**LABORATORUL 3**

**SORTĂRI**

1.a) Se citește o propoziție cu cuvinte separate prin spațiu. Să se formeze o nouă propoziție cu cuvintele din prima propoziție care au lungime cel puțin 2 ordonate descrescător lexicografic.

a)citim propozitia si ii facem split; sortare lexicografica descrescator  
  
***#filtrare ls cu append***  
  
s=input()  
ls=s.split()  
ls2=[]  
**for** cuv **in** ls:  
 **if** len(cuv)>=2:  
 ls2.append(cuv)ls2.sort(reverse=**True**)  
ls2 =**" "**.join(ls2)  
print(ls2)

***#filtrare cu comprehension***

ls2=[i for i in input().split() if len(i)>=2]   
ls2.sort(reverse=True) *#modifica lista initiala, sorted returneaza alta lista*  
aux=**" "**.join(ls2) *#lipeste cuvintele din lista intr-o propozitie*  
print (aux)

**sau:**

ls = **" "**.join(sorted([cuv **for** cuv **in** input().split() **if** len(cuv) >= 2], reverse=**True**))  
print(ls)

b) Aceeași cerință ca la a), dar cu cuvintele ordonate descrescător după lungime.  
#b) sortare dupa lungimea cuvintelor  
   
print(**" "**.join(sorted([cuv **for** cuv **in** input().split() **if** len(cuv) >= 2], key=len, reverse=**True**)))

2. Se citesc un număr natural n și următoarele informații despre n persoane: cod de cel mult 3 cifre, nume (un singur nume), prenume (un singur prenume), vârsta (informația despre o persoana este dată pe o linie). Să se creeze o listă de tupluri cu 3 elemente cu informațiile despre cele n persoane și să se afișeze lista cu print.

3

100 Ana Maria 5

5 Ion Ionel 12

80 Mihai Marin 7

n=int(input())  
persoane=[]  
**for** i **in** range(n):  
 s=input()  
 ls=s.split() *#ls are 4 elemente: cod, nume, prenume, varsta - de tip str* t=(int(ls[0]), ls[1]+ **" "**+ls[2] ,int(ls[3]))  
 *#t = (int(ls[0]), " ".join(ls[1:3]), int(ls[3]))  
 #t = (int(ls[0]), f"{ls[1]} {ls[2]}", int(ls[3]))* persoane.append(t)  
print(persoane)

**sau:**

n=int(input())  
l=[]  
**for** i **in** range(n):  
 s=input()  
 cod,nume,prenume,varsta=s.split()  
 nume=nume+**" "**+prenume  
 t=(int(cod),nume,int(varsta))  
 l.append(t)  
print(l)

a)Să se afișeze persoanele ordonate după cod și, în caz de egalitate, după nume (pentru o persoană se va afișa codul pe 4 caractere și numele)

persoane.sort()  
 for p in persoane:  
 *#print("{:4}: {}".format(p[0],p[1]))*  
print(**f"**{p[0]:**4**}**:** {p[1]}**"**)

b) Să se afișeze persoanele ordonate după nume și, în caz de egalitate, după cod (pentru o persoană se va afișa codul pe 4 caractere și numele)

**def** comparator(x,y): *#return <0 daca x<y , 0 daca x=y, >0 daca x>y* **if** x[1]<y[1]:  
 **return** -1  
 **if** x[1]>y[1]:  
 **return** 1  
 **if** x[1]==y[1]:  
 **return** x[0]-y[0] *#v. curs - detaliat cu if-uri***import** functools  
persoane\_nume=sorted(persoane,key=functools.cmp\_to\_key(comparator))  
print(persoane\_nume)  
  
**"""def cheie(x):  
 return (x[1], x[0])  
persoane\_nume = sorted(persoane, key=cheie)  
# persoane\_nume=sorted(persoane,key=lambda x:(x[1],x[0]))  
print(persoane\_nume) """**

c) Să se afișeze vârsta maximă a unei persoane din listă (folosind max(lista,key))

#max-pentru orice secventa: sir, lista, tuplu  
#max(lista de tupluri) => tuplul cel mai mare comparat dupa criteriul default = componenta cu componenta  
print(max(persoane\_nume))  
 *#schimbat criteriul/cheia de comparare - varsta=al treilea element din tuplu*  
defvarsta(x):  
 return x[2]  
print(max(persoane\_nume, key=varsta)[2])

**MATRICE, VECTORI**

5. Se citește un număr natural N.

a) Să se genereze și afișeze o matrice de dimensiune NxN, cu elementele de la 1 la N\*N - în ordine crescătoare, de la stânga la dreapta și de sus în jos.

N=int(input())  
matrice=[list(range(1+N\*i, 1+N\*(i+1)))**for** i **in** range(N)]  
**for** linie **in** matrice: ***# afisarea sub forma unei matrici*****for** x **in** linie:  
 print(**f"{**x**:4} "**,end=**""**)  
 print() ***#afisarea sub forma unei matrici***

b) Pentru a parcurge elementele matricei în spirală, pornind din colțul din stânga-sus (spre dreapta, în jos, spre stânga, în sus, …), să se obțină întâi o listă având elemente de tip tuplu (linie, coloană) care să reprezinte pozițiile care trebuie parcurse în această spirală.   
lista=[]  
m=N-1  
k=0  
**while** k<m:  
 i=k  
 **while** i<m:  
 t=(k,i)  
 lista.append(t)  
 i=i+1  
 i=k  
 **while** i<m:  
 t = (i, m)  
 lista.append(t)  
 i = i + 1  
 i=m  
 **while** i>k:  
 t = (m, i)  
 lista.append(t)  
 i = i-1  
 i=m  
 **while** i>k:  
 t = (i, k)  
 lista.append(t)  
 i = i-1  
 k=k+1  
 m=m-1  
**if** m==k:  
 t=(k,k)  
 lista.append(t)  
print(lista)

c) Folosind lista de tupluri de mai sus, să se afișeze elementele din matrice aflate la acele poziții.



lista\_poz = [(0, 0), (0, 1), ..., (0, N-2), (0, N-1), (1, N-1), ..., (N-2, N-1), (N-1, N-1), (N-1, N-2), ..., (N-1, 1), (N-1, 0), (N-2, 0), ..., (1, 0), (1, 1), (1, 2), ...]

spirala = [1, 2, 3, 4, 5, 10, 15, 20, 25, 24, 23, 22, 21, 16, 11, 6, 7, 8, 9, 14, 19, 18, 17, 12, 13]

spirala=[]

**for** x **in** lista:  
 i=x[0]  
 j=x[1]  
 spirala.append(matrice[i][j])  
print(spirala)

6. **Ciurul lui Eratostene.** Se dă un numar natural n. Să se creeze o listă cu numerele prime mai mici sau egale cu n.

n=int(input())  
**def** prime(x):  
 ok=1  
 **for** d **in** range(2, int(x\*\*0.5)+1):  
 **if** x%d==0:  
 ok=0  
 **if**(ok==1 **and** x>1):  
 **return True  
 else**:  
 **return False**list=[]  
**for** i **in** range(2, n+1):  
 **if** prime(i)== **True**:  
 list.append(i)  
print(list)

7. Se dau două mulțimi cu elementele ordonate crescător (câte una pe linie). Să se determine eficient reuniunea și intersecția celor două mulțimi (fără a folosi set).

s=input()  
a=[int(x) **for** x **in** s.split()]  
p=input()  
b=[int(x) **for** x **in** p.split()]  
intersectie=[]  
reuniune=[]  
i=0  
j=0  
n=len(a)  
m=len(b)  
**while** i<n **and** j<m:  
 **if** a[i]<b[j]:  
 reuniune.append(a[i])  
 i = i + 1  
 **else**:  
 **if** a[i]==b[j]:  
 reuniune.append(a[i])  
 intersectie.append(a[i])  
 i = i + 1  
 j = j + 1  
 **else**:  
 reuniune.append(b[j])  
 j = j + 1  
**while** i<n:  
 reuniune.append(a[i])  
 i = i + 1  
**while** j<m:  
 reuniune.append(b[j])  
 j = j + 1  
print(reuniune)  
print(intersectie)

8. Se citesc: un număr natural n și un șir de n numere naturale.

a) Să se afișeze toate perechile distincte de elemente din șir cu suma 0 (2-SUM)

b) Să se afișeze toate tripletele de elemente din șir cu suma 0

c) Să se afișeze toate tripletele **distincte** de elemente din șir cu suma 0 (3-SUM)

*#a)*

n=int(input())  
s=input()  
l=[int(x) **for** x **in** s.split()]  
l=sorted(l,key=**None**)  
**for** i **in** range(n):  
 **for** j **in** range(i+1,n):  
 **if** l[i]+l[j]==0:  
 print(l[i],l[j])  
*#b)***for** i **in** range(n):  
 **for** j **in** range(n):  
 **for** k **in** range(n):  
 **if** l[i] + l[j] + l[k] == 0:  
 print(l[i], l[j], l[k])  
*#c)***for** i **in** range(n):  
 **for** j **in** range(i+1,n):  
 **for** k **in** range(j+1,n):  
 **if** l[i] + l[j] + l[k] == 0:  
 print(l[i], l[j], l[k])

**ȘIRURI DE CARACTERE**

4. Se citește de la tastatură un text. Se cere să se “traducă” în limba păsărească textul dat astfel: după fiecare vocală se adaugă litera p și încă o dată acea vocală (după a, e, i, o, u se adaugă respectiv pa, pe, pi, po, pu). Exemplu: “Ana are mere.” devine “Apanapa aparepe meperepe.” Fiind dat un astfel de text în limba păsărească, se poate obține textul original? Dacă da, scrieți un program care primind un text în limba păsărească construiește în memorie și afișează textul inițial.

s=input()  
rez=**""**voc=**"aeiouAEIOU"**l=len(s)  
i=0  
**while** i < l:  
 **if** i>0 **and** s[i]==**'p' and** (s[i-1]==s[i+1] **or** ord(s[i-1])==ord(s[i+1])+32 **or** ord(s[i-1])+32==ord(s[i+1])) **and** s[i-1] **in** voc:  
 i=i+2  
 **else**:  
 rez=rez+s[i]  
 i=i+1  
print(rez)

*#ord(a) returneaza codul ascii al elementului a*