Lucrarea 1
Elemente de Python

1. Recapitulare Linux CLI (*Command-Line Interface*)

Prescurtări

. folderul (directorul) curent
.. folderul (directorul) părinte al celui curent
~ *home-directory* (folderul de lucru al userului)

comanda *pwd* afișează folderul curent (*print working directory*) dacă acesta nu se vede la prompt.

APLICAŢIE A1

Deschideţi un terminal, experimentați următoarele comenzi:

```
cd ~            vă aflați în home implicit cînd porniți terminalul
cd .            verificați că nu s-a schimbat nimic
cd ..           unde sîntem ?
cd ~            ne întoarcem în home
mkdir test1
cd /test1       va da eroare căci calea este greșită (absolută)
cd test1        așa da; adresare relativă
pwd
cd /            în rădăcină
cd ~test1       va da eroare, ar însemna home-ul userului test1
cd ~/test1      așa da; sînteți în /home/asi/asiN/test1
cd ../../..     adresare relativă: "coborîți" 3 niveluri.
pwd             ar trebui să fiți în /home
```

Permisiuni

schimbarea permisiunilor se face cu chmod:
chmod a+x fișier face un fișier executabil pt toți (all)
chmod g+w fișier face un fișier citibil pt cei din grupul respectiv
chmod o-rw fișier scoate permisiunile de citire și scriere pt cei care nu-s nici proprietar nici grup.

a *all* u *user (proprietar)* g *group* o *others*
\+ *setează permisiunea* - *elimină permisiunea*
r *read* w *write* x *execute*

permisiunea "x" (execuție) este o diferență de fond între Unix și Windows. În Unix, dacă nu dați această permisiune (de obicei, manual) unui fișier, el nu se poate executa, chiar dacă îl puteți citi sau scrie. Deci cînd creăm un fișier care este un program executabil (în cazul Python, fiind interpretat, este totuna fișierul sursă și cel executabil), tb. să facem chmod a+x pe acest fișier.

APLICAŢIE A2

Experimentați următoarele comenzi în terminal

```
cd ~/test1          directorul a fost creat mai sus
touch fis1          touch crează un fisier normal (gol)
touch .fis2         fisier ce incepe cu punct
ls                  cîte fișiere vedeți ?
ls -la              dar acum ? "a" le afișează și pe cele cu punct
chmod u-r fis1
ls -la              ce permisiune a dispărut ?
chmod a+r fis1      punem la loc
./fis1              încercăm să-l executăm; merge ?
chmod a+x fis1      ce am adăugat ?
./fis1              acum merge execuția(dar nu face nimic, căci e gol)
fis1                de ce nu-l execută, deși evident sîntem în .  ?
rm fis1 .fis2       ștergem ce-am creat
cd ~
rm test1            va da eroare
rm -r test1         așa da, pentru foldere
```

Midnight Commander (*mc*) și editorul intern *mcedit*

Pentru administrarea comodă a fișierelor, inclusiv editarea, se poate folosi *mc*
Dacă, în funcție de setările terminalului, tastele de funcții Fn nu merg, se pot înlocui cu ESC n; de exemplu Exit – F10 se înlocuiește cu ESC 0.

CTRL-o alternează între panourile de editare și linia de comandă, fiind o soluție de a suspenda *mc* fără a-l părăsi complet.

Selectarea editorului:

- F9 – PullDown --> Options --> Configuration --> Other options --> Use internal edit ON

Editarea unui fișier: F4; crearea unui fișier nou și deschiderea în editor: SHIFT F4
Editorul se poate apela din afara *mc* folosind *mcedit*

Setări în editor:

- F9 – PullDown --> Options --> General:
  - Tabulation --> Fake Half Tabs OFF
  - Fill Tabs with spaces ON
  - Tab Spacing 3 sau 4
  - Syntax Highlighting ON
- F9 – PullDown --> Options --> Syntax Highlighting --> Python

În editor se poate face Paste (Shift-Ins) cu material copiat din alte surse
Copy-Paste local se face cu F3 (marcare) și F5/F6 (Copy-Paste/ Cut-Paste)

Copy-Paste în Linux în general (nu *mc* în special): **Copy: CTRL-Insert Paste: Shift-Insert**

De asemenea, direct în terminal (nu în editor) Copy-Paste se mai poate face astfel, dacă dorim să preluăm o porțiune de text deja afișată pe ecran (de exemplu, rezultatul produs de un program)

```
ms@matrix:~$ ls test
examples.desktop  profil  test1  test2
ms@matrix:~$
```

dacă dorim să preluăm text „test1 test2", îl marcăm cu mouse-ul ținînd butonul stîng apăsat:

```
ms@matrix:~$ ls test
examples.desktop  profil  test1  test2
ms@matrix:~$
```

apoi copiem *la locul unde este cursorul*, făcînd click simultan stînga+dreapta:

```
ms@matrix:~$ ls test
examples.desktop  profil  test1  test2
ms@matrix:~$  test1  test2
```

Observație: în această lucrare puteți folosi orice editor doriți, mcedit este doar un exemplu.
Dacă se intră din greșeală în alte editoare, modalitatea de ieșire este:
joe: CTRL K Q
vi: ESC :q!

2. Elemente de sintaxă Python

Vom studia în cele ce urmează elemente ale limbajului Python, versiunea 3.x.
Pentru referinţa completă asupra limbajului, conţinînd numeroase exemple, folosiţi [1].

Variabile:
Spre deosebire de C nu se declară tipul, acesta se recunoaşte după iniţializarea/prima folosire a variabilei;

Din terminal, executați interpretorul `python` și încercați exemplele de mai jos:

```
a=1             # intreg
print(a)
a="abc"         # sir
a='abc'         # tot sir
a="That's OK"   # solutie pentru folosirea apostrofului simplu
print(a)
print(len(a))   # lungimea sirului
print a[1]

a=1             # il facem din nou intreg
print(a[1])     # va genera eroare caci acum a e intreg
type(a)         # verificati!
a=3.14
type(a)         # verificati!
```

Nespecificarea tipului poate crea confuzii, de exemplu între *int* şi *float*:

```
a=1
type(a)
a=a/2
print(a)        # verificaţi !

a=1.0           # fortam float
type(a)
a=a/2
print(a)        # verificati !

a=1.0
a=a//2          # forteaza impartirea intreaga
print(a)
a               # direct în interpretor se poate omite print
```

Tupluri:
O variabilă cu mai multe elemente (similară unui vector) formează un *tuplu*. Dimensiunea unui tuplu poate varia:

```
x = (1,2)
y = (1,2,3,4)

print(x)
print(y[1]) # elementul n dintr-un tuplu se accesează cu []
```

afișarea unui tuplu se face întotdeauna cu paranteze, dar asignarea se poate face și fără:

```
x= 1,2,3
x
len(x)    # lungimea (dimensiunea) tuplului
```

de asemenea, un tuplu poate conține orice tipuri de date, inclusiv alte tupluri:

```
y = x, 'bla', 5
y
```

Un tuplu se poate redefini cu totul (precum x mai sus) dar elementele unui tuplu sînt "*immutable"* (fixe)

```
x[0] = 2        # va da eroare
```

Blocuri:
Blocurile de instrucțiuni nu se includ între {} sau *begin... end* ci se *indentează* cu un număr egal de spaţii

```
if x >= 1:
    print('x supraunitar')
    if y >= 1:
       print('x si y supraunitare')
```

OBS: pentru a preveni problemele, configurați editorul să insereze spații, nu TAB-uri, la apăsarea tastei TAB.

Structura de control *if-elif-else:*

```
if y>=1:
   print('y supraunitar')
elif y>0:
   print('y pozitiv')
else:
   print('y negativ')
```

Structura de control *while:*

```
## acceseaza fiecare al treilea caracter intr-un sir:
a = 'abcdefghijkl'
i = 0
while i < len(a):
    print(a[i])
    i = i + 3
```

OBS: *break* și *continue* funcționează la fel ca în C

Funcţii:
Definim 2 funcţii, *repeat()* şi *main()*; apelăm apoi funcția *main().*

APLICAŢIE A3

- Deschideţi un terminal, lansaţi un editor

scrieţi codul de mai jos în editor, salvaţi cu numele *test.py* , dar *nu închideţi* editorul, pentru a putea face modificări după dorinţă cf. exemplelor următoare.

OBS: pentru a putea fi apelat direct cu numele, prima linie trebuie să fie:

```
#!/usr/bin/python
```

a.î. sistemul de operare să știe cărui program executabil îi e asociat fișierul (e valabil şi pt. /bin/bash, etc)

- Deschideţi *o fereastră adiţională* de terminal, în care rulaţi programul folosind `./test.py` sau în Windows `python test.py` . Terminalul inițial rămîne pentru modificări în cod.

OBS: doar fișierele executabile pot fi executate în Linux, trebuie deci să-i schimbați permisiunile folosind:

```
chmod a+x test.py
```

```
#!/usr/bin/python

def repeat(s):
    #Returns the string 's' repeated 3 times.
    #If exclaim is true, add exclamation marks.
    result = s + s + s    # can also use "s * 3" which is faster
    return result

def main():
    print(repeat('Xyz'))
```

```
    print(repeat('Bla bla'))

main()
```

OBS: spre deosebire de C, existenţa unei funcţii *main()* este opţională.

Module:
Conţin grupuri de funcţii, metode etc. De exemplu, *sys* pentru a avea acces la *argv*:

```
# import modules used here -- sys is a very standard one
import sys

def main():
    print('Hello world ', sys.argv[1])
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

main()
```

APLICAŢIE A4

Deschideţi un terminal, lansaţi interpretorul python

listaţi funcţiile disponibile în modulul sys

```
import sys
```

```
help(sys)
```

listaţi toate funcţiile cu:

```
dir(sys)
```

părăsiţi interpretorul cu `exit()`

3. Manipularea şirurilor

Elementele unui şir se accesează cu [] similar cu C. Python este un limbaj de tip OOP, doi operatori care sînt *overloaded* fiind **+** (concatenează şirurile) și **==** (compară șiruri).

```
s = 'hello'
print(s)
print(s[1])
s = s + ' there'    # hello there
if s=='hello there':
     print('da, merge concatenarea')

s = 'hello'
print(s[1:4])  # o "felie" dintr-un șir (string slice)
print(s[1:100])     # se trunchiază fără erori!
print(s[100])  # aici nu merge!
print(s[:1])   # este similar cu s[0:1]
print(s[1:])
```

```
print(s[-1])    # de la coadă la cap
print(s[-5])    # idem; de la cap la coadă Hello are poziţiile 0...4
                # invers are -1...-5
print(s[:-3])
print(s[-3:])
```

Se observă că s[:n] + s[n:] = s, pentru orice n

Șiruri sau alte construcţii mai lungi:

```
s="""acesta este          # triple quote pentru a depasi o linie
un sir lung"""

s1='''acesta este         # alta varianta de triple quote
alt sir lung'''

print(s,s1)
print(s,'\n',s1)              # \n la fel ca în C
```

OBS: Este nevoie de triple quote sau alte metode de a informa explicit interpretorul că s-a depăşit o linie, întrucît Python este bazat pe linii – acesta este motivul pentru care *nu* se foloseşte caracterul " ; " sau alt delimitator. O linie nouă este un delimitator în sine. Altă metodă de a trece cu aceeaşi comandă pe o linie nouă:

```
s="un al treilea sir \
foarte lung"
print(s)
```

Altă asemănare cu C: se pot folosi modificatorii din *printf* (%d %f %s ...) pentru a compune un şir; observaţi însă şi deosebirea: valorile care vor fi inserate dinamic în şir formează un *tuplu* (în acest caz de lungime 4) adăugat la sfîrşit cu sintaxa:

```
s = "Cei %d apostoli erau %d: %s si %s" %(4, 3, "Luca", "Matei")
print(s)
```

Citirea unui șir de la *stdin* (implicit tastatură)

functia *input([prompt])* afișează *prompt*-ul opțional și întoarce rezultatul sub formă de șir:

```
s = input('Introduceti codul ')
print('Codul este ', s)
```

Funcţii şi metode

O *funcţie*, scrisă *functie(var1, var2... varN)* se poate aplica oricărei variabile de tipul corespunzător.
O *metodă*, scrisă *variabila.metoda(var1, var2,... varN)* este o funcţie care se aplică doar variabilelor membre ale unei clase (concept OOP).

Şirurile sînt membre ale clasei *str* şi există o multitudine de metode care pot fi folosite asupra lor – vedeţi lista lor completă în [2]. Cîteva exemple:

`s.lower(), s.upper()` – converteşte în *lowercase/uppercase*
`s.isalpha()/s.isdigit()/s.isspace()...` – testează dacă e de tipul respectiv
`s.replace('old', 'new')` – întoarce un şir în care 'old' e înlocuit de 'new'

`s.split('delim')` – întoarce o *listă* de subşiruri separate de delimitatorul dat.
Exemplu:
`'aaa,bbb,ccc'.split(',') -> ['aaa', 'bbb', 'ccc']`
Observaţie: vezi definiţia conceptului de "listă" mai jos.
Caz particular: `s.split()` (fără argumente)
splitează acolo unde întîlneşte spaţii.

`s.join(list)` – opusul lui `split()`, reuneşte şirurile din listă folosind delimitatorul dat
Exemplu:
`'---'.join(['aaa', 'bbb', 'ccc']) -> aaa---bbb---ccc`

OBS: şirurile în Python sînt fixe (engl. *immutable*), adică nu este permisă modificarea valorii unui şir (similar cu tuplurile). În consecinţă, nu este permisă inversarea elementelor, de genul s[1]=s[2]. De aceea, veţi folosi o listă, nu un şir, pentru astfel de operaţii.
Pentru transformarea şirului S în lista L se poate folosi L=list(S).

4. Liste; structura de control *in*

O listă este o structură de tipul *list* care prezintă multe similitudini cu şirurile și tuplurile: lungimea se determină cu *len()*, componentele se accesează cu [], începînd de la indexul 0.

```
colors = ['red', 'blue', 'green']
print(colors[0])    ## red
print(colors[2])    ## green
print(len(colors))  ## 3
```

Ca şi în cazul şirurilor, operatorul + realizează apendarea:

```
L1=[1,2]
L2=[3,4]
print(L1+L2)
```

Similar cu şirurile, se pot defini domenii (*list slices*) de tipul L[1:2] sau similar.

```
L=[0,1,2,3,4,5]
L[:2]                # incercati !
L[2:]
L[-1]
L[:-2]
```

Structura de control *for*

Spre deosebire de C, gama în care se iterează variabila de control nu se scrie direct, ci se defineşte în prealabil o listă sau un *range:*

```
squares = [1, 4, 9, 16]
sum = 0
```

```
for num in squares:
    sum += num
print(sum)          ## 30
```

structura de control *in* se poate folosi şi în afara *for*:

```
list = ['alfa', 'beta', 'gama']
if 'beta' in list:
    print('membru al listei')
```

O altă variantă de *for* este *for...in range*

```
## printează numerele 0..9
for i in range(10):
    print(i)
```

puteţi încerca rezultatul lui *range* folosit singur:

```
range(10)
range(5,10)
range(0,10,3)
```

Metode pentru liste: se pot găsi aici [3] referinţele la toate metodele disponibile.

Ştergerea variabilelor

del permite ştergerea de variabile, inclusiv la nivel de element din listă (nu şi de şir – reamintim că şirurile sînt *immutable* – fixe – şi nu se pot şterge decît cu totul)

```
a = 6
del a
print(a)       # a dispărut orice referinţă la a

L=[0,1,2,3,4,5,6]   # listă
print(len(L))
print(L[3])

del L[2]       # ştergeţi un element
print(len(L))  # încercaţi !
print(L[3])    # s-a modificat?
del L[-2:]     # ce s-a şters ?

sir='abcd'
print(sir[1])
del sir[1]     # încercaţi !
del sir
```

APLICAŢIE A5*

Scrieţi un program (folosind editorul) care:

- să ruleze într-o buclă infinită (vezi `while`)

- să aştepte un şir de la utilizator

- să verifice că şirul corespunde unei adrese de mail (să conţină semnul @)

- să printeze separat username-ul şi domeniul.

APLICAŢIE A6*

Scrieţi un program (folosind editorul) care:

- să preia un şir de la utilizator (şirul va conţine doar cifre, de exemplu 11287436523)

- să folosească metoda *bubble sort* pentru a sorta cifrele din şir

- să afişeze rezultatul, format din cifrele sortate

OBS1: metoda *bubble sort* [4] este o metodă de sortare simplă care funcţionează astfel:

- se parcurge şirul în ordine, de mai multe ori

- pentru fiecare 2 elemente adiacente, se verifică dacă-s în ordine, dacă nu se inversează între ele

- dacă în cadrul unei parcurgeri nu s-a mai făcut nici o inversare, înseamnă că şirul e sortat şi se poate încheia.

OBS2: deoarece şirurile în Python sînt fixe (engl. *immutable*), nu este permisă inversarea elementelor, de genul s[1]=s[2]. De aceea, veţi folosi o listă, nu un şir, pentru operaţia de sortare.

Pentru transformarea şirului S în lista L se poate folosi `L=list(S)`. La sfîrşit, construiţi şirul cifră cu cifră (eventual cu o construcţie de genul `for i in L` ...)

APLICAŢIE A7*

Scrieţi un program (folosind editorul) care:

- să ruleze într-o buclă infinită (vezi while)

- să aştepte un număr de la utilizator

- pentru toate numerele mai mici decît numărul respectiv, să afişeze descompunerea în factori (minim 2 factori), sau faptul că numărul este prim. Indicaţie: folosiţi *range*.

Observaţie: *doar aplicaţiile marcate cu * vor fi notate*

Bibliografie

[1] Python 3.x online docs: *https://docs.python.org/3*
[2] Python string methods: *https://docs.python.org/3/library/stdtypes.html#string-methods*
[3] Python lists methods: *https://docs.python.org/3/tutorial/datastructures.html*
[4] președintele Barack Obama despre Bubble sort: https://youtu.be/1nnj7r1wCD4?t=426