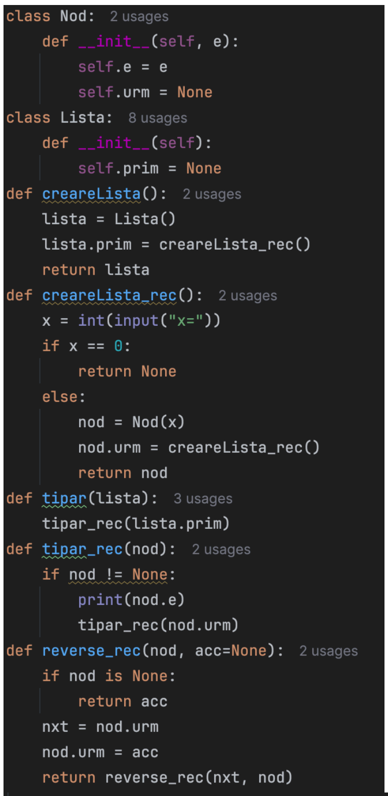
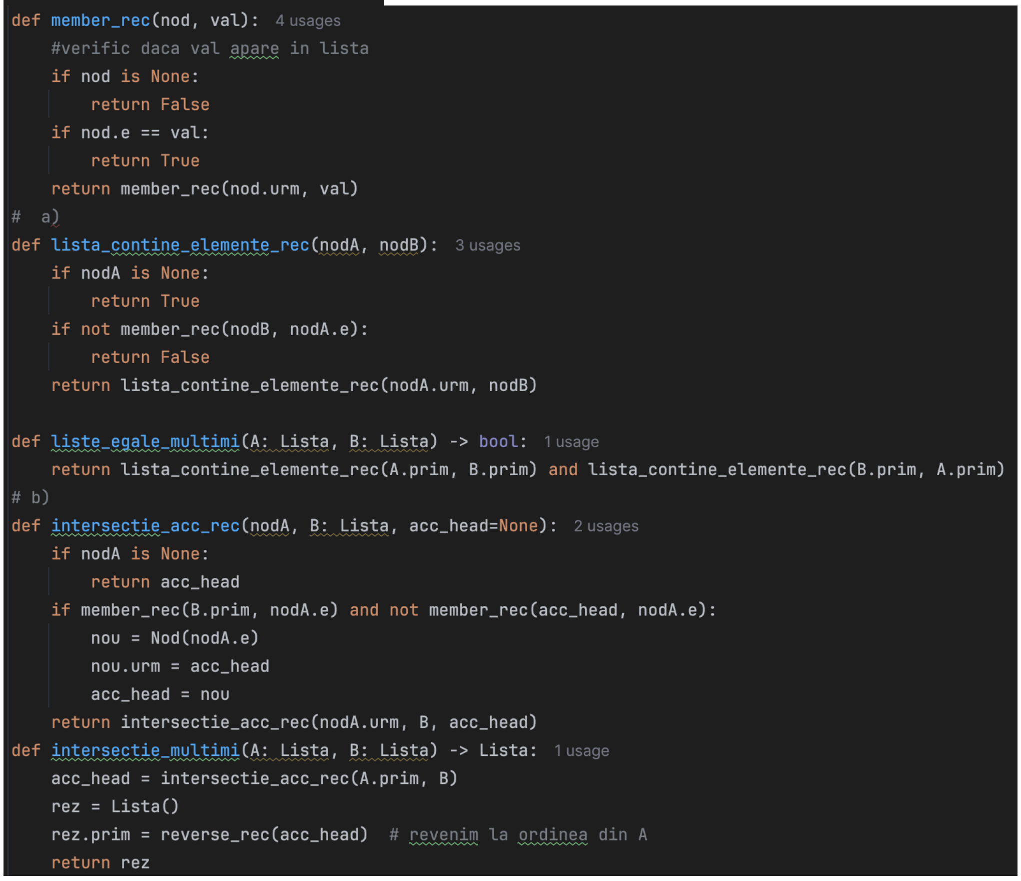
Laborator1 ———> R1

Write a recursive program (Python or C++ programming language) for the following requirements. You can use and extend for implementation the C++/Python model available in /Lab/R1, this model already containing recursive implementations for creating, printing and destroying of a **List.**
It is mandatory to work with a structure/class **List.**
For a **List** will be used a linked representation. Do not use containers from STL or predefined operations on lists in Python (append, len, slicing, etc.).

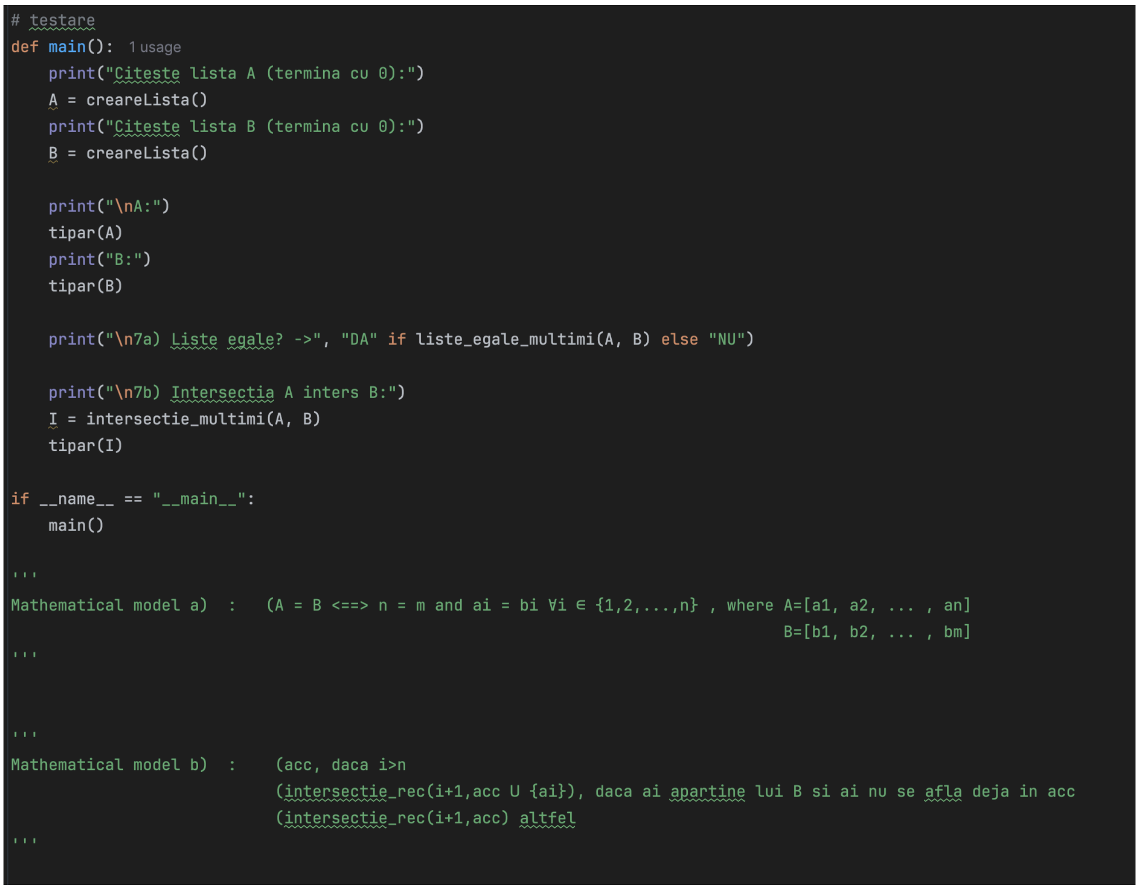
7. a. Test the equality of two lists.
b. Determine the intersection of two sets represented as lists.



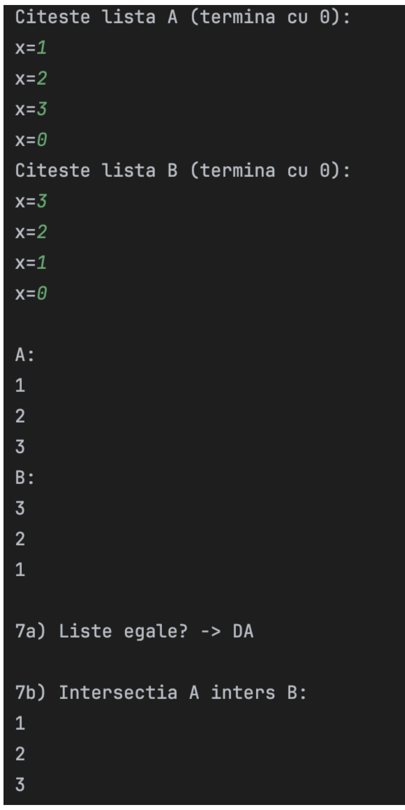
```
class Nod:  2 usages
    def __init__(self, e):
        self.e = e
        self.urm = None
class Lista:  8 usages
    def __init__(self):
        self.prim = None
def creareLista():  2 usages
    lista = Lista()
    lista.prim = creareLista_rec()
    return lista
def creareLista_rec():  2 usages
    x = int(input("x="))
    if x == 0:
        return None
    else:
        nod = Nod(x)
        nod.urm = creareLista_rec()
        return nod
def tipar(lista):  3 usages
    tipar_rec(lista.prim)
def tipar_rec(nod):  2 usages
    if nod != None:
        print(nod.e)
        tipar_rec(nod.urm)
def reverse_rec(nod, acc=None):  2 usages
    if nod is None:
        return acc
    nxt = nod.urm
    nod.urm = acc
    return reverse_rec(nxt, nod)
```



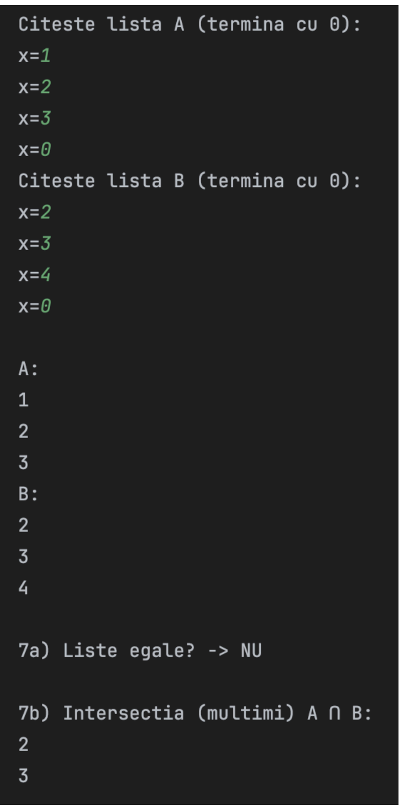
```
def member_rec(nod, val):  4 usages
    #verific daca val apare in lista
    if nod is None:
        return False
    if nod.e == val:
        return True
    return member_rec(nod.urm, val)
#  a)
def lista_contine_elemente_rec(nodA, nodB):  3 usages
    if nodA is None:
        return True
    if not member_rec(nodB, nodA.e):
        return False
    return lista_contine_elemente_rec(nodA.urm, nodB)

def liste_egale_multimi(A: Lista, B: Lista) -> bool:  1 usage
    return lista_contine_elemente_rec(A.prim, B.prim) and lista_contine_elemente_rec(B.prim, A.prim)
# b)
def intersectie_acc_rec(nodA, B: Lista, acc_head=None):  2 usages
    if nodA is None:
        return acc_head
    if member_rec(B.prim, nodA.e) and not member_rec(acc_head, nodA.e):
        nou = Nod(nodA.e)
        nou.urm = acc_head
        acc_head = nou
    return intersectie_acc_rec(nodA.urm, B, acc_head)
def intersectie_multimi(A: Lista, B: Lista) -> Lista:  1 usage
    acc_head = intersectie_acc_rec(A.prim, B)
    rez = Lista()
    rez.prim = reverse_rec(acc_head)  # revenim la ordinea din A
    return rez
```



```
# testare
def main():  1 usage
    print("Citeste lista A (termina cu 0):")
    A = creareLista()
    print("Citeste lista B (termina cu 0):")
    B = creareLista()

    print("\nA:")
    tipar(A)
    print("B:")
    tipar(B)

    print("\n7a) Liste egale? ->", "DA" if liste_egale_multimi(A, B) else "NU")

    print("\n7b) Intersectia A inters B:")
    I = intersectie_multimi(A, B)
    tipar(I)

if __name__ == "__main__":
    main()

'''
Mathematical model a)  :   (A = B <==> n = m and ai = bi ∀i ∈ {1,2,...,n} , where A=[a1, a2, ... , an]
                                                                                B=[b1, b2, ... , bm]
'''



'''
Mathematical model b)  :    (acc, daca i>n
                            (intersectie_rec(i+1,acc U {ai}), daca ai apartine lui B si ai nu se afla deja in acc
                            (intersectie_rec(i+1,acc) altfel
'''
```



```
Citeste lista A (termina cu 0):
x=1
x=2
x=3
x=0
Citeste lista B (termina cu 0):
x=3
x=2
x=1
x=0

A:
1
2
3
B:
3
2
1

7a) Liste egale? -> DA

7b) Intersectia A inters B:
1
2
3
```



```
Citeste lista A (termina cu 0):
x=1
x=2
x=3
x=0
Citeste lista B (termina cu 0):
x=2
x=3
x=4
x=0

A:
1
2
3
B:
2
3
4

7a) Liste egale? -> NU

7b) Intersectia (multimi) A ∩ B:
2
3
```