Programarea calculatoarelor si limbaje de programare II

Organizarea modulelor in pachete

Universitatea Politehnica din Bucureşti

Sumar

- ***Importarea pachetelor***
- Exemplu de import de pachet
- Utilizarea pachetelor
- Importari relative
- Pachete de spatii de nume (v3.3+)

Sintaxa importului de pachete

- Un director (de pe disc) poate reprezenta un **pachet** in Python, ce are asociat un spatiu de nume cu atribute care corespund subdirectoarelor si fisierelor modul continute
- Sintaxa – importarea se poate face specificand o lista de nume de subdirectoare, separate cu punct (in loc de un nume simplu de fisier):

```
import dir1.dir2.mod

from dir1.dir2.mod import x
```

- *mod.py* (sau alta extensie) este un fisier din directorul *dir2*, continut in directorul *dir1* – care este fie intr-unul din directoarele din *sys.path* sau intr-un director in care se afla scriptul principal

Calea de cautare a pachetelor

- Importarea **nu** se poate face folosind sintaxa numelor de fisiere specifica sistemului de operare: ~~C:\dir1, My Documents.dir2, ../dir1~~. In *sys.path*, da!

  import C:\mycode\dir1\dir2\mod # Sintaxa ilegala!

- Adaugand *C:\mycode* la PYTHONPATH sau intr-un fisier cu extensia *.pth*, atunci este corect:

  import dir1.dir2.mod

- Sintaxa cu puncte intre subdirectoare este independenta de platforma
- Prefixul caii trebuie sa se gaseasca in *sys.path*:
  - directorul curent (al scriptului principal sau chiar directorul sesiunii interactive), director de biblioteca standard, sau directorul Lib\site-packages

Pachete cu fisierul *__init__.py*

- Pana inainte de v3.3, directoarele din calea importului trebuie sa contina un fisier numit **__init__.py**
- **import dir1.dir2.mod** este corect cu urmatoarea structura de fisiere:

```
dir0
└── dir1
    ├── __init__.py
    └── dir2
        ├── __init__.py
        └── mod.py
```

- *dir0* trebuie sa fie listat in *sys.path*
- Numai *dir1* si *dir2* au nevoie de *__init__.py*

Rolul fisierului de initializare

- Fisierul *__init__.py* (poate) contine cod Python care este executat in mod automat cand directorul este importat
  - Astfel se face intializarea unui pachet e.g. creare de fisiere de date, acces la o baza de date
  - Nu sunt menite a fi executate direct, ci la primul acces al pachetului
- Declara directorul ce-l contine ca fiind un pachet Python (se elimina confuziile)
- Spatiul de nume asociat subdirectoarelor, e.g. *dir2*, creat dupa import, va contine toate numele asignate in fisierul *dir1\dir2__init__.py*, plasate in obiectul modul accesat cu expresia *dir1.dir2*

Rolul...

- Instructiunea *import* * **nu** incarca submodule daca fisierul *__init__.py* nu contine variabila **__all__** de tip *list*, in care se enumera numele exportate automat cu *import* *
- Fisierele *__init__.py* pot fi vide, dar trebuie sa existe (inainte de v3.3)

Sumar

- Importarea pachetelor
- ***Exemplu de import de pachet***
- Utilizarea pachetelor
- Importari relative
- Pachete de spatii de nume (v3.3+)

Exemplu

- Continutul directoarelor *dir1* si *dir2*:

```
# Fisierul dir1\__init__.py
print('dir1 init')
x = 1
# Fisierul dir1\dir2\__init__.py
print('dir2 init')
y = 2
# Fisierul dir1\dir2\mod.py
print('in mod.py')
z = 3
```

- Importarea:

```
C:\code> python # In directorul ce contine dir1
>>> import dir1.dir2.mod # Primul import executa fisierele __init__.py
dir1 init
dir2 init
in mod.py
>>> import dir1.dir2.mod # Importari ulterioare, nu!
>>>
```

Exemplu...

- Reincarcarea cu *reload*() a directoarelor deja importate:

```
>>> from importlib import reload
>>> reload(dir1)
dir1 init
<module 'dir1' from
    'C:\\Users\\Dan\\Desktop\\code\\dir1\\__init__.py'>
>>> reload(dir1.dir2)
dir2 init
<module 'dir1.dir2' from
    'C:\\Users\\Dan\\Desktop\\code\\dir1\\dir2\\__init__.py'>
```

- Calea dintr-un *import* devine o selectie/calificare de obiecte (incluse):

```
>>> dir1
<module 'dir1' from
    'C:\\Users\\Dan\\Desktop\\code\\dir1\\__init__.py'>
>>> dir1.dir2
<module 'dir1.dir2' from
    'C:\\Users\\Dan\\Desktop\\code\\dir1\\dir2\\__init__.py'>
>>> dir1.dir2.mod
<module 'dir1.dir2.mod' from
    'C:\\Users\\Dan\\Desktop\\code\\dir1\\dir2\\mod.py'>
```

Exemplu...

- Numele asignate in *__init__.py* fac parte din spatiul de nume al obiectelor modul respective:

```
>>> dir1.x
1
>>> dir1.dir2.y
2
>>> dir1.dir2.mod.z
3
```

from vs. *import* de pachete

- *import* este mai dificil de folosit cu pachete, din cauza cailor lungi:

```
>>> dir2.mod # Erori daca caile nu sunt incluse
NameError: name 'dir2' is not defined
>>> mod.z
NameError: name 'mod' is not defined
```

- *from* este mai convenabil; iar modificarea structurii pachetului necesita doar o actualizare de cale cu *from* (multe la *import)*:

```
>>> from dir1.dir2 import mod # Calea doar aici, in from
dir1 init
dir2 init
in mod.py
>>> mod.z # Fara repetitii de cale
3
>>> from dir1.dir2.mod import z
>>> z
3
>>> import dir1.dir2.mod as mod # import cu as permite nume mai scurte
>>> mod.z
3
>>> from dir1.dir2.mod import z as modz # Se evita conflictul de nume
>>> modz
3
```

Sumar

- Importarea pachetelor
- Exemplu de import de pachet
- ***Utilizarea pachetelor***
- Importari relative
- Pachete de spatii de nume (v3.3+)

Avantajele pachetelor

- Importarea de pachete ofera informatii despre directoare:
  - localizarea fisierelor este mai simpla
  - fara caile pachetelor, trebuie consultata *sys.path*
- Codul este mai usor de inteles:

  import utilities # *Vag*

  import database.client.utilities # *Mai clar*
- PYTHONPATH si fisierele cu extensia *.pth* sunt mai usor de gestionat daca se folosesc importari explicite cu cai relative la un singur director radacina – de adaugat la *sys.path*

Rezolvarea ambiguitatilor

- Cazul modulelor cu aceleasi nume din directoare diferite, system1 si system2 – utilities.py, main.py, other.py:
  - Executia scriptului principal asigura cautarea cu precedenta in directorul care il contine – OK
  - Reutilizarea codului – cu nume comun in mai multe directoare, este problematica:
    - Schimbarea PYTHONPATH sau manipularea lui *sys.path* sunt dificile si pot conduce la erori, respectiv.
    - Solutia: organizarea directoarelor ca pachete – cu adaugarea fisierelor *__init__.py* – intr-un director comun, plasat pe *sys.path*, si folosirea importului de pachete, cu cai distincte:

```
import system1.utilities
import system2.utilities
system1.utilities.function('spam')
system2.utilities.function('eggs')
```

Rezolvarea...

- De notat faptul ca **nu** este necesara schimbarea *import*-urilor din system1 si system2 (dar **da** in v3.x...)

```
root
├── system1
│   ├── __init__.py
│   ├── main.py
│   ├── other.py
│   └── utilities.py
├── system2
│   ├── __init__.py
│   ├── main.py
│   ├── other.py
│   └── utilities.py
└── system3
    ├── __init__.py
    └── myfile.py
```

Si system3, care foloseste utilities.py, poate fi transformat intr-un pachet, spre viitoare folosinta

Sumar

- Importarea pachetelor
- Exemplu de import de pachet
- Utilizarea pachetelor
- ***Importari relative***
- Pachete de spatii de nume (v3.3+)

Importarea in Python v3.x

- **Importarea din interiorul unui pachet**:
  - poate folosi aceeasi sintaxa ca pentru importari din exterior, cu cai complete
  - exista insa posibilitatea importarii relative la pachet
- In Python v2.x importarea cauta mai intai in directorul pachetului.
- In Python v3.x:
  - Cautarea incepe cu *sys.path*, iar directorul pachetului este sarit in mod implicit – asa zisa **importare absoluta**
  - Sintaxa instructiunii *from* este extinsa, spre a se cauta *numai* in directorul pachetului, cu punct/e la inceputul caii – asa zisa **importare relativa**

Importarea relativa

- Importarea cu ***from*** si puncte la inceputul caii: in v2.x si v3.x inseamna numai cautare relativa la acelasi pachet
- Fara puncte, in v2.x cautarea este intai relativa si apoi absoluta. In v3.x este numai absoluta – pe directoarele din *sys.path*
- Sintaxa – doar cu *from*, nu si *import*!:

```
from  .  import spam          # Se importa modulul spam din acelasi pachet
from .spam import name        # Se importa variabila name din modulul spam
                                aflat in acelasi pachet
```

- Exemple:

```
import string # modulul string din sys.path in v3.x
from . import string # modulul string din pachet, obligatoriu
```

Importarea...

- Alte exemple, dintr-un modul aflat in pachetul mypkg:

```
from .string import name1, name2 # Se importa name1 si name2 din modulul mypkg.string

from . import string # Se importa modulul mypkg.string

from .. import string # Se importa modulul string din pachetul parinte
```

Rolul importarilor relative

- Rezolva ambiguitatile de nume pentru fisiere denumite la fel si aflate in mai mai multe locuri pe calea de cautare:

```
mypkg
├── __init__.py
├── main.py
└── string.py
```

- In v2.x instructiunea *import string* din main.py va gasi intotdeauna fisierul mypkg\string.py, iar modulul standard *string* va fi ascuns.
- Evitarea folosirii numelor standard de module este posibila acum, dar in viitor?

Rolul...

- Importarile relative din Python v3.x:

import string *# Se importa absolut, din afara pachetului*

from string import name *# Se importa variabila name din modulul string din afara pachetului*

from . import string *# Se importa relativ, modulul mypkg.string*

from .string import name1, name2 *# Se importa name1 si name2 din modulul mypkg.string*

from .. import spam *# Se importa un modul "frate" cu mypkg*

- In general, cod din modulul **A.B.C** semnifica:

from . import D *# Se importa modulul A.B.D (. este A.B)*

from .. import E *# Se importa modulul A.E (.. este A)*

from .D import X *# Se importa var. A.B.D.X (. este A.B)*

from ..E import X *# Se importa var. A.E.X (.. este A)*

Rolul...

- Importari relative vs. importari absolute:

from mypkg import string *# Se importa mypkg.string, absolut, daca mypkg se afla intr-un director listat in sys.path*

from system.section.mypkg import string *# Cazul lui mypkg aflat in adancime, necesita listarea completa a caii*

from . import string *# Solutia simpla, indiferent de adancimea lui mypkg (import relativ)*

Aplicabilitatea importarilor relative

- Se aplica numai importarilor din interiorul pachetelor
  - Importarile din afara pachetelor cauta mai intai in directorul ce cuprinde scriptul principal, apoi directoarele din *sys.path*
- Se codifica numai cu instructiunea *from* (**nu** *import*)
  - Numele modulului incepe cu un punct sau cu doua puncte
    - Module ce contin puncte in nume, dar nu incep cu punct sunt importari de pachete, **nu** relative
- In esenta, in importarile intra-pachet din v3.x:
  - s-au eliminat cautarile relative la pachet din v2.x
  - s-a adaugat sintaxa extinsa a *from* pentru cautari numai relativ la pachet

Cautarea modulelor, 3.x, sumar

- Modulele cu nume simple (e.g. A) se cauta in fiecare director din *sys.path*
- Pachetele sunt directoare ce contin module si fisierul __init__.py:
  - ***import A.B.C*** refera (de obicei) un modul C aflat in pachetul/subdirectorul B care se afla in pachetul/subdirectorul A, iar A se afla intr-un director din *sys.path*
- Din interiorul unui pachet, *import* si *from* se comporta ca in exterior
  - extensia lui *from* cu punct/e face o cautare relativa la pachet, iar *sys.path* este ignorat
    - ***from . import A*** cauta doar in directorul care contine fisierul cu aceasta instructiune

Cautarea...

- Python 2.x se comporta la fel, insa cauta automat mai intai in directorul pachetului
- Sumar:
  - Importarile cu punct: ***from . import m*** sunt relative in v2.x si v3.x
  - Importarile fara punct: **import m, from m import x** sunt relative-si-apoi-absolute in v2.x si numai absolute in v3.x

Exemple de importari relative

- Importari din afara pachetelor:

```
C:\code>py -3
>>> import string
>>> string # Modulul standard
<module 'string' from 'C:\\Program Files\\Python37\\lib\\string.py'>
```

- Dar, cu modulul *code\string.py* (interior):

```
# Fisierul code\string.py
print('string' * 4 )
C:\code>python
>>> import string
stringstringstringstring
>>> string # CWD este primul in sys.path!
<module 'string' from 'C:\\Users\\Dan\\Desktop\\code\\string.py'>
```

- Importarea relativa nu este permisa decat cu pachet:

```
C:\code>py -3
>>> from . import string # Atributul __package__ este None !
ImportError: cannot import name 'string' from '__main__' (unknown location)
>>> print( __package__ )
None
>>>
```

Exemple...

- Cu executia unui script principal:

```
# Fisierul code\main.py
import string
print(string) # print este necesar intr-un script
    pentru afisare!
```

```
C:\code>rem La fel, CWD e primul in sys.path
C:\code>py -3 main.py
stringstringstringstring
<module 'string' from
    'C:\\Users\\Dan\\Desktop\\code\\string.py'
    >
```

- Importari din pachet, relative:

```
# Pachetul code\pkg, arbore:
pkg
 ├── __init__.py
 ├── eggs.py
 ├── main.py
 └── spam.py
```

```
# Fisierul code\pkg\spam.py
import eggs # <== Da in 2.X , nu in 3.X!
print(eggs.X)
# Fisierul code\pkg\eggs.py
X = 99999
import string
print(string)
```

Exemple...

```
C:\code>py -2
>>> import pkg.spam
<module 'string' from
    'C:\Python27\lib\string.pyc'>
99999
```

```
C:\code>py -3
>>> import pkg.spam
ModuleNotFoundError: No module named
    'eggs'
```

- Cu ***from . import eggs***, corect in v2.x si v3.x:

```
# Fisierul code\pkg\spam.py, modificat:
from . import eggs # Importare relativa, in v2.x
    si v3.x, corect
print(eggs.X)
C:\code>py -2
>>> import pkg.spam
<module 'string' from
    'C:\Python27\lib\string.pyc'>
99999
```

```
C:\code>py -3
>>> import pkg.spam
<module 'string' from 'C:\\Program
    Files\\Python37\\lib\\string.py'>
99999
```

Exemple...

- Daca exista code\string.py, are prioritate, fiind in CWD:

```
# Fisierul code\string.py
print('string' * 4)
# Restul, la fel, cu importare relativa
C:\code>py -3 sau -2
>>> import pkg.spam
```

```
stringstringstringstring
<module 'string' from
    'C:\\Users\\Dan\\Desktop\\code\\string.py'
    >
99999
```

- Fara *code\string.py*, dar **cu** *code\pkg\string.py*:

```
# Fisierul code\pkg\string.py:
print('Ni' * 4 )    # The Knights Who Say "Ni!" -
    Monty Python and the Holy Grail
# Fisierul code\pkg\spam.py:
import string # Relativ in v2.X, absolut in v3.X
print(string)
C:\code>py -3
>>> import pkg.spam
```

```
<module 'string' from 'C:\\Program
    Files\\Python37\\lib\\string.py'>
99999
C:\code>py -2
>>> import pkg.spam
NiNiNiNi
<module 'pkg.string' from 'pkg\string.py'>
99999
```

Exemple...

- In v3.x se forteaza cautarea relativa cu ***from .*** :

```
# Fisierul code\pkg\spam.py:
from . import string # <== Relativ in v2.X si in v3.X
print(string)
# Restul, la fel
```

```
C:\code>py -3
>>> import pkg.spam
NiNiNiNi
<module 'pkg.string' from 'C:\\Users\\Dan\\Desktop\\code\\pkg\\string.py'>
```

```
C:\code>py -2
>>> import pkg.spam
NiNiNiNi
<module 'pkg.string' from 'pkg\string.pyc'>
# Byte code, nu s-a mai compilat
```

- Fara *code\pkg\string.py (si fara byte code)*:

```
# Importarea relativa rateaza in v3.x si v2.x daca modulul nu se afla in interiorul pachetului!
C:\code>py -3
>>> import pkg.spam
ImportError: cannot import name 'string' from 'pkg'
```

```
(C:\Users\Dan\Desktop\code\pkg\__init__.py)
C:\code>py -2
>>> import pkg.spam
ImportError: cannot import name 'string'
```

Exemple...

- Importarile absolute sunt relative la CWD in v3.x:

```
# Cu string.py in code si code\pkg          C:\code>py -2
# Fisierul code\string.py:                  >>> import pkg.spam
print('string' * 4)                        NiNiNiNi
# Fisierul code\pkg\string.py               <module 'pkg.string' from 'pkg\string.py'>
print('Ni' * 4)                            C:\code>py -3
# Fisierul code\pkg\spam.py                 >>> import pkg.spam
import string # Relativ in 2.x,absolut in 3.x: CWD  stringstringstringstring
print(string)                               <module 'string' from 'C:\\...\\code\\string.py'>
```

- Pachetele pot cere module din propriul director cu **from .**
- Importarile absolute sunt relative la *sys.path*
- In v3.x se poate decide intre interior si exterior
  - Absolut poate fi de pe CWD, in loc de standard!

Importari care merg in v2.x si v3.x

- Importarile absolute de pachete (relative la *sys.path*) sunt preferabile atat fata de importarile implicit relative la pachet din v2.x, cat si fata de importarile explicit relative la pachet din v3.x
- Probleme cu importarile relative si cautarea absoluta din v3.x:
  - In v2.x si v3.x instructiunile de importare relativa la pachet leaga fisierul de pachet si nu mai poate fi folosit si altfel
  - In v3.x noua cautare relativa impiedica folosirea unui fisier atat ca script cat si ca modul din pachet
- Astfel:
  - In v2.x si v3.x sintaxa: ***from .*** nu poate fi folosita decat daca importatorul face parte din pachet
  - In v3.x pachetul este cautat numai cu ***from .***

Importari...

- Importarile relative in v2.x si v3.x impiedica construirea de directoare care sa fie atat programe executabile cat si importabile din exterior.
- Unele fisiere nu mai pot fi simultan module din pachet si script principal:

  from . import mod *# Nu e permisa in non pachete in v2.x si v3.x*

  import mod *# Nu cauta in directorul pachetului in v3.x*

- Se poate alege un singur mod de lucru: pachet – cu importari relative sau program – cu importari simple
- Sau se poate modifica *sys.path* – nerecomandabil
- Sau se pot folosi cai complete pana la pachet cu importari absolute, cu radacina pachetului in *sys.path*:

Importari...

- **from system.section.mypkg import mod** *# Merge atat in mod program cat si pachet*

- Problema: in v2.x merge, in v3.x nu:

 # Fisierul code\pkg\main.py:

import spam

 # Fisierul code\pkg\spam.py:

import eggs *# <== Merge daca in directorul scriptului principal*

 # Fisierul code\pkg\eggs.py:

print('Eggs' * 4) *# Nu merge ca pachet in v3.x*

C:\code>**python pkg\main.py** *# OK ca program*

EggsEggsEggsEggs

C:\code>**python pkg\spam.py** *# OK ca program*

EggsEggsEggsEggs

C:\code>py **-2** *# OK ca pachet in v2.x: relativ la pachet si apoi absolut*

>>> **import pkg.spam**

EggsEggsEggsEggs

C:\code>py **-3** *# In v3.x, cauta numai in CWD si sys.path, doar absolut **nu** relativ la pachet*

>>> **import pkg.spam**

ModuleNotFoundError: No module named 'eggs'

Importari...

- Corectie cu importarea relativa la pachet din v3.x, OK ca importare de pachet, dar NU ca program:

```
# Fisierul code\pkg\spam.py, modificat:
from . import eggs # <== Nu este pachet daca scriptul principal este aici, in pkg
# Restul, la fel
C:\code>python # OK ca pachet, NU ca program, in v2.x si v3.x
>>> import pkg.spam
EggsEggsEggsEggs
```

```
C:\code>python pkg\main.py # Python v2.x sau v3.x
ImportError: attempted relative import with no known parent package
C:\code>python pkg\spam.py # Python v2.x sau v3.x
ImportError: cannot import name 'eggs' from '__main__' (pkg\spam.py)
```

- Corectie cu pachete in subdirector:

```
# Structura code\pkg, cu sub:
pkg
├── __init__.py
├── main.py
└── sub
    ├── __init__.py
    ├── eggs.py
    └── spam.py
```

Importari...

 # Fisierul code\pkg\main.py, modificat:

import sub.spam

 # Restul, la fel, dar mutate in directorul sub

C:\code>**python pkg\main.py** *# v2.x sau v3.x OK*

EggsEggsEggsEggs

C:\code>**python** *# v2.x sau v3.x OK*

>>> **import pkg.sub.spam**

EggsEggsEggsEggs

C:\code>**python pkg\sub\spam.py** *# Testarea individuala nu este posibila!*

ImportError: cannot import name 'eggs' from '__main__' (pkg\sub\spam.py)

- Corectie cu cai complete si absolute:

 # Structura code\pkg:

```
pkg
├── __init__.py
├── eggs.py
├── main.py
└── spam.py
```

 # Directorul code se adauga la sys.path:

C:\code>**set PYTHONPATH=C:\Users\Dan\Desktop\code**

 # Fisierul pkg\main.py:

import spam

 # Fisierul pkg\spam.py: Cale completa la pachet

import pkg.eggs

 # Fisierul pkg\eggs.py, la fel

Importari...

C:\code>**python pkg\main.py** *# Ca script, in v2.x si v3.x*

EggsEggsEggsEggs

C:\code>**python** *# Din afara, in v2.x si v3.x*

>>> **import pkg.spam**

EggsEggsEggsEggs

C:\code>**python pkg\spam.py** *# Testare modul*

EggsEggsEggsEggs

- Alt exemplu cu cai complete, absolute si cod de test:
  - Un modul rulat ca script principal are numele '**__main__**'

 # Structura code\dualpkg:

```
dualpkg
├── __init__.py
├── m1.py
└── m2.py
```

 # Fisierul code\dualpkg\m1.py

```
def somefunc():
    print('m1.somefunc')
```

 # Fisierul code\dualpkg\m2.py

```
...importarea lui m1 aici... # De ales!
def somefunc():
    m1.somefunc()
    print('m2.somefunc')
if __name__ == '__main__':
    somefunc() # Testare
```

doar *ca script principal*

Importari...

 # Import relativ, OK ca pachet, NU ca script

from . import m1 *# In m2*

C:\code> **py –3**

>>> **import dualpkg.m2** *# OK*

C:\code> **py –2**

>>> **import dualpkg.m2** *# OK*

C:\code> **py -3 dualpkg\m2.py** *# Eroare!*

ImportError: cannot import name 'm1' from '__main__' (dualpkg\m2.py)

C:\code> **py -2 dualpkg\m2.py** *# Eroare!*

ValueError: Attempted relative import in non-package

 # Import simplu, OK ca script, NU ca pachet in v3.x – care nu cauta in pachet/director

import m1 # In m2

C:\code>py **-3**

>>> **import dualpkg.m2** *# Eroare in v3.x!*

ModuleNotFoundError: No module named 'm1'

C:\code>py **-2**

>>> **import dualpkg.m2** *# OK in v2.x!*

C:\code>**py -3 dualpkg\m2.py** *# OK, script*

m1.somefunc

m2.somefunc

C:\code>**py -2 dualpkg\m2.py** *# OK, script*

m1.somefunc

m2.somefunc

Importari...

```
# Import absolut, cale completa, directorul code in sys.path
import dualpkg.m1 as m1 # In m2
C:\code>py -3
>>> import dualpkg.m2 # OK ca modul
C:\code>py -2
>>> import dualpkg.m2 # OK ca modul
```

```
C:\code>py -3 dualpkg\m2.py # OK ca script
m1.somefunc
m2.somefunc
C:\code>py -2 dualpkg\m2.py # OK ca script
m1.somefunc
m2.somefunc
```

- Concluzii:
  - cale completa si import absolut sunt preferabile plasarii modulelor in subdirectoare
  - reorganizarea directoarelor poate afecta caile absolute
  - importarea relativa poate esua atunci cand modulele locale sunt relocatate

Sumar

- Importarea pachetelor
- Exemplu de import de pachet
- Utilizarea pachetelor
- Importari relative
- ***Pachete de spatii de nume (v3.3+)***

Clasificarea modelelor de importare

- Importare de baza: *import mod, from mod import atrib*
  - se importa fisiere si continutul lor, din *sys.path*
- Importari de pachete: *import dir1.dir2.mod, from dir.mod import atrib*
  - se importa din directoare cu fisier de initializare, cu cale extinsa, dar incepand dintr-un director din *sys.path*, in Python v2.x si v3.x
- Importari relative la pachet: *from . import mod* (relativ), *import mod* (absolut)
  - se importa din interiorul pachetelor, diferit intre v2.x si v3.x
- Pachete de spatii de nume: *import **splitdir**.mod*
  - importarea unui modul eventual extins in mai multe directoare, fara fisier de initializare, **in v3.3+**

Clasificarea...

- In Python 3.3+ exista doua tipuri de pachete:
  1. Modelul original – asa zisele pachete normale
  2. Modelul alternativ – acela al pachetelor de spatii de nume (*namespace*)
- Ambele sisteme coexista, cel de-al doilea fiind aplicabil cand modulele si pachetele normale lipsesc
- Modelul alternativ, fara fisiere *__init__.py* – care pot conduce la conflicte de initializare cand pachetele sunt grupate impreuna, dar cu posibilitatea unui pachet de a se afla in directoare multiple din *sys.path*, mareste flexibilitatea instalatiilor si elimina diverse incompatibilitati

Mecanismul pachetelor de spatii de nume

- Au comun cu pachetele normale obligativitatea ca primul element din calea extinsa sa se afle intr-un director din *sys.path*
- Structura este diferita: **nu** pot contine fisiere *__init__.py* si se pot intinde in mai multe directoare, la fel numite, colectate la importare

Algoritmul de importare in v3.3+

Fie *directory* in *sys.path*, iar *spam* modulul de gasit:

1. Daca exista *directory\spam__init__.py*, atunci se importa un pachet normal
2. Daca exista *directory\spam.{py|pyc|sau alta extensie valida}* se importa un modul simplu
3. Daca *directory\spam* este un director, se memoreaza
4. Se continua cu urmatorul *directory* din *sys.path*

Daca pasii (1) si (2) nu reusesc, dar cel putin un director a fost memorat in pasul (3) atunci se creeaza un pachet de spatiu de nume:

- are atributul ***__path__***, un iterabil al cailor gasite la pasul (3), stringuri, reprezinta calea parinte, virtuala, pentru componenetele pachetului
- nu are atributul ***__file__***

__init__.py este optional in v3.3+

- Fisierele de initializare pot lipsi
  - daca prezente, sunt luate in considerare
  - daca nu, directorul este considerat un pachet de spatiu de nume
    - nu mai este nevoie sa fie creat un *__init__.py* vid cand nu sunt necesare initializari
- Un pachet normal, cu *__init__.py*, este creat mai repede – algoritmul se termina la pasul (1)
  - cand nu se intentioneaza un spatiu de nume extins, pachetul normal este de preferat

Exemple de pachete de spatii de nume

- Fie directorul *sub* prezent in doua locuri de pe calea sys.path:

*# Structura directorului **code\ns**:*

```
ns
├── dir1
│   └── sub
│       ├── mod1.py
│       └── pkg
│           └── __init__.py
└── dir2
    └── sub
        ├── lower
        │   └── mod3.py
        └── mod2.py
```

C:\code>**set PYTHONPATH=C:\...\code\ns\dir1;C:\...\code\ns\dir2** *# dir1 si dir2 sunt adaugate la sys.path!*

 # Fisierul code\ns\dir1\sub\mod1.py:

print(r'dir1\sub\mod1')

 # Fisierul code\ns\dir2\sub\mod2.py:

print(r'dir2\sub\mod2')

C:\code>py **-3**

>>> **import sub**

>>> **sub** *# Pachet de spatiu de nume*

<module 'sub' (namespace)>

Exemple...

```
>>> sub.__path__
_NamespacePath(['C:\\Users\\Dan\\Desktop\\code\\ns\\dir1\\sub',
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\sub'])
>>> from sub import mod1 # Importare din directoare diferite
dir1\sub\mod1
>>> import sub.mod2 # Alt director, importare
dir2\sub\mod2
>>> mod1
<module 'sub.mod1' from
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir1\\sub\\mod1.py'>
>>> sub.mod2
<module 'sub.mod2' from
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\sub\\mod2.py'>
```

- Importarea lui *sub* se produce la primul acces, ordinea nu conteaza:

Exemple...

```
C:\code>py -3
>>> import sub.mod1
dir1\sub\mod1
>>> import sub.mod2
dir2\sub\mod2
>>> sub.mod1
<module 'sub.mod1' from
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir1\\
    sub\\mod1.py'>
>>> sub.mod2
<module 'sub.mod2' from
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\
    sub\\mod2.py'>
>>> sub
<module 'sub' (namespace)>
>>> sub.__path__
_NamespacePath(['C:\\Users\\Dan\\Desktop\\c
    ode\\ns\\dir1\\sub',
    'C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\
    sub'])
```

Exemple...

- Importarile relative in pachete de spatii de nume – sunt posibile chiar daca fisierul este in alt director:

```
# Fisierul ns\dir1\sub\mod1.py, modificat:
from . import mod2
print(r'dir1\sub\mod1')
```

```
C:\code>py -3
>>> import sub.mod1
dir2\sub\mod2
dir1\sub\mod1
>>> import sub.mod2 # A fost deja importat, nu se mai executa
>>> sub.mod2
<module 'sub.mod2' from 'C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\sub\\mod2.py'>
```

- Concluzie: pachetele de spatii de nume sunt ca pachetele normale, dar fara initializari (fara __init__.py) si amplasate eventual in mai multe directoare (acelasi nume dar cale diferita)

Incluziune in pachete de spatii de nume

- Subdirectoare dintr-un pachet de spatiu de nume sunt la randul lor spatii de nume:

*# Fisierul **code\ns\dir2\sub\lower\mod3.py**:*

print(r'dir2\sub\lower\mod3')

C:\code>py **-3**

>>> **import sub.lower.mod3** *# pachetul lower inclus in pachetul sub, ambele spatii de nume*

dir2\sub\lower\mod3

C:\code>py **-3**

>>> **import sub** *# Ordinea de acces nu conteaza!*

>>> **import sub.mod2**

dir2\sub\mod2

>>> **import sub.lower.mod3**

dir2\sub\lower\mod3

>>> **sub.lower** *# Spatiu de nume, pachet*

<module 'sub.lower' (namespace)>

>>> **sub.lower.__path__**

_NamespacePath(['C:\\Users\\Dan\\Desktop\\code\\ns\\dir2\\sub\\lower'])

Incluziune...

- Incluziunea intr-un spatiu de nume este posibila pentru un *modul simplu*, *pachet normal* sau *alt spatiu de nume*; spatiul de nume se adauga la caile de cautare:

*# Fisierul **code\ns\dir1\sub\pkg__init__.py**:*

print(r'dir1\sub\pkg__init__.py')

C:\code>py **-3**

>>> **import sub.mod2** *# Modul inclus*

dir2\sub\mod2

>>> **import sub.pkg** *# Pachet normal inclus*

dir1\sub\pkg__init__.py

>>> **import sub.lower.mod3** *# Modul din spatiu de nume inclus*

dir2\sub\lower\mod3

>>> **sub**

<module 'sub' (namespace)>

>>> **sub.mod2**

<module 'sub.mod2' from 'C:\\...\\code\\ns\\dir2\\sub\\mod2.py'>

>>> **sub.pkg**

<module 'sub.pkg' from 'C:\\...\\code\\ns\\dir1\\sub\\pkg__init__.py'>

>>> **sub.lower**

<module 'sub.lower' (namespace)>

>>> **sub.lower.mod3**

<module 'sub.lower.mod3' from 'C:\\...\\code\\ns\\dir2\\sub\\lower\\mod3.py'>

Fisierele au precedenta fata de directoare

- Fisiere/modul cu acelasi nume (minus extensia) ca un pachet de spatiu de nume (fara *__init__.py*, ar fi pachet normal) au precedenta in algoritmul de cautare:

*# Structura **code\ns2** si **code\ns3***

```
ns2
ns3
└── dir
    └── ns2.py
```

*# Fisierul **code\ns3\dir\ns2.py**:*

```
print(r'ns3\dir\ns2.py!')
```

 # PYTHONPATH este stearsa, deci dir nu este in sys.path => ns2.py nu poate fi gasit

```
C:\code>set PYTHONPATH=
C:\code>py -2
>>> import ns2 # v2.7 nu stie de spatii de nume!
ImportError: No module named ns2
C:\code>py -3
>>> import ns2 # Gasit in CWD (intotdeauna in sys.path la inceput)
>>> ns2 # E pachet de spatiu de nume in v3.3+
<module 'ns2' (namespace)>
>>> ns2.__path__
_NamespacePath(['C:\\Users\\Dan\\Desktop\\code\\ns2'])
```

Fisierele...

- PYTHONPATH=*C:\...\code\ns3\dir*, adaugat la sys.path:

```
# dir este adaugat la sys.path:
C:\code> set PYTHONPATH=C:\...\code\ns3\dir
C:\code>py -3
>>> import ns2 # Modul fisier, are precedenta
    fata de directorul ns2 din CWD
ns3\dir\ns2.py!
```

```
>>> ns2
<module 'ns2' from
    'C:\\...\\code\\ns3\\dir\\ns2.py'>
>>> import sys
>>> sys.path[:2]
['', 'C:\\Users\\Dan\\Desktop\\code\\ns3\\dir']
```

- La fel si in Python v3.2-:

```
C:\code>py -2 # Am doar v2.7 instalata...
>>> import ns2
ns3\dir\ns2.py!
```

```
>>> ns2
<module 'ns2' from 'C:\...\code\ns3\dir\ns2.py'>
```

- Pachetele normale si modulele fisier (simple) au precedenta fata de pachetele de spatii de nume:

Fisierele...

*# Structura **code\ns4:***

```
ns4
├── dir1
|   └── sub
└── dir2
    └── sub
```

C:\code>**set PYTHONPATH=C:\...\code\ns4\dir1;C:\...\code\ns4\dir2**

C:\code>py **-3**

>>> **import sub**

>>> **sub** *# Spatiu de nume, concatenat*

<module 'sub' (namespace)>

>>> **sub.__path__**

_NamespacePath(['C:\\...\\code\\ns4\\dir1\\sub', 'C:\\...\\code\\ns4\\dir2\\sub'])

*# Cand se creeaza **dir2\sub__init__.py**:*

```
ns4
├── dir1
|   └── sub
└── dir2
    └── sub
        └── __init__.py
```

C:\code>py **-3**

>>> **import sub** *# Pachet normal, cu __init__.py care are precedenta fata de directorul sub*

>>> **sub**

<module 'sub' from 'C:\\...\\code\\ns4\\dir2\\sub__init__.py'>