Mediu virtual în Python. Folosirea funcțiilor și a modulelor în Python Recursivitate

1. Crearea unui mediu virtual pentru lucrul folosind limbajul python

Pe orice calculator putem crea un mediu virtual (virtual environment) pentru a lucra cu **Python**. În felul acesta putem lucra cu diferite versiuni de **Python** și instala pachetele necesare.

Crearea unui mediu virtual pentru **Python** se face folosind comanda (mediu virtual se va numi **MD** în exemplul nostru):

```
[elena@Elenas-iMac ~ % python3 -m venv MD
 elena@Elenas-iMac ~ %
```

Activarea acestui mediu virtual MD se face:

```
[elena@Elenas-iMac ~ % source MD/bin/activate
[(MD) elena@Elenas-iMac ~ %
[(MD) elena@Elenas-iMac ~ %
 (MD) elena@Elenas-iMac ~ %
```

Dezactivarea mediului:

```
[(MD) elena@Elenas-iMac ~ % deactivate
[elena@Elenas-iMac ~ %
 elena@Elenas-iMac ~ %
```

2. Folosirea funcțiilor și a modulelor în Python

Ca și în limbajul **C** putem împărți un program **Python** în module. Pentru o mai bună organizare a programelor scrise, putem avea un modul care conține funcții care implementează prelucrările ale datelor. Fișierul în care se găsesc funcțiile se numește **modul**, iar funcțiile scrise sunt accesibile în fișierul care conține funcția **main** prin importarea modulului corespunzător. Dacă presupunem că funcțiile scrise se găsesc în fișierul **functii.py**, atunci în fișierul care conține funcția **main** trebuie importat acest modul. Se poate importa un modul în întregime sau doar anumite funcții care ne interesează (vezi și exemplele din **Curs nr. 1**).

Caz 1 – ***Importarea unui modul folosind numele fișierului în care se găsește modulul***

```
import functii

def main():
      ..............
      rez=functii.cmmdc(a, b)
            # Exemplu de folosire a funcției definite în modulul functii
      ......................

if __name__ == '__main__':
      main()
```

Caz 2 – ***Importarea unui modul cu un nume stabilit de programator***

```
import functii as fct

def main():
      ..............
      rez=fct.cmmdc(a,b)
            # Exemplu de folosire a funcției definite în modulul functii
      ......................

if __name__ == '__main__':
      main()
```

Caz 3 - ***Importarea unei singure funcții dintr-un modul***

```
from functii import cmmdc

def main():
      ..............
      rez=cmmdc(a,b)  # Exemplu de folosire a funcției definite în modulul functii
      ......................

if __name__ == '__main__':
      main()
```

Caz 4 - ***Importarea mai multor funcții dintr-un modul***

```
from functii import cmmdc, valMax

def main():
      ..............
      #In acest caz am importat doar funcțiile cmmdc și valMax
      rez=cmmdc(a,b)
      r=valMax(c, d)
      ......................

if __name__ == '__main__':
      main()
```

3. Recursivitate

Recursivitatea are drept corespondent în matematică relația de recurență.

O funcție este recursivă dacă se autoapelează înainte de a se reveni din ea. Procesul în care o funcție se apelează pe ea însăși poartă numele de recursivitate. La fiecare apel al unei funcții adresa de retur, parametrii și variabilele locale sunt stocați pe stivă. Atunci când se revine dintr-o funcție, zona de pe stivă folosită pentru a memora variabilele locale, parametrii reali (parametrii de apel) și adresa de retur este eliberată.

Pentru a ilustra modul în care funcționează recursivitatea vom lua ca exemplu problema factorialului. În cazul unei funcții recursive de fiecare dată trebuie sa avem o condiție de stop (caz de bază). Dacă evităm acest aspect, funcția nu se va sfârși niciodată. Pentru problema de față condiția de stop este factorial de 0, care prin definiție este 1. Pentru a face o comparație între funcțiile recursive și cele nerecursive, vom rezolva problema factorialului și în mod clasic.

Implementarea celor două funcții în **Python** se va face într-un modul numit **factorial**.

Modulul **factorial**. Implementare nerecursivă și recursivă pentru calculul factorialului

```
def factorialIterativ( n ):     #implementare nerecursivă
      f=1
      for i in range(1, n+1):
            f=f*i
      return f

def factorialRecursiv( n ):     #implementare recursivă
      if n == 0:
            r = 1
      else:
            r = n * factorialRecursiv( n - 1 ) #adr2
      return r
```

Program 1. Program de calcul al factorialului folosind varianta nerecursivă:

```
import factorial as F

def main():
      x = input("Numarul pentru care doriti sa calculati factorialul: ")
      a = F.factorialIterativ(int(x))
      print(a) #adr1

if __name__ == "__main__":
      main()
```

Știm că la apelul unei funcții, variabilele locale și parametrii funcției sunt depuși pe stivă. În Figura 1 este prezentată folosirea stivei în cazul implementării nerecursive a funcției de calcul al factorialului, iar în Figura 2 este cazul implementării recursive.

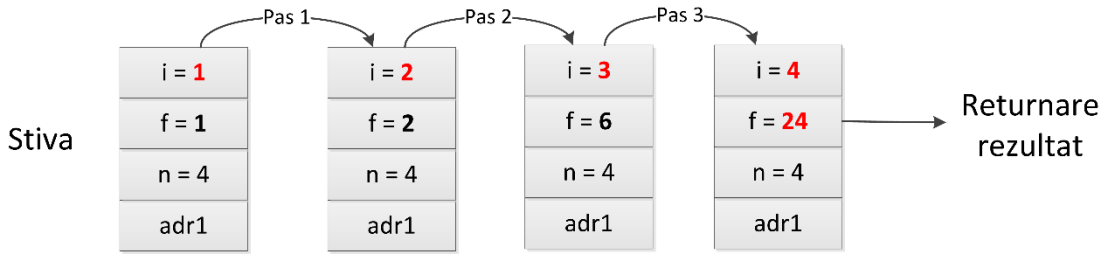

Figura 1. Folosirea stivei în cazul implementării nerecursive

Program 2. Calculul recursiv al factorialului

```
import factorial as F

def main():
      x = input("Numarul pentru care doriti sa calculati factorialul: ")
      a = F.factorialRecursiv(int(x))
      print(a) #adr1

if __name__ == "__main__":
      main()
```

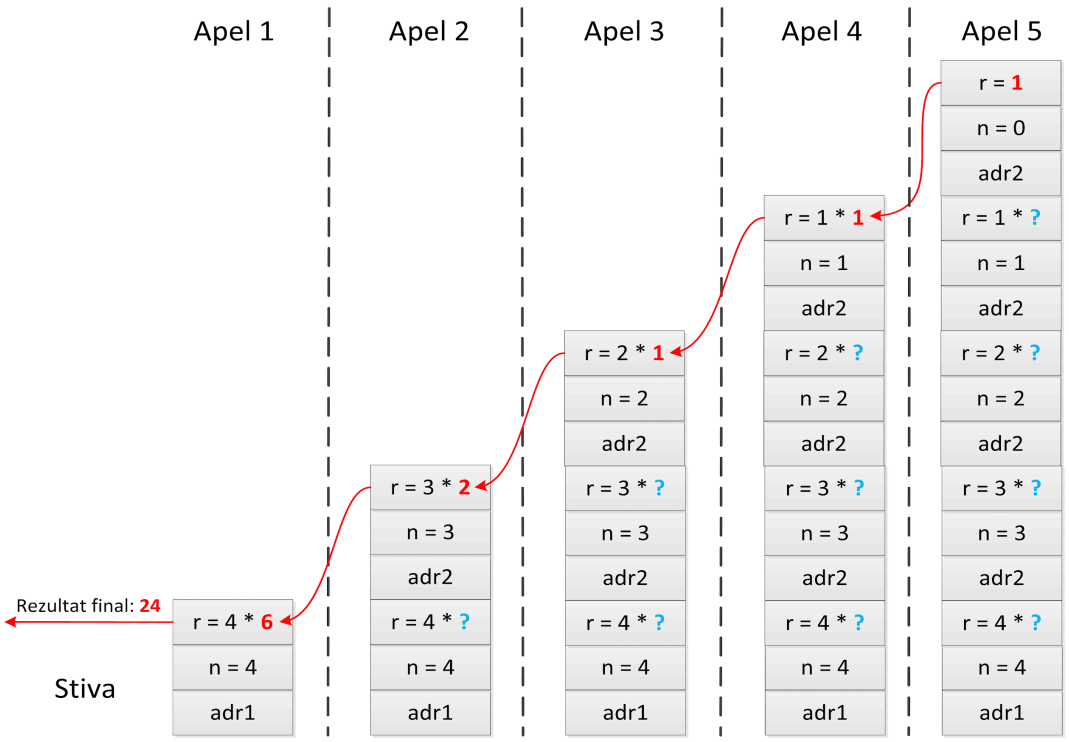

Figura 2. Folosirea stivei în cazul implementării recursive recursiv

Apelurile recursive pot conduce frecvent la depășirea stivei. De cele mai multe ori recursivitatea nu conduce nici la economie de memorie și nici la execuția mai rapidă a programelor. Funcțiile recursive sunt mai lente decât cele nerecursive deoarece la fiecare apel de funcție se depune pe stivă adresa de revenire și valorile parametrilor. Avantajul recursivității constă în faptul că ne permite o descriere mai compactă a funcțiilor.

TEME

Tema 1 (pentru fiecare din problemele următoare se va scrie și programul care testează funcțiile)

1. Să se determine formula de recurență pentru calculul factorialului și să se implementeze o funcție recursivă pentru aceasta (se va implementa exemplul din laborator).
2. Să se determine formula de recurență și să se implementeze recursiv ridicarea numărului n la puterea k (recursivitatea se va face după **k**).

3. Să se determine recursiv al n-lea termen din șirul lui Fibonacci (primii doi termeni din șir se vor considera a fi 1 și 2).
4. Să se determine formula de recurență și să se scrie o funcție recursivă pentru calculul aranjamentelor de n obiecte luate câte k (recursivitatea se va face după **k**).

Tema 2

1. Să se implementeze calculul valorii unui polinom într-un punct folosind o funcție recursivă.

Observație:

Pentru polinomul:

$$P_n(x) = a_0 \cdot x^n + a_1 \cdot x^{n-1} + \cdots + a_{n-1} \cdot x + a_n$$

Relația de recurență este:

$$P_n(x) = \begin{cases} a_0 & n = 0 \\ P_{n-1}(x) \cdot x + a_n & n > 0 \end{cases}$$

iar coeficienții polinomului sunt memorați într-o listă citită de la tastatură (vezi **Anexa**).

2. Să se scrie o funcție recursivă pentru calculul produsului scalar a doi vectori de întregi, fiecare cu câte *n* elemente.

3. Fiind dat un șir de numere întregi să se scrie funcții recursive pentru:
   a. determinarea numărului de valori negative din șir
   b. determinarea produsului tuturor valorilor elementelor din șir
   c. determinarea sumei tuturor valorilor elementelor din șir
   d. determinarea produsului valorilor elementelor negative din șir
   e. determinarea valorii minime din șir
   f. determinarea valorii maxime din șir

   Să se scrie un program **Python** care citește un șir de numere întregi și afișează valorile returnate de funcțiile scrise pentru subpunctele a – f.

4. Să se scrie o funcție recursivă pentru determinarea celui mai mare divizor comun (cmmdc) a două numere întregi pozitive folosind algoritmul lui Euclid.
   Să se scrie o funcție care determină cmmdc al elementelor unui vector de numere pozitive.

Tema 3

1. Să se scrie un program care folosește o funcție recursivă pentru calculul numărului cifrelor unui număr natural.
2. Să se scrie un program care folosește o funcție recursivă pentru calculul produsului cifrelor unui număr natural.

Anexa

Funcții pentru citirea unei liste cu elemente de tip întreg de la tastatură (se găsesc în fișierul **vector.py**):

```
#se cunoaște numărul de elemente din colecție
def citireLista1( v:list, n:int ):
     for i in range(0,n):
          print('v[%d]=' % i, end=' ')
          x=int(input())
          v.append(x)

#nu se cunoaște a priori numărul de elemente din colectie
def citireLista2( v:list ) -> int:
     n = 0
     while True:
          try:
               x = int(input('x = '))
               v.append(x)
               n = n + 1
          except EOFError:
               print('\nAti terminat de introdus valorile! ')
               break

     return n
```

Program 3. Folosirea funcțiilor pentru citirea unei liste cu elemente de tip întreg de la tastatură și afișarea acesteia:

```
import vector as V

def main():
     v1=[]
     n=int(input('Introduceti nr. de elemente:'))
     V.citireLista1(v1,n)
     print("Vectorul citit este:")
     print(v1) #afisare vector

     v2=[]
     n = V.citireLista2(v2)
     print("Vectorul citit este:")
     print(v2)

if __name__ == "__main__":
     main()
```