# Lucrarea 2

# Python - Sockets

## 1. Elemente de sintaxă

### Excepţii

În cazul întîlnirii unei erori, în mod normal programul se termină cu un cod de eroare. Este posibil ca programul să continue dacă se foloseşte structura *try ... except* . În acest caz, la întîlnirea erorii specificate de *except*, programul execută instrucţiunile respective, fără a se termina.

Următorul program permite detectarea împărţirii la zero:

```
while 1:
     x=int(input('x= '))
     try:
          y=10/x
          print(y)
     except ArithmeticError:
          print('Division by zero')
```

În afară de `ArithmeticError` există multe alte excepţii, vezi documentația online.

### Citirea parametrilor din linia de comandă

Există lista *argv*, similară cu C, în modulul *sys*. argv[0] este numele programului, argv[1] primul parametru, etc. Nu există *argc*, dar este echivalent cu *len(sys.argv)*

***APLICAȚIA A1***

Creați un program în editor care să-și afișeze toți parametrii cu care a fost apelat, ca mai jos (nu puteți rula exemplul interactiv, în interpretor, căci *argv* folosește parametrii din linia de comandă a programului). Rulați programul cu un număr variabil de parametri (0 sau mai mulți):

```
#!/usr/bin/python

import sys

for i in range(len(sys.argv)):
     print(sys.argv[i])
```

## 2. Operaţii pe fişiere

*Vizualizarea fişierelor şi alte operaţii în bash*

Deschideţi un terminal pentru a încerca exemplele următoare, direct la promptul din Linux (nu din Python); comenzile de Linux de mai jos sînt utile pentru a testa operaţiile cu fişiere făcute ulterior în Python:

```
touch f.txt     crează fişier de lg. 0, dacă nu există; îi actualizează
                data şi ora, dacă există deja

cat  >  f.txt   redirectarea ieşirii lui cat în fişier, echivalent cu
                scrierea în fişier
bla bla bla
bla
bla
CTRL-D          încheie cat şi salvează; nu folosiţi CTRL-Z !

less f.txt      afișați conținutul lui f.txt; ieșiți cu q

cat >> f.txt    redirectare cu apendare la sfîrşitul fişierului
alte bla bla
CTRL-D

cat f.txt       verificați că „alte bla bla” s-a adăugat la sfîrșit
```

Observați că comanda *cat* :
- fără parametri, citește din *stdin* – intrarea standard deci tastatura consolei și scrie la *stdout* adică tot consola (ecranul)
- `cat > fisier` nu reprezintă apelarea cu parametru, ci *redirectarea* ieșirii lui cat către fișier
- *cat* cu parametru nume de fişier citeşte din fişier

```
cat             nu se întîmplă nimic, aşteaptă
un test         va fi repetat (copiat la stdout)
alt test
CTRL-C          ieşim, nu avem ce salva

cat f.txt       listare fisier;
less f.txt      altă formă de listare, pagină cu pagină; ieşiţi cu q
more f.txt      altă formă de listare, pagină cu pagină; ieşiţi cu q

tail f.txt          listează ultimele 10 linii
tail -n 2 f.txt     listează ultimele 2 linii
ls -l f.txt         arată detalii
cp f.txt fff.txt    copiere în alt fişier
rm f.txt            ştergere
```

## *Operaţii cu fişiere în Python*

În Python, operaţiile pe fişiere prezintă similitudini cu C:

```
f=open('fisier.txt','w')      # w=write, r=read, a=append, t=text
                              # acum f este un descriptor de fişier
                              # f.xxx sînt metode pentru fişiere
f.write('abcd\n')             # argumentul trebuie să fie şir
f.write('efg\n')              # \n = newline
```

```
f.write('hij\n')
f.close()
```

După rulare, verificaţi din terminalul de Linux conţinutul fişierului, folosind comenzile cat, more sau less!

Metoda *f.read()* citește întreg fișierul într-un șir; pentru a defini fișierul ca fiind text se adaugă 't' :

```
f=open('fisier.txt','rt')
S=f.read()
L=f.readlines()
f.close()

print(S)
print(L)
```

încercați să folosiți cele 2 funcții read() și readlines() în ordine inversă!
*f.readlines()* citește linie cu linie și crează elemente ale unei liste:

```
f=open('fisier.txt','rt')
L=f.readlines()
print(L)
```

Pentru fișiere mari, citirea întregului fișier în memorie, ca mai sus, nu este practică/posibilă. De aceea, se poate citi linie cu linie, folosind construcția specială *for…. in f* , care citește fiecare linie într-un șir:

```
f=open('fisier.txt','rt')
for s in f:
     print(s)
```

Indicație: la citirea linie cu linie, *for* apendează automat \n după fiecare linie citită.

Deschiderea unui fişier este unul din cazurile cele mai frecvente de eroare în timpul funcţionării (*run-time error*), întrucît fişierul poate să nu existe (în cazul citirii) sau să nu existe drept de scriere (în cazul scrierii). Tipul de excepţie în acest caz este IOError.

***APLICAȚIA A2 ****

Creați un program numit *littlemore* care să permită citirea unui fișier aproximativ în același mod ca utilitarele *more* și *less* pre-existente pe sistem (fără toate opțiunile avansate):

- numele fișierului va fi specificat în linia de comandă, alături de numărul de linii de citit o dată, de exemplu ./littlemore fisier.txt 25

- se vor afișa primele 25 de linii, apoi se va aștepta orice tastă pt a afișa următoarele linii, sau tasta q pentru a ieși.

## 3. Network Sockets

Așa cum la nivel local un program poate scrie/citi în/dintr-un fișier, comunicația în rețea se face scriind/citind în/dintr-un *socket*. Un *socket* este un identificator bazat pe următoarele informații: *adresa_IP, protocol, port.*
- adresa IP poate fi oricare din adresele existente pe PC; pt. PC-urile neconectate la rețea, există adresa de *loopback* 127.0.0.1

- protocolul poate fi, de obicei, TCP sau UDP
- portul permite mai multe socket-uri simultane pe acelaşi calculator (de exemplu, pentru a avea aplicaţii de reţea diferite gen mail, web etc, funcţionînd simultan).
În cazul mai general, un socket poate fi folosit şi pentru comunicaţia locală între procese de pe acelaşi calculator, nu prin reţea (nu vom studia acest caz).
Un socket se crează folosind metoda:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

*socket_family* se ia AF_INET pentru comunicaţii prin Internet (implicit)
*socket_type* este protocolul de transport: SOCK_STREAM pt. TCP, SOCK_DGRAM pt. UDP
*protocol* este protocolul de rețea (implicit IP=0)

Modul de inițializare a socket-ului și funcțiile folosite depind, in continuare, de tipul de socket:

a) socket de tip *server* este destinat programelor folosite ca servere, care ascultă si așteaptă conexiuni de la clienți. Se folosesc funcțiile:

*s.bind(host, port)* asociază socket-ul cu adresa (host) si portul (port) dorite
*s.listen()* deschide socket-ul in modul server, la care se vor putea conecta clienții
*s.accept()* așteaptă pînă cînd se conectează un client (este o funcție *blocantă*, se blochează așteptînd conexiunea; există și opțiunea de a face funcția ne-blocantă, vezi pagina de manual).

b) socket de tip *client*: folosește o singură funcţie specifică, pentru a se conecta la server:

*s.connect(host, port)* se conectează la adresa specificată, unde trebuie să asculte un server

c) client și server: se folosesc aceleaşi functii de transfer de date, întrucît ambele părți (client si server) pot transmite si recepționa date

*s.recv()* recepționează date folosind protocolul TCP
*s.send()* transmite date folosind protocolul TCP
*s.recvfrom()* recepţionează date folosind protocolul UDP
*s.sendto()* transmite date folosind protocolul UDP
*s.close()* închide socket, similar cu închiderea fișierelor.
*socket.gethostname()* întoarce numele mașinii locale (nume, nu adresa; vezi *gethostbyname()* mai jos pentru translația nume-adresă).

Lista completă de funcții și metode se găsește in [2].
Diferența între TCP si UDP este mult mai mare decît numele funcției, datele trimise prin TCP (SOCK_STREAM) fiind un *flux* în care TCP face automat retransmisia în caz de eroare, în timp ce prin UDP (SOCK_DGRAM) un pachet de date este o *datagramă* care, în caz de pierdere, nu se retransmite. In functie de aplicație se va folosi una sau alta (avantajul TCP este atenuat de complexitatea mai mare și timpul mai mare necesar retransmisiei).

Un exemplu de server:

```
#!/usr/bin/python

# fisierul server.py

import socket               # modulul necesar
```

```
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)   #alegem TCP
host = ``                    # orice adresa; sintem server
port = 12345                 # portul pe care vom asculta (>1024)
s.bind((host, port))         # acum socketul e asociat

s.listen(5)                  # asteptam max. 5 clienti
while True:
   c, addr = s.accept()      # virgula, caci este un tuplu
   print('Conexiune de la:', addr)
   c.send('Mesaj catre client'.encode())
   c.close()                 # inchide
```

Obs. ca folosim socketul *c* pt clientul curent, care s-a conectat deja, în timp ce socketul *s* așteaptă alți client!
Întrucît programul server va folosi adresa IP a mașini fizice pe care rulează, s-a alocat de la început o adresă vidă ''. Aceasta e o deosebire de bază față de un program client, la care *trebuie* să specificăm adresa serverului la care dorim să ne conectăm!
Funcția *encode()* este necesară în Python 3.x pentru a transforma șirul (indicat prin prezența ghilimelelor) într-o succesiune de octeți.

Un client care se conectează la server:

```
#!/usr/bin/python

# fisierul client.py

import socket

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
host = socket.gethostname() # numele masinii locale
port = 12345                # acelasi ca la server

s.connect((host, port))     # socket si client pe aceeasi masina!
print(s.recv(1024))         # ce s-a primit de la server ?
s.close                     # Close the socket when done
```

Intrucît s-a folosit *aceeași* adresă, clientul si serverul vor fi testate pe *aceeași* masină, în 2 terminale diferite; se salvează ca fișiere executabile și se lansează:

| | |
|---|---|
| `./server.py` | într-un terminal; se lansează primul! |
| `./client.py` | în alt terminal |

***APLICAŢIE A3***

Rulați cele 2 programe ca mai sus, în 2 terminale diferite.

a) ce adresă raportează serverul în mesajul "conexiune de la adresa…" ? încercaţi din terminal comanda ping *n.n.n.n* (*n.n.n.n* este adresa de mai sus). Rulaţi comanda *ifconfig* şi verificaţi că apare o interfaţă cu această adresă.

*Indicaţie:* adresa de *loopback* este utilă, printre altele, şi pentru a testa aplicaţii de reţea pe maşini care nu au nici o interfaţă de reţea activă.

*Observaţie:* în afara adresei, observaţi şi portul clientului. Este un port oarecare, alocat de către sistemul de operare în mod aleator. Observaţi că nu are nici o legătură cu portul serverului pe care l-am alocat noi.

b) observați că șirul afișat de client este prefixat cu un *b* de la "*bytes literal"*, un tip de date diferit de *string*, întrucît implicit prin socket-uri nu se trimit *string*-uri (Python 3)

c) încercaţi să modificaţi serverul să ruleze pe portul 123 (care e <1024). Ce se întîmplă ?

d) mai deschideţi un terminal (al treilea) pentru a investiga serverul d.p.d.v. al sistemului de operare.

Rulaţi comanda:

```
netstat –tn
```

(-t = TCP, -n = *do not translate names*) în timp ce rulează serverul; verificaţi că există serverul nostru în lista de servere, dintre cele care rulează pe diferite porturi.

Verificaţi cu `man netstat` ce alte opţiuni mai are acest program.

***APLICAŢIE A4***

Modificaţi clientul astfel ca adresa serverului la care se conectează să fie luată din linia de comandă, pentru a putea rula, opțional, cele 2 programe *pe mașini diferite*.

- se pornește serverul pe un PC; se află adresa PC-ului folosind comanda *ifconfig* ; se caută adresa

care începe cu 141.85 dacă afișează mai multe adrese

- se porneşte clientul pe alt PC (dacă este disponibil) specificînd în linia de comandă adresa serverului;

În cazul în care nu sînt disponbilie 2 calculatoare, oricum, folosiți în linia de comandă adresa de internet (141.85…), nu cea *localhost.*

*Observație:* după oprirea programului, dacă primiți eroarea *socket.error: [Errno 98] Address already in use* înseamnă că socketul nu a fost încă închis. Puteți verifica cu *netstat* că socketul apare în starea *TIME_WAIT*. Puteți închide folosind *kill* programul, sau puteți aștepta cîteva secunde să se închidă socketul.

***Translaţia numelor DNS***

Functia *socket.gethostbyname(hostname)* face interogarea serverului DNS (folosindu-se de sistemul de operare), și translatează un nume DNS *hostname* (de exemplu matrix.elcom.pub.ro) într-o adresă de tipul 141.85.x.y (întoarsă sub forma unui șir).

***APLICAŢIA A5****

Scrieţi un program care folosește această functie:

- parametrul din linia de comanda sa fie un nume de DNS

- să afișeze adresa IP corespunzătoare numelui

Testați acest program si comparați-l cu programele Linux *dig* sau *nslookup*, care fac aproximativ același lucru (dar cu mult mai multe opțiuni)

***APLICAŢIA A6****

Modificați programele client și server astfel:

- serverul să rămînă conectat ca pînă acum, primind într-o buclă *while True* date sub formă de numere de la client, și adunînd numerele primite la suma anterioară

- clientul să fie apelat cu sintaxa *./client.py număr* , unde numărul să fie transmis serverului

- la recepția fiecărui număr, serverul să-l adune la suma precendentă, și să afișeze mesajul “suma curentă:” și suma corespunzătoare.

Aplicațiile cu asterisc (*) vor fi verificate în vederea notării.

## Bibliografie

[1] Python 3.x online docs: *https://docs.python.org/3*
[2] Python 3.x sockets online doc: *https://docs.python.org/3/library/socket.html*
[3] Python sockets tutorial: *https://realpython.com/python-sockets/*