pribadinya, yang digunakan sebagai tabel simbol global oleh semua fungsi yang didefinisikan dalam modul. Dengan demikian, penulis modul dapat menggunakan variabel global dalam modul tanpa mengkhawatirkan bentrokan kecelakaan dengan variabel global pengguna. Di sisi lain, jika Anda tahu apa yang Anda lakukan, Anda dapat menyentuh variabel global modul dengan notasi yang sama yang digunakan untuk merujuk pada fungsinya, nama modname.itemname.
Modul bisa mengimpor modul lainnya. Sudah menjadi kebiasaan tapi tidak diharuskan untuk menempatkan semua pernyataan impor di awal modul (atau naskah, dalam hal ini). Nama modul yang diimpor ditempatkan di tabel simbol global modul pengimpor.

6.1.1. Executing modules as scripts

Saat Anda menjalankan modul Python dengan
python fibo.py <arguments>
Kode dalam modul akan dieksekusi, sama seperti jika Anda mengimpornya, tapi dengan _ _name _ _ set ke " _ _main _ _". Itu berarti bahwa dengan menambahkan kode ini di akhir modul Anda:

```
if _ _name _ _ == " _ _main _ _":
    import sys
    fib(int(sys.argv[1]))
```

Anda dapat membuat file tersebut dapat digunakan sebagai skrip dan juga modul yang dapat diimpor, karena kode yang mem-parsing baris perintah hanya berjalan jika modul dijalankan sebagai file "utama":

```
 $ python fibo.py 50
1 1 2 3 5 8 13 21 34
```

Jika modul diimpor, kode tidak dijalankan:

```
»>
»> import fibo
»>
```

Hal ini sering digunakan untuk menyediakan antarmuka pengguna yang mudah digunakan ke modul, atau untuk tujuan pengujian (menjalankan modul saat skrip menjalankan test suite).

6.1.2. The Module Search Path

Ketika sebuah modul bernama spam diimpor, penerjemah pertama-tama mencari modul built-in dengan nama itu. Jika tidak ditemukan, maka cari file bernama spam.py dalam daftar direktori yang diberikan oleh variabel sys.path. sys.path diinisialisasi dari lokasi ini:
• Direktori berisi skrip masukan (atau direktori saat ini bila tidak ada file yang ditentukan).
• PYTHONPATH (daftar nama direktori, dengan sintaks yang sama dengan variabel shell PATH).
• Default yang tergantung pada instalasi.
Setelah inisialisasi, program Python dapat memodifikasi sys.path. Direktori yang berisi skrip yang dijalankan ditempatkan di awal jalur pencarian, di depan jalur perpustakaan standar. Ini berarti skrip di direktori itu akan dimuat alih-alih modul dengan nama yang sama di direktori perpustakaan. Ini adalah kesalahan kecuali penggantinya. Lihat bagian Modul Standar untuk informasi lebih lanjut.

6.1.3. "Compiled" Python files

Untuk mempercepat pemuatan modul, Python menyimpan versi yang dikompilasi setiap modul di direktori _ _pycache _ _ dengan nama module.version.pyc, di mana versi tersebut mengkodekan format file yang dikompilasi; Umumnya berisi nomor versi Python. Sebagai contoh, dalam rilis CPython 3.3 versi terkompilasi dari spam.py akan di-cache sebagai _ _pycache _ _ / spam.cpython-33.pyc. Konvensi penamaan ini memungkinkan modul yang dikompilasi dari berbagai rilis dan versi Python yang berbeda untuk hidup berdampingan. Python memeriksa tanggal modifikasi sumber dari versi yang dikompilasi untuk melihat apakah sudah ketinggalan zaman dan perlu dikompilasi ulang. Ini adalah proses yang benar-benar otomatis. Selain itu, modul yang dikompilasi bersifat platform-independen, sehingga perpustakaan yang sama dapat dibagi antar sistem dengan arsitektur yang berbeda. Python tidak memeriksa cache dalam dua situasi. Pertama, selalu mengkompilasi ulang dan tidak menyimpan hasilnya untuk modul yang dimuat langsung dari baris perintah. Kedua, tidak memeriksa cache jika tidak ada modul sumber. Untuk mendukung distribusi non-source (dikompilasi saja), modul yang dikompilasi harus berada dalam direktori sumber, dan tidak boleh ada modul sumber.
Beberapa tip untuk para ahli:
• Anda dapat menggunakan switch -O atau -OO pada perintah Python untuk mengurangi ukuran modul yang dikompilasi. Sakelar -O menghapus pernyataan tegas, tombol -OO menghapus kedua pernyataan tegas dan string _ _doc _ _. Karena beberapa program mungkin bergantung pada ketersediaan ini, Anda sebaiknya hanya menggunakan opsi ini jika Anda tahu apa yang Anda lakukan. Modul "Dioptimalkan" memiliki pilihan dan biasanya lebih kecil. Rilis masa depan dapat mengubah efek pengoptimalan.
• Program tidak berjalan lebih cepat saat dibaca dari file .pyc daripada saat dibaca dari file .py; Satu-satunya yang lebih cepat tentang file .pyc adalah kecepatan pengisiannya.
• Compileall modul dapat membuat file .pyc untuk semua modul dalam sebuah direktori.
• Ada lebih banyak rincian mengenai proses ini, termasuk bagan alir keputusan, dalam PEP 3147.