

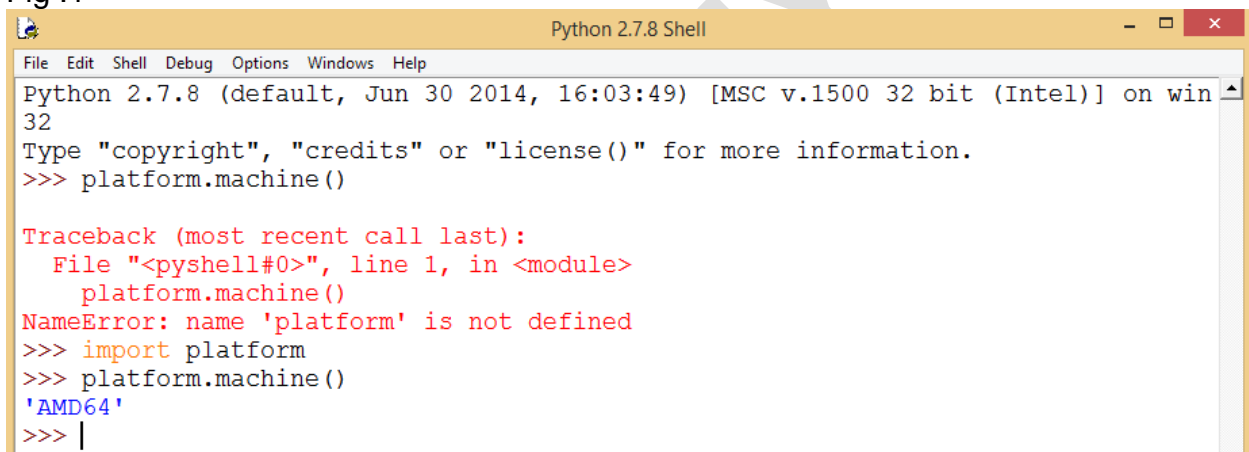
SEDINTA 5 - MODULE ÎN PYTHON

Crearea modulelor în Python

În această sedință vom discuta despre crearea și utilizarea modulelor în Python. Un modul este un program care este conceput pentru a fi reutilizat de un alt program.

Reutilizarea codului este foarte importantă, și pe acest concept se bazează de fapt toată evoluția Python-ului în sensul că spre deosebire de alte limbaje de programare de scripting (script = program micuț ce are rolul de a face modificări sistemului de fișiere sau operare), Python-ul deține o vastă comunitate care creează module și întreține cod. Spre exemplu există module de exportare în Excel, de creare de aplicații Android etc.

Reutilizarea codului se realizează prin cuvântul cheie `import` cum se poate vedea și în Fig. 1



```
Python 2.7.8 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> platform.machine()

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    platform.machine()
NameError: name 'platform' is not defined
>>> import platform
>>> platform.machine()
'AMD64'
>>> |
```

Fig. 1

Așa cum se poate observa în Fig. 1 cuvântul `platform` nu există definit până nu îl importăm noi (`import platform`). Aceasta conține clase sau funcții. Una dintre acestea este `machine` care returnează tipul arhitecturii. Poate returna patru valori posibile:

- <<AMD64>> la mașini de 64 biți
- <<i386>> la mașini de 32 biți
- <<armv7l>> la Android
- <<>> dacă nu poate determina tipul

```

Python 2.7.2 (default, Oct 25 2014, 20:52:15)
[GCC 4.9 20140827 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import platform
>>> platform.machine()
'armv7l'
>>> platform.processor()
''
>>> platform.platform()
'Linux-3.4.5-armv7l-with-libc'
>>> █

```

Fig.2

In fig. 2 puteti vedea un output al variantei python implementat pe Android (QPython).

Asa cum probabil ati gandit, platform este un modul standard. Modulele standard vin odata cu instalarea limbajului de programare Python și nu este necesara nici o modificare inainte de utilizare. De asemenea, pe langa modulele standard avem și module non-standard ce extind gradul de flexibilitate, dar pentru utilizare este necesara o instalare in prealabil.

Alte functionalitati ale modulului platform sunt:

- platform.system() – returneaza tipul sistemului de operare.
- platform.win32_ver() - returneaza tipul de windows, tipul kernel si cum trateaza taskuri multi-Processor
- platform.processor() –tipul procesor
- platform.node() –nume dispozitiv in retea
- platform.platform() - scurta descriere sistem operare.

```

>>> platform.win32_ver()
('8', '6.2.9200', '', u'Multiprocessor Free')
>>> platform.processor()
'Intel64 Family 6 Model 42 Stepping 7, GenuineIntel'
>>> platform.system()
'Windows'
>>> platform.node()
'FamPopescu'
>>> platform.platform()
'Windows-8-6.2.9200'
>>>

```

Fig.3

Pentru a gasi si accesa module Python standard puteti accesa pagina:

<https://docs.python.org/2/py-modindex.html>

Pentru a gasi si accesa module python non-standard puteti accesa pagina:

<https://pypi.python.org/>

Vom reveni la modulele standard și non-standard, dar acum trebuie înainte de toate să învățăm să creăm un modul. Mai jos putem vedea un exemplu de creare a unui modul. Vom crea doua fisiere. Primul fisier este numit "modul.py" și se regăsește mai jos:

```
# Program modul
# Explica crearea unui modul
# Ion Studentul - 1/26/13

def pa(ratie,primul_termen,nr_termeni):
    """ progresie_aritmetica."""
    try:
        x=primul_termen*nr_termeni+(ratie*nr_termeni*(nr_termeni-1))/2
    except:
        x="Toate elementele trebuie să fie numere pozitive"

    return x

def cub(nr=1):
    """ Ridicarea la cub."""
    try:
        raspuns=nr*nr*nr
    except:
        raspuns="Trebuie să dai un numar"

    return raspuns

if __name__ == "__main__":
    print "Rulezi acest modul direct, deci nu va rula nimic. Importa acest modul."
    raw_input("\n\nApasa <enter> pt a iesi.")
```

Formula Progresie aritmetica este (sursa Wikipedia):

$$S_n = \frac{(a_1 + a_n) \cdot n}{2} = \frac{(2 \cdot a_1 + (n - 1) \cdot r) \cdot n}{2} = a_1 \cdot n + r \cdot \frac{n \cdot (n - 1)}{2}$$

În fisierul modul putem observa crearea a doua functii. Prima din functii este o functie numita pa ce calculeaza progresia aritmetica avand trei parametri de intrare. A doua functie calculeaza cubul unui numar dat ca parametru.

Ambele functii calculeaza aceste formule într-un bloc de try. Astfel, dacă apare o eroare ne va returna un mesaj cu cea mai probabila eroare. Binețeles ca se putea optimiza programul ca acesta să returneze eroarea sau să ridice exceptie doar la eroare generata de un tip diferit de integer (intreg). Dar ceea ce urmărim acum este ca noi să diferentiem comenzile ce creează un modul. În ultima parte a modulului observăm :

```
if __name__ == "__main__":
```

```
print "Rulezi acest modul direct, deci nu va rula nimic. Importa acest modul."  
raw_input("\n\nApasa <enter> pt a iesi.")
```

Conditia sitaxei if (`__name__ == __main__`) este adevarata dacă acest program este rulat direct și devine fals dacă este importat ca modul. Prin urmare dacă fisierul `modul.py` este rulat direct vom afisa un mesaj prin care anuntam rularea nu este realizata dintr-un modul și trebuie importat.

Al doilea fisier importa primul modul. Poate fi numit în orice maniera. Cele doua fisiere trebuie să fie în acelasi director sau in directorul lib din directorul unde este instalat Python (standard Python este instalat in C:\Python27, deci modulul trebuie sa fie instalat in directorul C:\Python27\lib).

```
# Program importa modul  
# Explica crearea unui modul  
# Ion Studentul - 1/26/13  
  
import modul  
  
#returneaza cubul unui numar  
print "cubul numarului 3 este :",modul.cub(3)  
  
#vom da argumentele: ratie=1,primul element=3, nr de elemente=5  
print "Suma progresiei aritmetice este: ", modul.pa(1,3,5)  
#returneaza progresia aritmetica  
  
raw_input("\n\nApasa <enter> pt a iesi.")
```

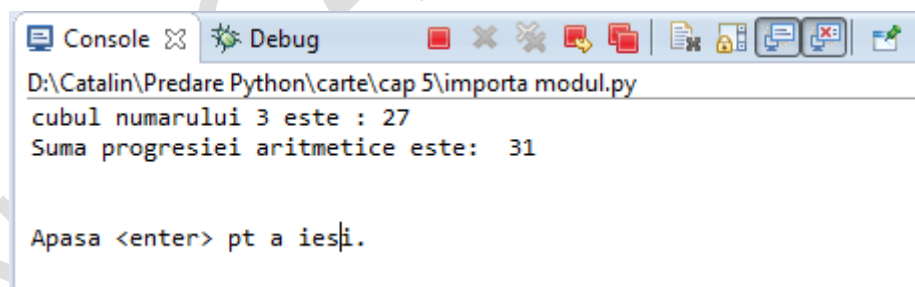


Fig.4

Al doilea fisier importa fisierul modul cu ajutorul comenzii `import`. Apoi toate functiile definite în modulul modul pot fi rulate apleand numele modulului urmat de punct.

Spre exemplu pentru a rula functia `pa`:

```
modul.pa(1,3,5)
```

De asemenea putem vedea ca acest procedeu se repeta pentru fiecare functie din modul. Asa cum era de asteptat se poate aplica și la clase.

O observatie destul de pertinenta este ca dacă dorim să importam 20 de module acestea pot fi importate pe un singur rand, astfel:

```
import modul1, modul2, modul3
```

Un package reprezinta o colectie de module.

Pentru a crea un package urmati instructiunile de mai jos:

1. Creaza un director ce va avea numele pachetului (package)
2. Pune modulele tale in fisiere separate.
3. Creaza un fisier `__init__.py` in director

Obs. Tot mereu utilizati litere mici pentru numele package-ului

Mai jos se regaseste un exemplu cu o structura de fisiere necesara sa cream un package:

```
importa_package.py
```

```
package_test\
```

```
    __init__.py
```

```
    prog_a.py
```

```
    cub_1.py
```

Unde fisierul `__init__.py` este gol, fisierul `prog_a.py` contine functia `pa` din fisierul `modul` si fisierul `cub_1.py` contine functia `cub` din fisierul `modul`. Fisierul `modul` este fisierul `modul.py` prezentat la crearea unui modul.

fisierul `prog_a.py`:

```
# Program pa
# Explica crearea unui package
# Ion Studentul - 1/26/13
```

```
def pa(ratie, primul_termen, nr_termeni):
    """ progresie_aritmetica. """
    try:
        x = primul_termen * nr_termeni + (ratie * nr_termeni * (nr_termeni - 1)) / 2
    except:
        x = "Toate elementele trebuie să fie numere pozitive"

    return x

if __name__ == "__main__":
    print "Rulezi acest modul direct, deci nu va rula nimic. Importa acest modul."
    raw_input("\n\nApasa <enter> pt a iesi.")
```

fiserul cub_1.py

```
# Program cub
# Explica crearea unui package
# Ion Studentul - 1/26/13

def cub(nr=1):
    """ Ridicarea la cub. """
    try:
        raspuns=nr*nr*nr
    except:
        raspuns="Trebuie să dai un numar"

    return raspuns

if __name__ == "__main__":
    print "Rulezi acest modul direct, deci nu va rula nimic. Importa acest modul."
    raw_input("\n\nApasa <enter> pt a iesi.")
```

Cream apoi un fisier numit importa_package.py in acelasi director unde se afla si directorul package_test si care contine:

```
# Program importa package
# Explica crearea unui package
# Ion Studentul - 1/26/13

import package_test.cub_1
import package_test.prog_a

#returneaza cubul unui numar
print "cubul numarului 3 este :", package_test.cub_1.cub(3)

#vom da argumentele: ratie=1, primul element=3, nr de elemente=5
print "Suma progresiei aritmetice este: ", package_test.prog_a.pa(1,3,5)
#returneaza progresia aritmetica

from package_test.cub_1 import cub
from package_test.prog_a import pa

#returneaza cubul unui numar
print "cubul numarului 3 este :", cub(3)

#vom da argumentele: ratie=1, primul element=3, nr de elemente=5
print "Suma progresiei aritmetice este: ", pa(1,3,5)
#returneaza progresia aritmetica

from package_test.cub_1 import cub as CeVreauEu
from package_test.prog_a import pa as CeVreiTu

#returneaza cubul unui numar
```

```
print "cubul numarului 3 este :",CeVreauEu(3)

#vom da argumentele: ratie=1,primul element=3, nr de elemente=5
print "Suma progresiei aritmetice este: ", CeVreiTu(1,3,5)
#returneaza progresia aritmetica

raw_input("\n\nApasa <enter> pt a iesi.")
```

In programul importa_package.py de mai sus putem vedea trei moduri distincte de a importa un package sau un modul.

Metoda 1:

```
import package_test.cub_1
```

Pentru a folosi functia cub din fisierul cub_1 trebuie sa importam acel fisier. Apelarea se face cu :

```
package_test.cub_1.cub(3)
```

Metoda 2:

```
from package_test.cub_1 import cub
```

Va importa din package-ul package_test din fisierul cub_1 functia cub si doar aceasta. Daca am avea 10 functii si 5 clase doar cea specificata va fi importata.

Apelarea se face aici in mod direct:

```
cub(3)
```

Metoda3:

```
from package_test.cub_1 import cub as CeVreauEu
```

Va importa din package-ul package_test din fisierul cub_1 functia cub si doar aceasta redenumind functia cub in CeVreauEu. Daca am avea 10 functii si 5 clase doar cea specificata va fi importata.

Apelarea se face aici in mod direct cu numele nou:

```
CeVreauEu(3)
```

Modulele SYS și OS

Prin importarea modului sys și os avem disponibile multe optiuni. Cele mai uzuale sunt:

- `sys.argv` – returneaza o lista de argumente date de utilizator dupa indicarea fisierului de rulare.

Spre exemplificare, vom rula un fisier din consola (command prompt în windows) astfel:

C:\Python27\python.exe C:\fisier.py argument1 argument2 argument3

`sys.argv` va deveni o lista ce va avea 4 elemente: ["numele_program", "argument1", "argument2", "argument3"]. Asa cum se poate observa tot mereu primul element din lista este numele programului. Cu aceasta functionalitate vom putea face programe ce pot interactiona cu userul prin argumentele folosite similar cu orice program de tip consola. Avem un mic exemplu mai jos:

Fisier: `sys-argv-exemplu.py`

```
import sys

print "Numele programului este : ", sys.argv[0]

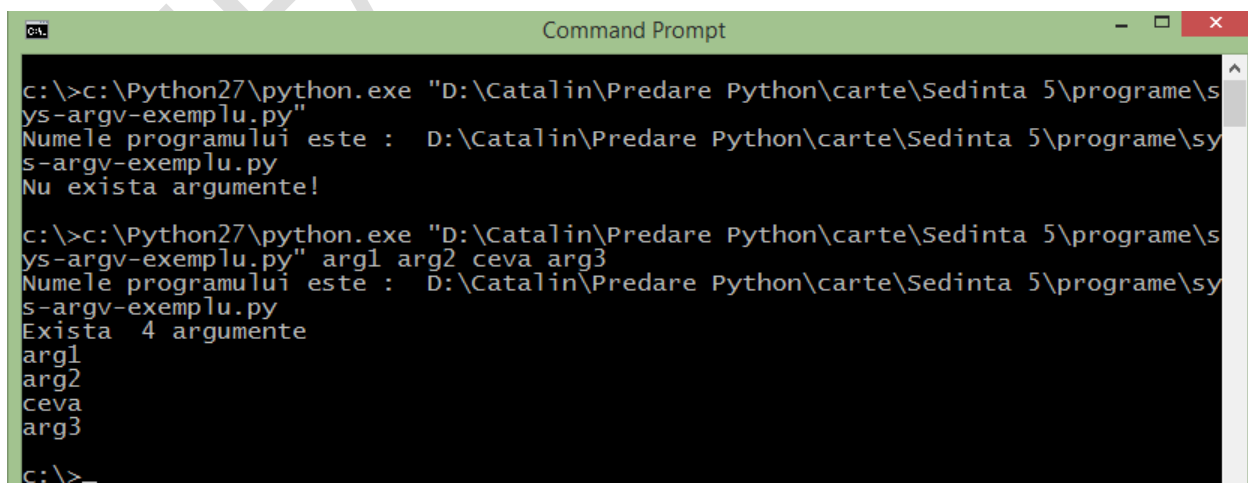
if len(sys.argv) > 1:
    print "Exista ", len(sys.argv)-1, "argumente"
    for arg in sys.argv[1:]:
        print arg
else:
    print "Nu exista argumente!"
```

La rularea acestui program fara argumente va returna:

```
Numele programului este : sys-argv-exemplu.py
Nu exista argumente!
```

Daca ar exista argumente, aceste vor fi afisate pe rand de buclă de tip "for".

Nu va afisa numele programului deoarece in linia `for arg in sys.argv[1:]`: eliminam primul element din lista prin indexare adaugand cifra 1 intre parantezele patrate.



```

C:\>c:\Python27\python.exe "D:\Catalin\Predare Python\carte\Sedinta 5\programe\sys-argv-exemplu.py"
Numele programului este : D:\Catalin\Predare Python\carte\Sedinta 5\programe\sys-argv-exemplu.py
Nu exista argumente!

c:\>c:\Python27\python.exe "D:\Catalin\Predare Python\carte\Sedinta 5\programe\sys-argv-exemplu.py" arg1 arg2 ceva arg3
Numele programului este : D:\Catalin\Predare Python\carte\Sedinta 5\programe\sys-argv-exemplu.py
Exista 4 argumente
arg1
arg2
ceva
arg3
c:\>
  
```



- Path list cu sys.path

Path List este o lista de directoare unde Python caută după module. Când pornești Python, această listă este inițializată cu conținutul variabilei de mediu a sistemului de operare PYTHONPATH și alte căi de acces standardizate.

Dacă dorim să adăugăm la această listă un modul dintr-o cale, am putea adăuga această cale cu `sys.path.append("cale")`.

Să presupunem că avem un modul în directorul `c:/Director`.

Prin importarea modului `sys` și prin rularea comenzii de mai jos putem adăuga și directorul dorit în interiorul listei pe care Python o folosește pt. a încărca un modul: `sys.path.append("c:/Director/")`



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> import modul

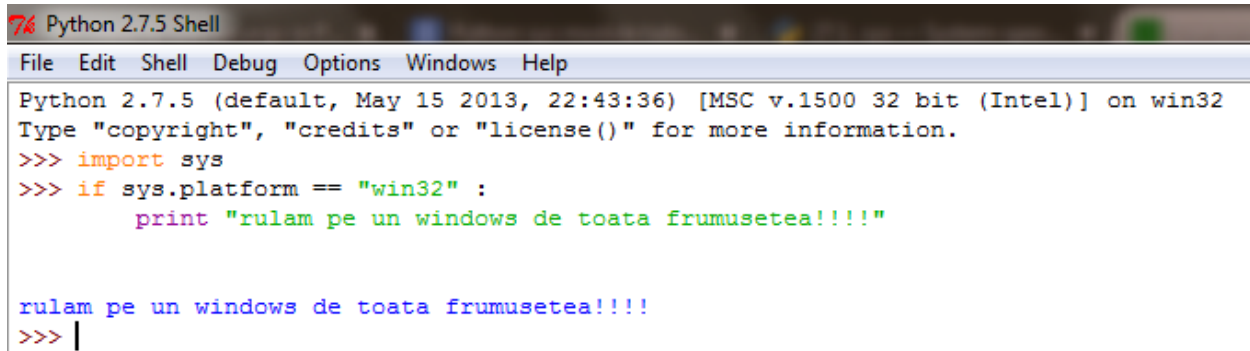
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import modul
ImportError: No module named modul
>>> import sys
>>> sys.path
['', 'C:\\Python27\\Lib\\idlelib', 'C:\\Windows\\system32\\python27.zip', 'C:\\P
ython27\\DLLs', 'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win', 'C:\\Python2
7\\lib\\lib-tk', 'C:\\Python27', 'C:\\Python27\\lib\\site-packages']
>>> print "Lungimea listei este ",len(sys.path)
Lungimea listei este 9
>>> sys.path.append("c:/Director")
>>> print "Lungimea listei este ",len(sys.path)
Lungimea listei este 10
>>> sys.path
['', 'C:\\Python27\\Lib\\idlelib', 'C:\\Windows\\system32\\python27.zip', 'C:\\P
ython27\\DLLs', 'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win', 'C:\\Python2
7\\lib\\lib-tk', 'C:\\Python27', 'C:\\Python27\\lib\\site-packages', 'c:/Directo
r']
>>> import modul
>>>
>>> modul.cub(2)
8
>>> |
```

Fig.3

Așa cum se poate vedea în Figura 3, modulul `modul` nu este disponibil de la început. După ce importăm modulul `sys` și utilizăm `sys.path.append` să adăugăm calea către modul, atunci acesta poate fi importat. De asemenea observăm că pentru a vizualiza lista curentă de căi de acces putem apela `sys.path`.

- `sys.platform`

Aceasta facilitate ne indica sistemul de operare unde ruleaza programul.



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import sys
>>> if sys.platform == "win32" :
    print "rulam pe un windows de toata frumusetea!!!!"

rulam pe un windows de toata frumusetea!!!!
>>> |
```

Fig.4

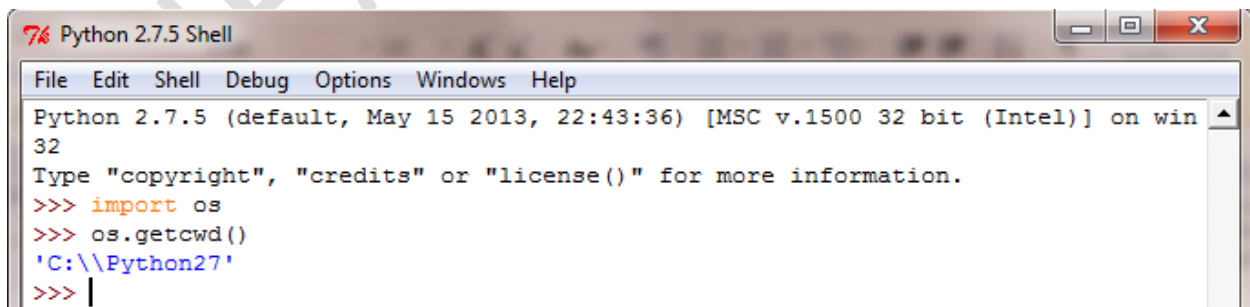
Toate sistemele de operare Windows returneaza "win32". Dacă rulam aceasta comanda pe mac va returna "mac". În general Linux returneaza "posix", dar outputul este derivat din comanda din linux "uname -r" ce poate intoarce și alte valori. Spre exemplu pentru solaris comanda `sys.platform` returneaza "sunos5". In android aceasta comanda returneaza „linux2” (Testat pe android 4.2)

- `sys.exit()`

Aceasta comanda are ca scop inchiderea aplicatiei curente. Poate fi folosit în cadrul unei bucle infinite pentru a iesi din program. Aceasta buclă apare în general în programarea interfetelor grafice.

- `os.getcwd()`

Aceasta sintaxă are rolul de a arata directorul curent din care interpretorul ruleaza programul. Dacă rulam aceasta comanda din IDLE putem observa directorul de unde ruleaza IDLE(Fig.4).

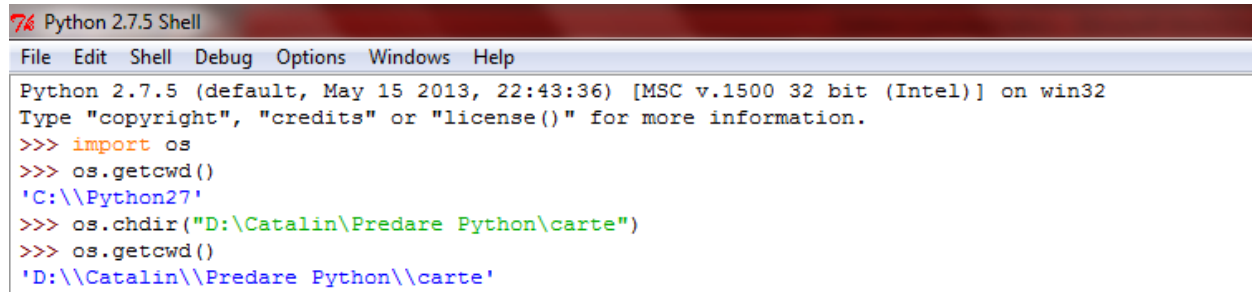


```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.getcwd()
'C:\\Python27'
>>> |
```

Fig.5

- `os.system()` - ruleaza o comanda în CMD/ Terminal. Observatie! Nu returneaza și outputul!

- `os.chdir("cale")` - schimba directorul curent



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.getcwd()
'C:\\Python27'
>>> os.chdir("D:\\Catalin\\Predare Python\\carte")
>>> os.getcwd()
'D:\\Catalin\\Predare Python\\carte'
```

Fig.6

- `os.listdir("cale")` - returneaza fisierele dintr-un director

```
>>> os.listdir("c:\\Director")
['modul.py', 'modul.pyc']
```

Fig.7

- `os.name` – similar cu `sys.platform`

Returneaza 'nt' pentru Windows, 'posix' pentru Linux si Android și 'mac' pentru mac.

```
>>> os.name
'nt'
```

Fig. 8

- `os.path.exists("cale")` - returneaza True dacă exista calea.
- `os.path.isdir("cale")` - returneaza True dacă calea indica un director.
- `os.path.isfile("cale")` - returneaza True dacă calea indica un fisier.

```
>>> os.path.exists("C:/Director")
True
>>> os.path.isdir("C:/Director")
True
>>> os.path.isfile("C:/Director")
False
```

Fig.8

- `os.linesep`

Pentru fiecare sistem de operare exista un separator de linie, adica un șir de caractere care face delimitarea intre liniile unui fisier. Pentru linux acest șir de caractere este "\n", iar pentru windows este "\r\n". Pentru text files deschise în windows cu functia open putem utiliza și direct "\n" pe toate platformele.

- `os.sep`

Caracterul folosit de sistemul de operare pentru a indica calea prin directoare.

Pentru Windows este `"\"`, iar pentru linux este `"/"`.

Se poate folosi din Python orice formă și se va converti automat în formă necesara.

```
>>> os.linesep
'\r\n'
>>> os.sep
'\'
```

Fig.9

Putem sterge fisiere sau directoare cu ajutorul modului `os` din python astfel:

- `os.remove("cale_fisier")` - sterge fisierul indicat. Ridica eroare daca nu e fisier sau fisierul este folosit
- `os.rmdir("cale_director")` - sterge directorul indicat. Ridica eroare daca directorul nu este gol sau directorul este folosit.
- `os.removedirs("cale_director")` - sterge directorul indicat recursiv. Ridica eroare daca in calea recursiva exista fisiere (in subdirectoare).

```
>>> os.listdir("C:\\test\\")
['test.txt']
>>> os.remove("C:\\test\\test.txt")
>>> os.listdir("C:\\test\\")
[]
>>> os.removedirs("C:\\test\\")
>>>
```

Fig. 10

Modulul Random

Prin importarea modului `random` putem utiliza urmatoarele optiuni:

- `random.choice("sir de caractere")` – alege un caracter aleatoriu din sirul de caractere.

- `random.randrange(start=0,stop=None,step=1)`

Spre exemplu, `random.randrange(1,6,1)` ar genera un numar aleatoriu cuprins intre 1 și 6 inclusiv.

Daca apleam comanda `random.randrange(6)` ar genera un numar aleatoriu cuprins intre 0 și 5 inclusiv, deoarece functia are start default la 0, noi indicam stop-ul,iar pasul este la 1.

```
>>> import random
>>> random.choice("avem mere frumoase")
'm'
>>> random.randrange(6)
3
```

Fig.11

Modulul tempfile

Acest modul creaza un fisier temporar în mod automat. Acest fisier poate fi utilizat ca oricare alt fisier.

```
>>> import tempfile
>>> tempfile = tempfile.mktemp()

>>> print "tempfile", "=>", tempfile
tempfile => c:\users\popescu\appdata\local\temp\tmpbtlb91
>>>
>>> import os
>>> T_file= open(tempfile,"w")
>>> T_file.write("Test infoacademy\n")
>>> T_file.close()
>>> os.remove(tempfile)
>>> |
```

Fig.12

Consider ca este benefica utilizarea acestor directoare de tip temporar deoarece de cele mai multe ori un program creaza log-uri sau fisier ce se pot genera în mod automat dacă sistemul sterge tot ce este în directoarele temporare.

Daca dorim să cream și să deschidem un fisier pentru eventuale adaugari putem utiliza în mod direct functia `TemporaryFile` din modulul `tempfile`:

```

>>> import tempfile
>>> fisier = tempfile.TemporaryFile()
>>> fisier = tempfile.TemporaryFile("w+")
>>> # asta este modul default pentru TemporaryFile => w+
>>>

```

Fig. 13

Asa cum se poate vedea și în Fig. 13 modul standard pentru a deschide fisierul creat automat este "w+".

```

>>> fisier.write("test test test")
>>> fisier.close()
>>>

```

Fig. 14

Ulterior se poate folosi acest fisier ca orice alt fisier. De asemenea dacă doriți să știți ce director este utilizat în momentul de față atunci trebuie să apelați la funcția `gettempdir()` din cadrul modulului `tempfile`. Cum puteți crea un fisier trebuie să aveți posibilitatea de a crea și un director temporar; iar aceasta operațiune este disponibilă cu ajutorul funcției `mkdtemp()`

```

>>> print tempfile.gettempdir()
c:\users\popescu\appdata\local\temp
>>>
>>> tempfile.mkdtemp()
'c:\\users\\popescu\\appdata\\local\\temp\\tmpulxsfb'
>>> |

```

Fig. 15

Stocarea în fișiere cu cPickle

Stocarea datelor în fișiere de tip text este convenabilă, dar are și minusuri. Spre exemplu un inconvenient este acela că tu trebuie să realizezi un mecanism prin care să extragi variabilele sau să le introduci. În Python se numește Pickling procesul de conversie a datelor complexe și de mai multe tipuri cu scopul de a fi stocate în fișiere, dar în alte limbaje de programare acest proces se numește serialization sau marshaling.

Există două variante de Pickle. Prima variantă este Pickle, iar cea a doua este cPickle. Varianta cPickle este scrisă în C, fiind un pic mai rapidă decât cea scrisă în Python. Din acest motiv se regăsește doar varianta cPickle ca modul standard.

Mai jos regăsim un exemplu în care utilizăm Pickle, apoi îl vom explica fiecare pas.

```

# Program Pickle
# Explica utilizarea modulului Pickle
# Ion Studentul - 1/26/13

import cPickle

nume = {1:"Popescu", 2:"Scaraoski", 3:"Mahomed",4:"McDonald"}
pasari=["vrabie","vultur","porumbel","corb"]

fisier = open("pickle_1.bazadedate", "w")

cPickle.dump(nume, fisier)
cPickle.dump(pasari, fisier)

fisier.close()

fisier_citit = open("pickle_1.bazadedate", "r")

nume_citit = cPickle.load(fisier_citit)
pasari_citit = cPickle.load(fisier_citit)

print nume_citit
print pasari_citit

raw_input("\n\nApasa <enter> pt a iesi.")

```

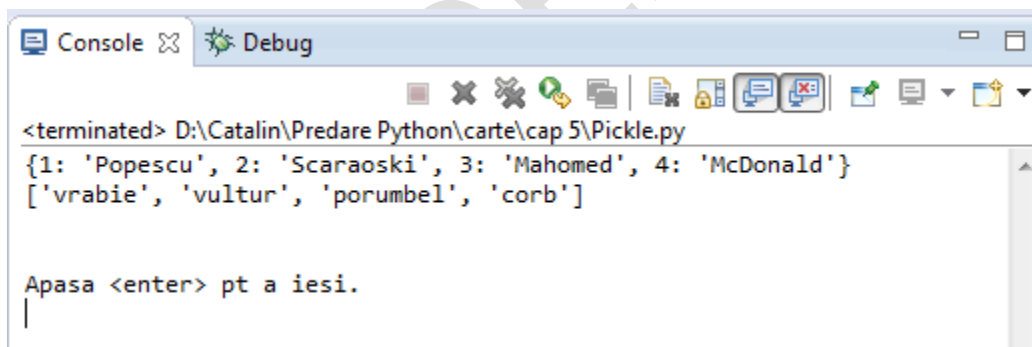


Fig .16

cPickle utilizeaza un limbaj propriu pentru a determina inceputul fiecarei variabile. Astfel dupa importarea modulului cPickle și dupa crearea unor variabile cu scopul de a le stoca, vom deschide un fisier, asa cum am deschis și pana acum.

Pentru a scrie o variabila vom utiliza cPickle.dump(variabila, nume fisier), iar pentru a citi o variabila vom utiliza cPickle.load(nume fisier).

Totusi trebuie să avem în vedere anumite aspecte; dacă avem doar doua variabile salvate în fisier și incercam să apelam load a treia oara, atunci va da eroare. De asemenea, trebuie să retinem ca variabilele sunt adaugate și citite incremental (adica de la prima la ultima).

Modulul getpass

Acest modul are doua functii foarte utile. Prima functie ce returneaza userul curent ce s-a logat la masina este getuser().

A doua functie getpass.getpass() poate fi folosita doar în cadrul consolei deoarece permite să nu se vada parola scrisa în consola(rulare prin dublu click a fisierului python).

Iata programul apelat și rularea porgramului:

```
# Program GetPass
# Explica functiile getPass
# Ion Studentul - 1/26/13

import getpass

usr = getpass.getuser()# returneaza userul logat

passw = getpass.getpass("Introdu parola pentru userul "+usr+":")

print usr, passw

raw_input("\n\nApasa <enter> pt a iesi.")
```

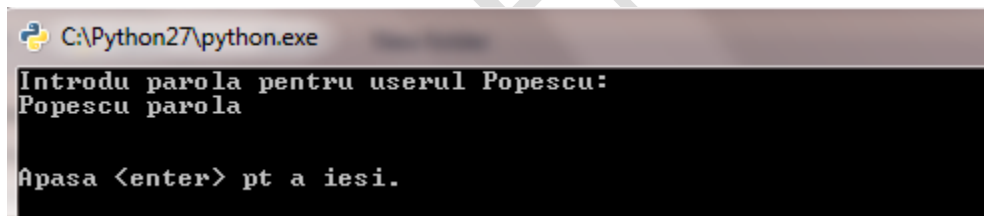


Fig.17

Modulul math

O data cu utilizarea acestui modul vreau să subliniez și anumite probleme ce pot aparea cand prelucram numere. Spre exemplu cu siguranta ati considera ca tipul de variabila pentru 1j este un string sau poate un integer. Similar is pentru 3+3j. Totusi numerele complexe au un element în plus ce ar fi reprezentat în matematica din Romania cu i. Adica exista numarul complex ce nu face parte din mutimea numerelor reale 1+i. În tarile din vest, i-ul a fost inlocuit cu j. Trebuie să fim cu bagare de seama de acest aspect în verificarea aditionala a numerelor introduse de la tastatura.


```
>>> var1 = 1j
>>> var2 = 3+3j
>>> var3 = 3
>>> type(var1)
<type 'complex'>
>>> type(var2)
<type 'complex'>
>>> type(var3)
<type 'int'>
>>>
```

Fig. 18

Pentru a realiza operatii matematice mai complexe puteti utiliza modulul matematic math. Acesta este dependent de platforma, deci exista cazuri în care raspunsurile pot fi diferite. Modulul math poate fi utilizat să obținem numerele pi și numărul e sau să realizăm ridicare la putere/radical.

```
>>> print "numarul natural e: ",math.e
numarul natural e:  2.71828182846
>>> print "numarul pi:", math.pi
numarul pi: 3.14159265359
>>> print "2 la puterea a treia este:", math.pow(2,3)
2 la puterea a treia este: 8.0
>>> print "radical din 8 este:",math.sqrt(8)
radical din 8 este: 2.82842712475
>>> print "radical din 9 este:",math.sqrt(9)
radical din 9 este: 3.0
>>> |
```

Fig. 19

Module pentru timp

În Python putem administra timpul și data în diferite moduri. Conversia datei este un lucru comun în programare

Intervalele de timp sunt numere ce sunt flotante (curgatoare) în secunde. Anumite instante de timp sunt exprimate în secunde de la 12:00am ,1 Ianuarie 1970 (numit pe scurt epoch). Exista un modul foarte utilizat numit **time** ce ofera functii pentru lucrul cu timpul și pentru a converi timpul.

```
>>> import time
>>>
>>>
>>> # Aceasta comanda va afisa timpul epoch
>>> print time.time()
1403297810.48
```

Fig.20

Multe functii ale modului Python time sunt reprezentate de un tuplu de 9 numere, dupa cum urmeaza:

Index	Camp	Valori
0	Anul	2008
1	Luna	1 to 12
2	Ziua	1 to 31
3	Ora	0 to 23
4	Minutul	0 to 59
5	Secunda	0 to 61 (60 or 61 are leap-seconds)
6	Ziua saptamanii	0 to 6 (0 is Monday)
7	Ziua anului	1 to 366 (Julian day)
8	Ora de Vara	-1, 0, 1, -1 means library determines DST

Aceasta poarta numele și de structura **struct_time** și se poate regasi mai jos:

Index	Atribut	Valori
0	tm_year	2014
1	tm_mon	1 – 12
2	tm_mday	1 – 31
3	tm_hour	0 – 23
4	tm_min	0 – 59
5	tm_sec	0 - 61 (60 sau 61 sunt secunde de reglaj)
6	tm_wday	0 - 6 (0 este Luni)
7	tm_yday	1 - 366 (Calendarul Iulian: 366 zile an bisect)

8	tm_isdst	-1, 0, 1, dacă se aplica ora de vara
---	----------	--------------------------------------

Exista metode de a formata timpul din epoch la data curenta, dar cea mai simpla este cea integrata în modulul time.

```
>>> import time
>>> localtime = time.asctime( time.localtime(time.time()) )
>>>
>>>
>>> print localtime
Sat Jun 21 00:35:38 2014
>>>
>>> print time.localtime(time.time())
time.struct_time(tm_year=2014, tm_mon=6, tm_mday=21, tm_hour=0, tm_min=36, tm_sec=15, tm_wday=5, tm_yday=172, tm_isdst=1)
>>> time.localtime()
time.struct_time(tm_year=2014, tm_mon=6, tm_mday=21, tm_hour=0, tm_min=36, tm_sec=36, tm_wday=5, tm_yday=172, tm_isdst=1)
>>> time.asctime( time.localtime())
'Sat Jun 21 00:37:46 2014'
```

Fig.21

Asa cum se poate vedea și în Fig. 21 `time.localtime()` este un formator de timp la timpul actual, apelând în mod automat `time.time()`. Dacă dorim să avem o forma și mai ușor de citit putem utiliza `time.asctime()`.

Putem obtine de asemenea și informații legate de calendar. Mai jos regăsim un cod de caractere care afișează calendarul nostru pe o anumită perioadă.

```
>>> import calendar
>>>
>>> Decembrie= calendar.month(2015,12)
>>> print Decembrie
December 2015
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Fig.22

De asemenea, dacă dorim să aflăm a câta zi din săptămână este o anumită zi.

```
>>>
>>> zi = calendar.weekday(2020,11,24)
>>> print "Ziua 24 din luna 11 a anului 2020 este a ",zi+1," a saptamanii"
Ziua 24 din luna 11 a anului 2020 este a 2 a saptamanii
>>> #se pleaca de la 0 cu luni iar 6 reprezinta duminica
>>>
```

Fig.23

Arhivarea și dezarhivarea de fisiere

Arhivarea de fisiere este foarte importanta deoarece instalarea propriu-zisa poate fi realizata cu ajutorul arhivarii. De asemenea, un raport complet de crash(prabusire a programului) poate contine o multime de informatii ce trebuiesc trimise arhivate. Cu ajutorul modulului zipfile permite arhivarea și dezarhivarea cu extensia zip.

Daca dorim să manipulam un fisier de tip arhiva trebuie să cream un obiect de tip zipfile cu ajutorul sintaxei:

Obiect=zipfile.ZipFile("nume_arhiva.zip",'r')

Sirul de caractere "r" reprezinta cum dorim să manipulam arhiva. În acest caz dorim să citim arhiva

Pentru a lista continutul unei arhive existente trebuie să uitlizam metodele namelist(retorneaza o lista cu fisierele) și infolist (retorneaza o lista cu instantele zipinfo).

```
# Program zipfile1
# Explica zipfile
# Ion Studentul - 1/26/13

import zipfile

fisier_zip = zipfile.ZipFile("exemplu.zip", "r")

# listeaza fisierele
for nume in fisier_zip.namelist():
    print "Fisier:",nume

# list file information
for info in fisier_zip.infolist():
    print '\n',info.filename
    print '\tComment:\t', info.comment
    print '\tSystem:\t\t', info.create_system, '(0 = Windows, 3 = Unix)'
    print '\tZIP version:\t', info.create_version
    print '\tCompressed:\t', info.compress_size, 'bytes'
    print '\tUncompressed:\t', info.file_size, 'bytes'

fisier_zip.close()
```

```
raw_input("\n\nApasa <enter> pt a iesi.")
```

```
<terminated> D:\Catalin\Predare Python\carte\cap 5\Zipfile1.py
Fisier: pickle_1.asd
Fisier: pickle_1.bazadedate
Fisier: Python_ico.ico
Fisier: setup.py

pickle_1.asd
  Comment:
  System:      0 (0 = Windows, 3 = Unix)
  ZIP version: 20
  Compressed:  115 bytes
  Uncompressed: 165 bytes

pickle_1.bazadedate
  Comment:
  System:      0 (0 = Windows, 3 = Unix)
  ZIP version: 20
  Compressed:  115 bytes
  Uncompressed: 165 bytes

Python_ico.ico
  Comment:
  System:      0 (0 = Windows, 3 = Unix)
  ZIP version: 20
  Compressed:  7787 bytes
  Uncompressed: 103147 bytes

setup.py
  Comment:
  System:      0 (0 = Windows, 3 = Unix)
  ZIP version: 20
  Compressed:  94 bytes
  Uncompressed: 115 bytes

Apasa <enter> pt a iesi.
```

Fig. 24

Pentru a citi un fisier arhivat trebuie să utilizăm metoda `read()`. Mai jos se regăsește un exemplu în care utilizăm metoda `read()` pentru a extrage un fisier

```
# Program zipfile2
# Explica zipfile
# Ion Studentul - 1/26/13

import zipfile

fisier_zip = zipfile.ZipFile("exemplu.zip", "r")

for name in fisier_zip.namelist():
    data = fisier_zip.read(name)
    print name, " Lungime:", len(data)
    print "primele 10 caractere:", repr(data[:10]), "\n"

for name in fisier_zip.namelist():
    fisier = open("D:\\\\" + name, "w+")
    fisier.write(data)
```

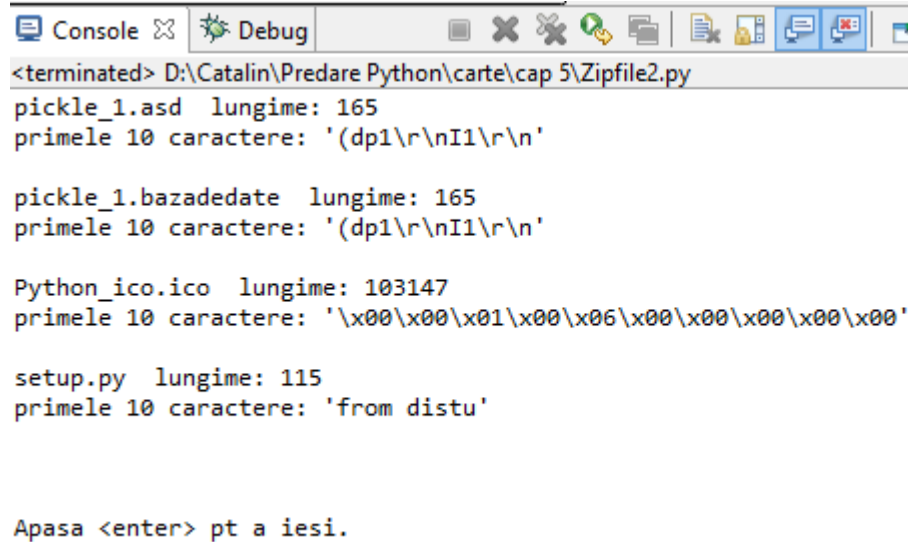
```

    fisier.close()

    fisier_zip.close()

    raw_input("\n\nApasa <enter> pt a iesi.")

```



```

<terminated> D:\Catalin\Predare Python\carte\cap 5\Zipfile2.py
pickle_1.asd  lungime: 165
primele 10 caractere: '(dp1\r\nI1\r\n'

pickle_1.bazadedate  lungime: 165
primele 10 caractere: '(dp1\r\nI1\r\n'

Python_ico.ico  lungime: 103147
primele 10 caractere: '\x00\x00\x01\x00\x06\x00\x00\x00\x00\x00'

setup.py  lungime: 115
primele 10 caractere: 'from distu'

Apasa <enter> pt a iesi.

```

Fig. 25

Daca dorim să extragem toata arhiva putem utiliza sintaxa:

```
fisier_zip.extractall("D:\\")
```

In cazul în care dorim să trimitem multiple fisiere, trebuie să utilizam functia write().

```

# Program zipfile3
# Explica zipfile
# Ion Studentul - 1/26/13

import zipfile

fz = zipfile.ZipFile('fisier_zip.zip', mode='w')
try:
    print 'Adauga fisiere'
    fz.write('setup.py')
    fz.write('Python_ico.ico')
finally:
    print 'Inchiderea arhivei'
    fz.close()

raw_input("\n\nApasa <enter> pt a iesi.")

```

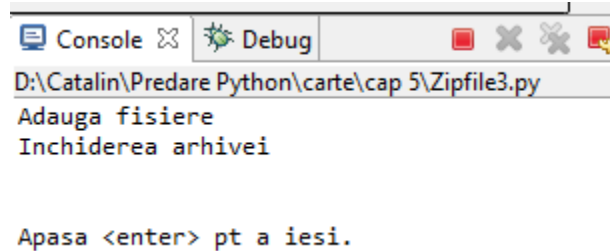


Fig. 26

Daca comparam dimensiunea fisierelor comprimate cu dimensiunile fisierelor ce formeaza fisierul zip putem vedea ca dimensiunile sunt egale. Dacă dorim să adaugam și o modalitate de compresie trebuie să adaugam la crearea obiectului zip expresia `compression=zipfile.ZIP_DEFLATED`. `ZIP_DEFLATED` are nevoie de modulul `zlib`, modul ce este instalat standard. Dacă nu specifici nimic, standard considera ca metoda de compresie este `zipfile.ZIP_STORED` care doar stocheaza fisierele fara ale compresa.

Se poate regasi un exemplu mai jos unde folosim comprimarea fisierelor!

```
# Program zipfile4
# Explica zipfile
# Ion Studentul - 1/26/13

import zipfile

print '\n\tCream o arhiva necompresata'
fz_necompresat = zipfile.ZipFile('fisiere_zip.zip', mode='w')
try:
    print '\tAdauga fisiere'
    fz_necompresat.write('setup.py')
    fz_necompresat.write('Python_ico.ico')
finally:
    print '\tInchiderea arhivei'
    fz_necompresat.close()

for info in fz_necompresat.infolist():
    print info.filename, "Marime fisier", info.file_size, "! Marime fisier compresat",
    info.compress_size, "!"

print '\n\tCream o arhiva compresata'
fz = zipfile.ZipFile('fisiere_zip.zip', mode='w', compression=zipfile.ZIP_DEFLATED)
try:
    print '\tAdauga fisiere'
    fz.write('setup.py')
    fz.write('Python_ico.ico')
finally:
    print '\tInchiderea arhivei'
    fz.close()
```

```

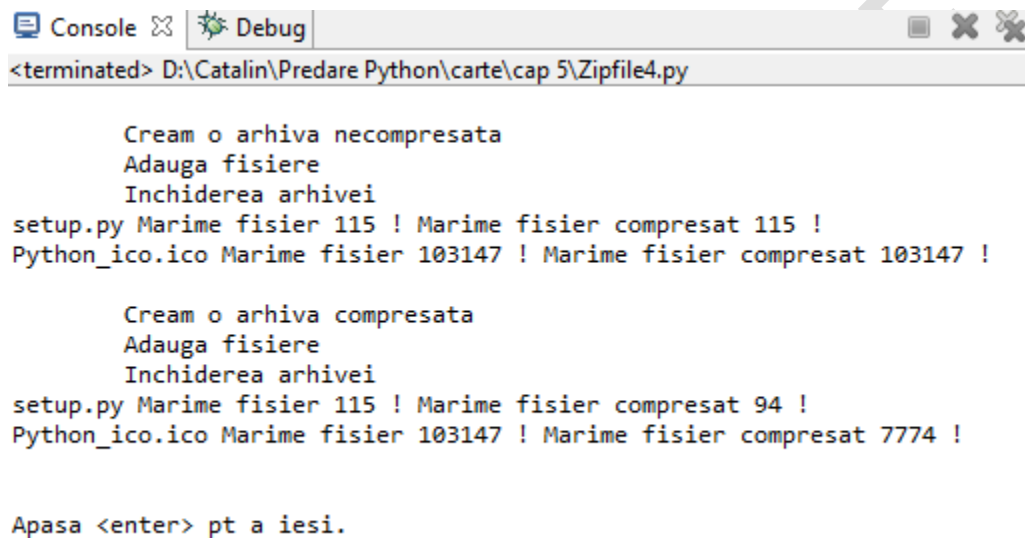
fisier_zip = zipfile.ZipFile("fisier_zip.zip", "r")

for info in fisier_zip.infolist():
    print info.filename, "Marime fisier",info.file_size,"! Marime fisier compresat",
    info.compress_size,"!"

fisier_zip.close()

raw_input("\n\nApasa <enter> pt a iesi.")

```



```

<terminated> D:\Catalin\Predare Python\carte\cap 5\Zipfile4.py

    Cream o arhiva necompresata
    Adauga fisiere
    Inchiderea arhivei
    setup.py Marime fisier 115 ! Marime fisier compresat 115 !
    Python_ico.ico Marime fisier 103147 ! Marime fisier compresat 103147 !

    Cream o arhiva compresata
    Adauga fisiere
    Inchiderea arhivei
    setup.py Marime fisier 115 ! Marime fisier compresat 94 !
    Python_ico.ico Marime fisier 103147 ! Marime fisier compresat 7774 !

    Apasa <enter> pt a iesi.

```

Fig. 27

Modulul Py2exe

Py2exe este un modul instalabil ce transforma un .py intr-un executabil de windows.

Executabilul creat în py2exe poate fi vandut sub propria licenta.

Acest modul poate fi downloadat de pe site-ul:

<http://sourceforge.net/projects/py2exe/files/py2exe/0.6.9/>

Acest modul are nevoie de pywin32 ce poate fi downloadat de pe site-ul:

<http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/>

Pentru a converti un .py în .exe trebuie să avem mai întâi un fișier *.py. Va rog să creați un fișier în c:/numit MyPyExe.py ce să contină afișarea unui text:

```
# Program MyPyExe
# Explica crearea unui exe
# Ion Studentul - 1/26/13

import os

print "Sistemul de operare este ",os.name()

raw_input("\n\nApasa <enter> pt a iesi.")
```

Trebuie să rulăm acest program să vedem dacă funcționează:

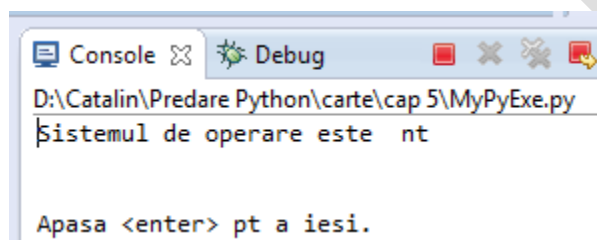


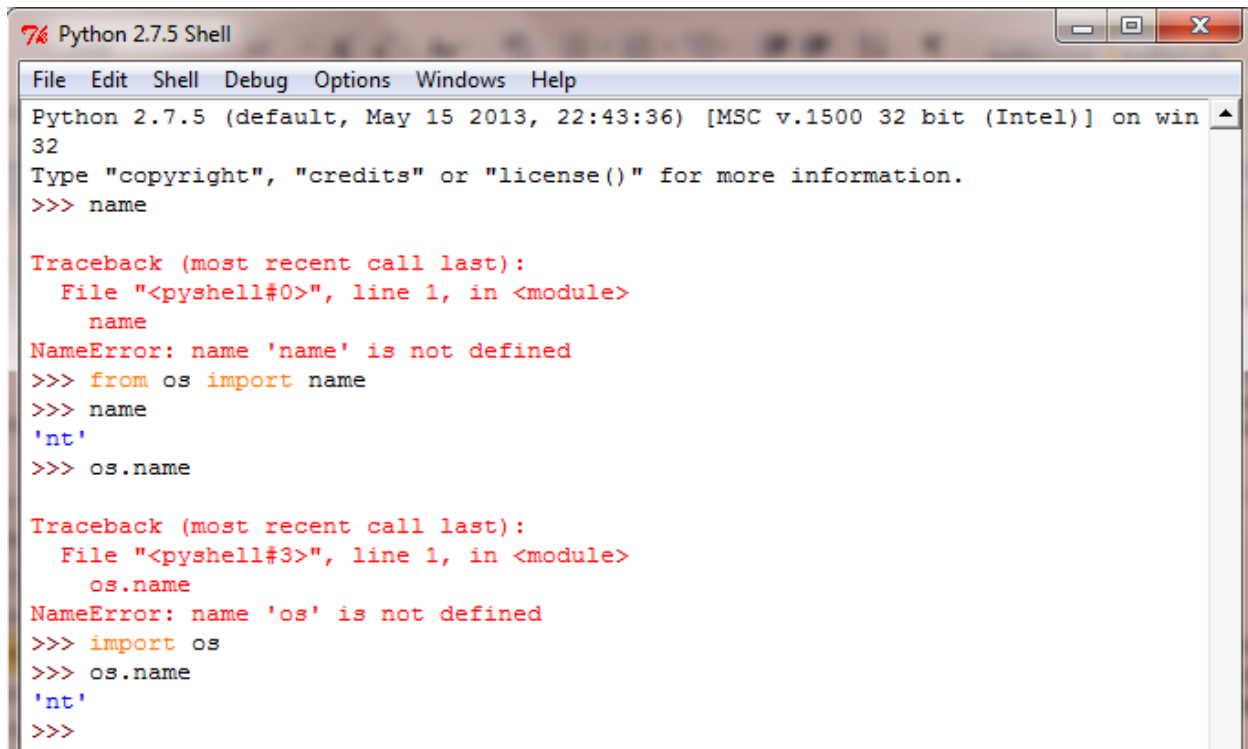
Fig.28

Pentru a transforma un fișier py într-un executabil trebuie să cream un alt fișier py în care să-i indicăm ce fișier trebuie să se convertească în executabil de tip exe. Nu există o condiție ca cele două fișiere să facă parte din același director. Îl vom denumi setup.py

```
from distutils.core import setup
import py2exe

setup(console=['c:\MyPyExe.py'])
```

Acest fișier importă din distutils.core doar modulul setup. Va reamintesc cum se poate folosi sintaxa from modul import clasa. Iată un exemplu mai jos.



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> name

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    name
NameError: name 'name' is not defined
>>> from os import name
>>> name
'nt'
>>> os.name

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    os.name
NameError: name 'os' is not defined
>>> import os
>>> os.name
'nt'
>>>
```

Fig.29

Prin urmare functia os este disponibila acum în mod direct, iar noi vom apela functia pentru a indica programul care dorim sa-l convertim prin a declara parametru console în mod direct.

Apoi deschidem un command prompt și navigam în locatia unde avem fiserul setup.py.

Vom rula din consola comanda "setup.py py2exe" cum se poate vedea și în Fig 30.

```

Administrator: C:\Windows\system32\cmd.exe

c:\Director>setup.py py2exe
running py2exe
*** searching for required modules ***
*** parsing results ***
creating python loader for extension 'unicodedata' (C:\Python27\DLLs\unicodedata.pyd -> unicodedata.pyd)
creating python loader for extension 'select' (C:\Python27\DLLs\select.pyd -> select.pyd)
creating python loader for extension '_hashlib' (C:\Python27\DLLs\_hashlib.pyd -> _hashlib.pyd)
creating python loader for extension 'bz2' (C:\Python27\DLLs\bz2.pyd -> bz2.pyd)
*** finding dlls needed ***
*** create binaries ***
*** byte compile python files ***
skipping byte-compilation of C:\Python27\lib\StringIO.py to StringIO.pyc
skipping byte-compilation of C:\Python27\lib\UserDict.py to UserDict.pyc
skipping byte-compilation of C:\Python27\lib\_future_.py to _future_.pyc
skipping byte-compilation of C:\Python27\lib\_abcoll.py to _abcoll.pyc
skipping byte-compilation of C:\Python27\lib\_strptime.py to _strptime.pyc
skipping byte-compilation of C:\Python27\lib\_threading_local.py to _threading_local.pyc
skipping byte-compilation of C:\Python27\lib\_weakrefset.py to _weakrefset.pyc
skipping byte-compilation of C:\Python27\lib\abc.py to abc.pyc
skipping byte-compilation of C:\Python27\lib\atexit.py to atexit.pyc
skipping byte-compilation of C:\Python27\lib\base64.py to base64.pyc
skipping byte-compilation of C:\Python27\lib\bdb.py to bdb.pyc
skipping byte-compilation of C:\Python27\lib\calendar.py to calendar.pyc
skipping byte-compilation of C:\Python27\lib\cmd.py to cmd.pyc
skipping byte-compilation of C:\Python27\lib\codecs.py to codecs.pyc
skipping byte-compilation of C:\Python27\lib\collections.py to collections.pyc
skipping byte-compilation of C:\Python27\lib\copy.py to copy.pyc
skipping byte-compilation of C:\Python27\lib\copy_reg.py to copy_reg.pyc
skipping byte-compilation of C:\Python27\lib\difflib.py to difflib.pyc
skipping byte-compilation of C:\Python27\lib\dis.py to dis.pyc
skipping byte-compilation of C:\Python27\lib\doctest.py to doctest.pyc
skipping byte-compilation of C:\Python27\lib\dummy_thread.py to dummy_thread.pyc
skipping byte-compilation of C:\Python27\lib\encodings\__init__.py to encodings\__init__.pyc
skipping byte-compilation of C:\Python27\lib\encodings\aliases.py to encodings\aliases.pyc
skipping byte-compilation of C:\Python27\lib\encodings\ascii.py to encodings\ascii.pyc
skipping byte-compilation of C:\Python27\lib\encodings\base64_codec.py to encodings\base64_codec.pyc
skipping byte-compilation of C:\Python27\lib\encodings\big5.py to encodings\big5.pyc
skipping byte-compilation of C:\Python27\lib\encodings\big5hkscs.py to encodings\big5hkscs.pyc
skipping byte-compilation of C:\Python27\lib\encodings\bz2_codec.py to encodings\bz2_codec.pyc
skipping byte-compilation of C:\Python27\lib\encodings\charmap.py to encodings\charmap.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp037.py to encodings\cp037.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1006.py to encodings\cp1006.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1026.py to encodings\cp1026.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1140.py to encodings\cp1140.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1250.py to encodings\cp1250.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1251.py to encodings\cp1251.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1252.py to encodings\cp1252.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1253.py to encodings\cp1253.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1254.py to encodings\cp1254.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1255.py to encodings\cp1255.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1256.py to encodings\cp1256.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1257.py to encodings\cp1257.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp1258.py to encodings\cp1258.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp424.py to encodings\cp424.pyc
skipping byte-compilation of C:\Python27\lib\encodings\cp437.py to encodings\cp437.pyc

```

Fig.30

Aceasta comanda va crea doua directoare în directorul unde se afla setup.py. În directorul build se vor compila toate dependentele. În directorul dist regasim executabilul și multe din dependentele necesare pentru a putea rula. Exista dependente care nu pot fi redistribuite datorita licentei Windows. Astfel regasim urmatorul mesaj la finalul rularii:

```

*** binary dependencies ***
Your executable(s) also depend on these dlls which are not included,
you may or may not need to distribute them.

Make sure you have the license if you distribute any of them, and
make sure you don't distribute files belonging to the operating system.

USER32.dll - C:\Windows\system32\USER32.dll
SHELL32.dll - C:\Windows\system32\SHELL32.dll
ADVAPI32.dll - C:\Windows\system32\ADVAPI32.dll
WS2_32.dll - C:\Windows\system32\WS2_32.dll
GDI32.dll - C:\Windows\system32\GDI32.dll
KERNEL32.dll - C:\Windows\system32\KERNEL32.dll

```

Fig.31

Aceste dll-uri sunt ale sistemului de operare, și conform politicii oferite de windows, nu pot fi redistribuite fara acordul lor. Cu siguranta nu vom avea probleme în ceea ce priveste aceste dependente deoarece se regasesc pe orice windows.

În ceea ce priveste directorul dist acesta contine executabilul și toate dependentele necesare pentru a putea rula. Puteti să mutați acest director și să-l redenumiti, dar aveți nevoie de toate dependentele pentru a putea rula corect.

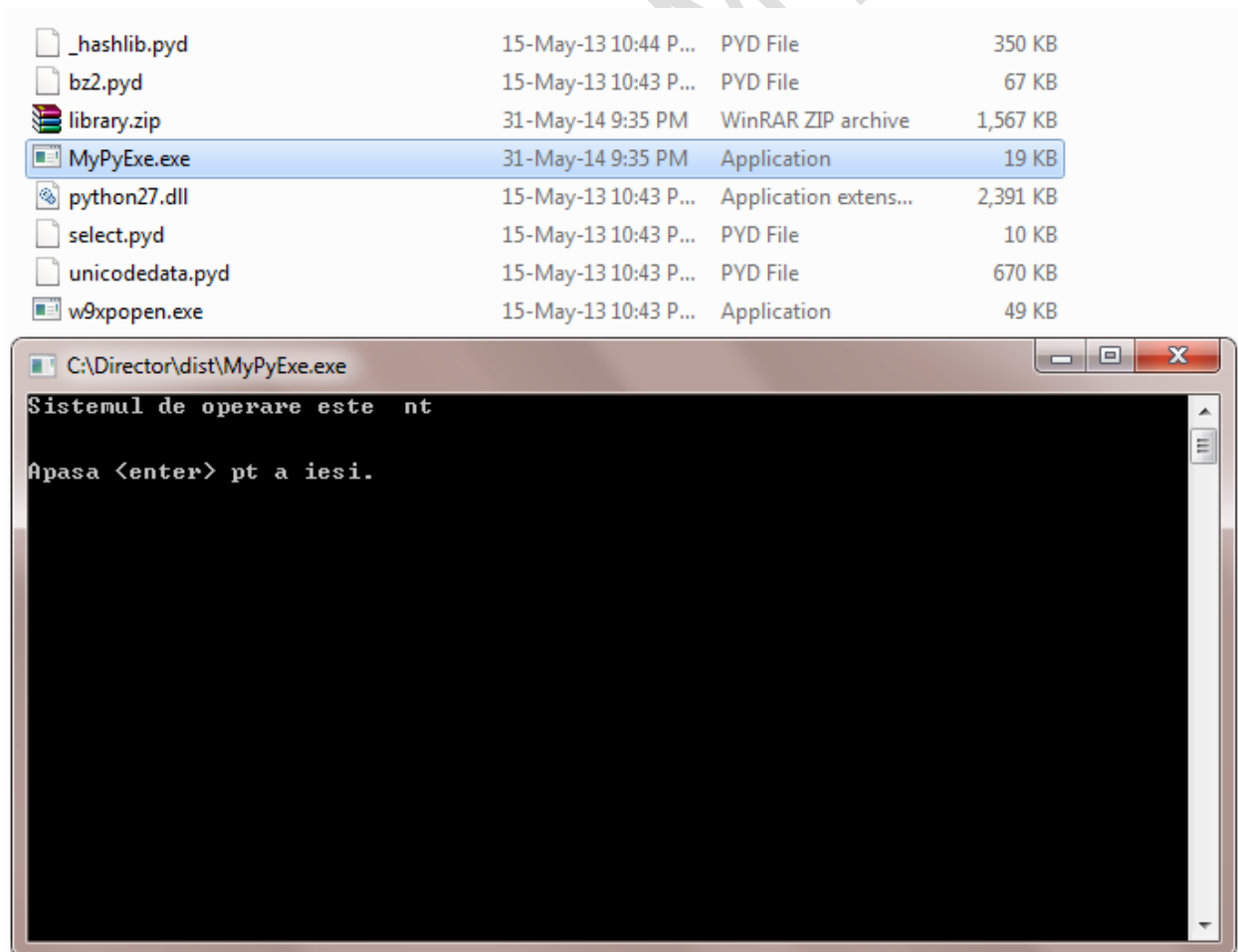


Fig.32

Pagina oficiala pentru modulul py2exe se regaseste mai jos:

<http://www.py2exe.org/index.cgi/FrontPage>

INFACADEMY.NET