8 — Moduli i paketi

8.1 Modul

U modul je moguće uključiti bilo koju valjanu datoteku s Python kôdom, te pozvati ga s naredbom `import`. Na primjer, neka je sljedeći kôd spremljen u datoteku '`primjer.py`':

```
# datoteka: primjer.py
a = 37                      # Varijabla
def A:                      # Funkcija
    print "Ja sam u A"
class K:                    # Klasa
    def B(self):
        print "Ja sam u K.B"
b = K()                     # Tvorba instance
```

Pozvati u memoriju ovakav kôd kao modul, postiže se naredbom `import primjer`. Prvi put kad se `import` koristi da se modul učita u memoriju, događaju se tri stvari:

1. Modul stvara novi prostor imena koji služi svim objektima definiranim u pripadnoj izvornoj datoteci. Ovaj prostor imena dohvaća se ako funkcije i metode definirane unutar modula koriste naredbu `global`.
2. Modul izvršava kôd koji je sadržan unutar novonastalog prostora imena.
3. Modul izvršava ime unutar pozivatelja, koje se odnosi na prostor imena modula. Ovo ime podudara se s imenom modula.

```
import primjer          # Učitava u memoriju i izvodi modul 'primjer'
print primjer.a         # Dohvaća izvodi neki član (varijablu) modula 'primjer'
primjer.A()             # Dohvaća i izvodi funkciju iz modula
c = primjer.K()         # Dohvaća i izvodi klasu iz modula, tvorba objekta
...
```

Učitavanje, uvoz (eng. *import*) višestrukih modula izvodi se tako da se iza naredbe `import` dopišu imena modula, odijeljena zarezom, na primjer ovako:

```
import socket, os, regex
```

Moduli se mogu učitati (*importirati*) koristeći alternativna imena, i to upotrebom poveznice *as*. Na primjer:

```
import os as sustav
import socket as mreza, thread as nit
sustav.chdir("..")
mreza.gethostname()
```

Naredba `from` koristi se za specifične definicije unutar modula s trenutačnim prostoru imena. Ova naredba predstavlja proširenje naredbe `import` gdje se uz novo stvoreni prostora imena, stvara i referenca na jedan ili više objekata definiranih untar modula:

```
from socket import gethostname
# Stavlja gethostname u trenutačni prostor imena

print gethostname()            # Koristi se bez imena modula
socket.gethostname()           # Pogreška imena (NameError: socket)
```

Naredba `from` također prihvaća zarezom odvojena imena objekata. Zvjezdica (engl. *asterisk* , *) je sveobuhvatni (engl. *wildcard*) znak koji se koristi za učitanje svih definicija u modulu, osim onih koje počinju s podvučenom crtom (_). Na primjer:

```
from socket import gethostname, socket
from socket import *                    # Puni sve definicije u trenutačni prostor imena
```

Moduli se mogu preciznije upravljati skupom imena koji se učita s `from module import * definiranjem liste __all__`. Na primjer:

```
# module: primjer.py
__all__ = [ 'K', 'primjer' ]     # To su sva ona imena koja će se uvući sa *
```

Nadalje, poveznica `as` može se koristiti za promjenu imena objekata koji se učitaju s naredbom `from`. Na primjer:

```
from socket import gethostname as ime_hosta
h = ime_hosta()
```

Naredba `import` može se pojaviti na bilo kojem mjestu u programu. Međutim, kôd u svakom modulu izvodi se samo jednom, bez obzira kako često se naredba `import` izvršava. Iduće `import` naredbe jednostavno stvaraju referencu na prostor imena modula koji je stvoren s početnom `import` naredbom. U varijabli `sys.modules` može se naći rječnik koji sadrži imena svih trenutačno napunjenih modula. On preslikavaa imena modula na njihove objekte. Sadržaj rječnika se koristi za određivanje je li `import` napunio svježu kopiju modula ili se isti modul poziva drugi put.

Naredba `from module import *` može se koristiti samo na vrhu modula. Drugim riječima, nije dopušteno koristiti ovaj oblik `import` naredbe unutar tijela funkcije, zbog njegove interakcije s pravilima funkcijskog dosega (engl. *scoping rules*). Svaki modul definira variablu `__name__` koja sadrži ime modula. Programi mogu ispitivati ovu varijablu i pritom odrediti modul u kojem se programske naredbe izvršavaju. Najviši (tzv. top-level) modul interpretera zove se `__main__`. Programi zadani na naredbenoj liniji ili unešeni interaktivno, uvijek se izvode unutar tog `__main__` modula. Ponekad program može promijeniti ovo ponašanje, ovisno o tomu je li uvučen iz modula ili je izveden u `__main__` okolišu. To se može postići na ovaj način:

```
# provjera je li se vrti kao program
if __name__ == '__main__':
      # Da
      Naredbe
else:
      # ne, uvucen je kao modul
      Naredbe
```

8.1.1 Traženje modula

Kad se moduli učitavaju, interpreter traži listu imenika, foldera u `sys.path`. Tipična vrijednost `sys.path` može izgledati ovako:

```
['', '/usr/local/lib/python2.5',
     '/usr/local/lib/python2.5/lib-tk',
     '/usr/local/lib/python2.5/lib-dynload',
     '/usr/local/lib/python2.5/site-packages']
```

Prazan string '' odnosi se na trenutačni imenik, folder. Novi imenici dodaju se u put traženja vrlo jednostavno - dodavanjem člana (stringa puta) u ovu listu.

8.1.2 Učitavanje modula i compilacija

Do sada su moduli prikazani kao datoteke koje sadrže Python kôd. Međutim, moduli učitani s naredbom `import` mogu pripadati nekoj od četiri opće kategorije:

- Programi pisani u Pythonu (.py datoteke)
- C ili C++ proširenja koja su compilirana u zajedničkim (shared) knjižnicama ili DLL-ovima.
- Paketi koji sadrže skupove modula
- Ugrađeni moduli pisane u C-u i povezani s Python interpreterom

Kad se na primjer, traži modul `primjer`, interpreter pretražuje svaki od direktorija u `sys.path` za sljedeće datoteke (navedene po redoslijedu pretraživanja):

1. Imenik ili folder `primjer` koji je definiran u paketu
2. `primjer.so`, `primjermodule.so`, `primjermodule.sl`, ili `primjermodule.dll` (compilirana proširenja)
3. `primjer.pyo` (samo ako je `-O` ili `-OO` opcija korištena)
4. `primjer.pyc`
5. `primjer.py`

Za `.py` datoteke, kad se modul prvi put učita, on se prevede, compilira u međukôd (eng. *bytecode*) i zapisuje na disk u datoteku s proširenjem `.pyc`. U idućim učitanjima, interpreter puni ovaj prirađeni međukôd, osim ako ne dođe do promjene originalne `.py` datoteke (pa se `.pyc` datoteka mora regenerirati. Datoteke koje završavaju na `.pyo` koriste se u svezi s interpreterskom `-O` opcijom. Ove datoteke sadrže međukôd s izbačenim brojevima linija i drugim informacijama potrebnim za traženje pogrešaka (eng. *debugging*). Kao rezultat toga, kôd je nešto manji i dopušta interpretoru nešto brže izvođenje. U slučaju `-OO` opcije svi dokumentacijski stringovi se također izbacuju iz datoteka. Ovo brisanje dokumentacijskih stringova pojavljuje se samo kad se `.pyo` datoteke stvaraju, a ne kad se pune. Ako niti jedna od ovih datoteka ne postoji u niti jednom imeniku (folderu) u `sys.path`, onda interpreter provjerava da li ime odgovara imenu nekog ugrađenog modula. Ako ni ono ne postoji, onda se podiže `ImportError` iznimka.

Compilacija datoteka u `.pyc` i `.pyo` datoteke događa se samo u sprezi s naredbom `import`. Programi izvedeni preko naredbene linije ili preko standardnog ulaza ne proizvode takve datoteke.

Naredba `import` u traženju datoteka razlikuje mala i velika slova u njihovom imenu. Kod operacijskih sustava koji ne podržavaju tu razliku, potrebno je o tomu voditi računa.

8.1.3 Ponovno punjenje modula

Ugrađena funkcija reload() koristi se za ponovno punjenje i izvršavanje kôda sadržanog unutar modula koji je prethodno učitan s naredbom `import`. Ona prima objekt modula kao pojedinačni argument. Na primjer:

```
import primjer
... neki kôd ...
reload(primjer)          # Ponovno puni modul 'primjer'
```

Sve operacije uključene u modul će se nakon izvođenja naredbe `reload()` izvesti iz novo učitanog kôda. Međutim, `reload()` neće retroaktivno obnoviti objekte koji su nastali iz starog modula. To znači da je moguće istodobno postojanje referenci na objekte i u staroj i u novoj verziji modula. Nadalje, compilirane ekstenzije pisane u C ili C++ ne mogu se ponovono puniti naredbom `reload()`. Kao opće pravilo, treba izbjegavati ponovno punjenje modula, osim za vrijeme razvitka programa i traženja pogreški u njemu.

8.2 Paketi

Paketi dopuštaju da se više modula skupi zajedno pod jednim zajedničkim imenom. Ova tehnika pomaže u razlučivanju sukoba u prostoru imena modula koji se koriste u različitim primjenama. Paket se definira stvaranjem imenika, foldera (engl. *directory, folder*) s istim imenom kao paket i stvaranjem `__init__.py` datoteke u tom imeniku. Moguće je nakon toga dodavati druge izvorne datoteke (tekstovne datoteke s Python kôdom), compilirana proširenja ili podpakete u istom imeniku. Na primjer, paket se može ovako organizirati:

```
Graphics/
      __init__.py
      Primitive/
         __init__.py
         linije.py
         ispuna.py
         tekstovi.py
         ...
      Graph2d/
         __init__.py
         plot2d.py
         ...
      Graph3d/
         __init__.py
         plot3d.py
         ...
      Formati/
         __init__.py
         gif.py
         png.py
         tiff.py
         jpeg.py
```

Naredba `import` koristi se za punjenje modula iz paketa na nekoliko načina:

```
import Graphics.Primitive.ispuna
```

Ovo učitava podmodul `Graphics.Primitive.ispuna`. Sadržaj tog modula mora se eksplicitno imenovati, kao na primjer: `Graphics.Primitive.ispune.crtaj(img,x,y,color)`.

Drugi način:

```
from Graphics.Primitive import ispuna
```

učitava podmodul `ispuna` i čini ga raspoloživim i bez paketnog prefiksa, na primjer: `ispuna.crtaj(img,x,y,color)`.

Treći način:

```
from Graphics.Primitive.ispuna import crtaj
```

Ova naredba učitava podmodul `ispuna` i čini `crtaj` funkciju direktno primjenljivom, primjerice: `crtaj(img,x,y,color)`. Kad god se neki dio paketa učita, importira, kôd u datoteci `__init__.py` se tad izvrši. U najmanjem slučaju ova datoteka može biti i prazna, ali također može sadržavati kôd kojim se izvode inicijalizacije koje su specifične za paket. Sve nađene datoteke u `__init__.py` za vrijeme učitavanja, automatski se izvršavaju. Tako će naredba `import Graphics.Primitive.ispuna` izvršiti `__init__.py` datoteke i u `Graphics` imeniku i u `Primitive` imeniku.

Jedan poseban problem s paketima je obradba ovakve naredbe:

```
from Graphics.Primitive import *
```

Željeni cilj ove naredbe je učitati sve module pridružene u paketu s trenutačnim prostorom imena. Međutim, s obzirom da dogovori oko imena datoteka (pogotovo u smislu razlikovanja velikih i malih slova) variraju od sustava do sustava, Python ne može točno odrediti koje module točno treba uključiti. Kao rezultat, ova naredba samo učitava sve reference koje su definirane u `__init__.py` datoteci u `Primitive` imeniku. Ovo se ponašanje može promijeniti definiranjem liste `__all__` koja sadrži sva imena modula pridružena s paketom.

Takva lista treba biti definirana unutar paketa u `__init__.py` datoteci, kao na primjer:

```
# Graphics/Primitive/__init__.py
__all__ = ["linije", "tekstovi", "ispuna",...]
```

Ako korisnik pokrene naredbu `from Graphics.Primitive import *` onda se svi navedeni podmoduli učitaju, kako se i očekuje. Učitavanje samo imena paketa neće automatski učitati sve podmodule koji se nalaze u paketu. Na primjer, sljedeći kôd neće raditi:

```
import Graphics
Graphics.Primitive.fill.floodfill(img,x,y,color)  # Pogrešno!
```

Međutim, budući da `importGraphics` naredba izvršava `__init__.py` datoteku u Graphics imeniku, moguće ju je promijeniti tako da se automatski učitaju svi podmoduli:

```
# Graphics/__init__.py
import Primitive, Graph2d, Graph3d

# Graphics/Primitive/__init__.py
import linije, tekstovi, ispuna ...
```

Na ovaj način `importGraphics` naredba učitava sve podmodule i čini ih raspoloživima za pozive koristeći njihova potpuna imena. Moduli koji su sadržani unutar istog imenika (foldera) paketa, mogu referencirati jedan drugog bez navođenja punog imena paketa. Na primjer, modul `Graphics.Primitive.ispuna` može učitati `Graphics.Primitive.linije` `modul` jednostavno koristeći naredbu `import linije`. Međutim, ako je modul smješten u drugom imeniku, onda se mora koristiti puno ime paketa. Na primjer, ako `plot2d` iz modula `Graphics.Graph2d` traži upotrebu `linije` iz modula `Graphics.Primitive`, mora se koristiti puna staza imena: `from Graphics.Primitive import linije`. Ako je potrebno, modul može ispitati svoju `__name__` varijablu kako bi našao potpunu stazu svog imena. Na primjer, sljedeći kôd učitava modul iz sestrinskog podpaketa znajući samo ime tog podpaketa (a ne i najviše ime s vrha paketa):

```
# Graphics/Graph2d/plot2d.py
# Određuje ime paketa, gdje je smješten moj paket
import string
base_package = string.join(string.split(__name__,'.')[:-2],'.')

# Import the ../Primitive/ispuna.py module
exec "from %s.Primitive import ispuna" % (base_package,)
```

Konačno, kad Python učitava paket, on definira posebnu varjablu `__path__` koja sadrži listu imenika koji se pretražuju, kad se se traže paketni podmoduli. (Varijabla `__path__` je posebna inačica `sys.path` varijable.) `__path__` je dohvatljiv za kôd sadržan u datoteci `__init__.py` i inicijalno sadrži jedan član s imenom foldera (imenika) paketa. Ako je potrebno, paket može dodati dodatne imenike u `__path__` listu da bi se promijenila staza koja se koristi u traženju podmodula.