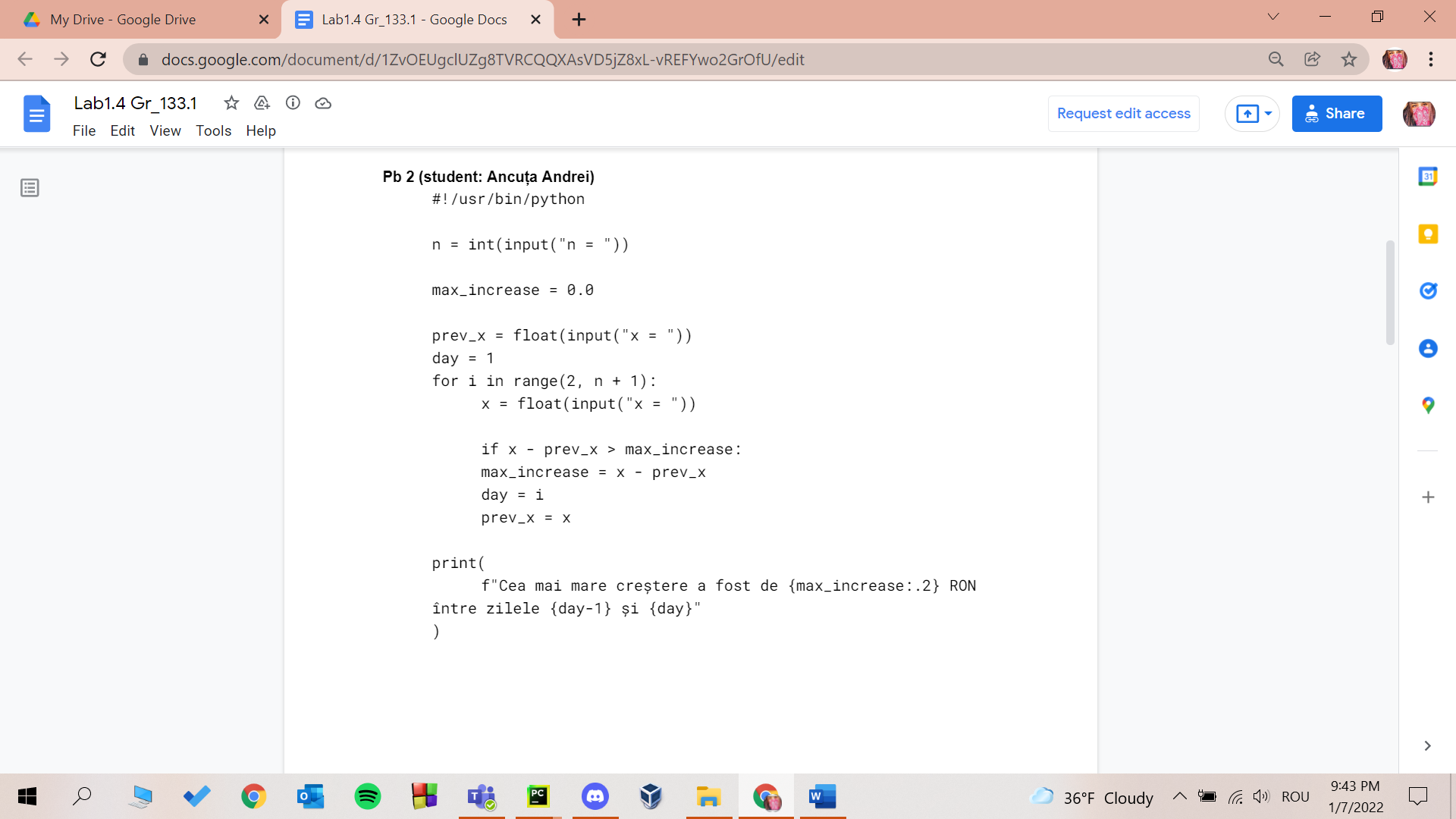
LABORATOR 1

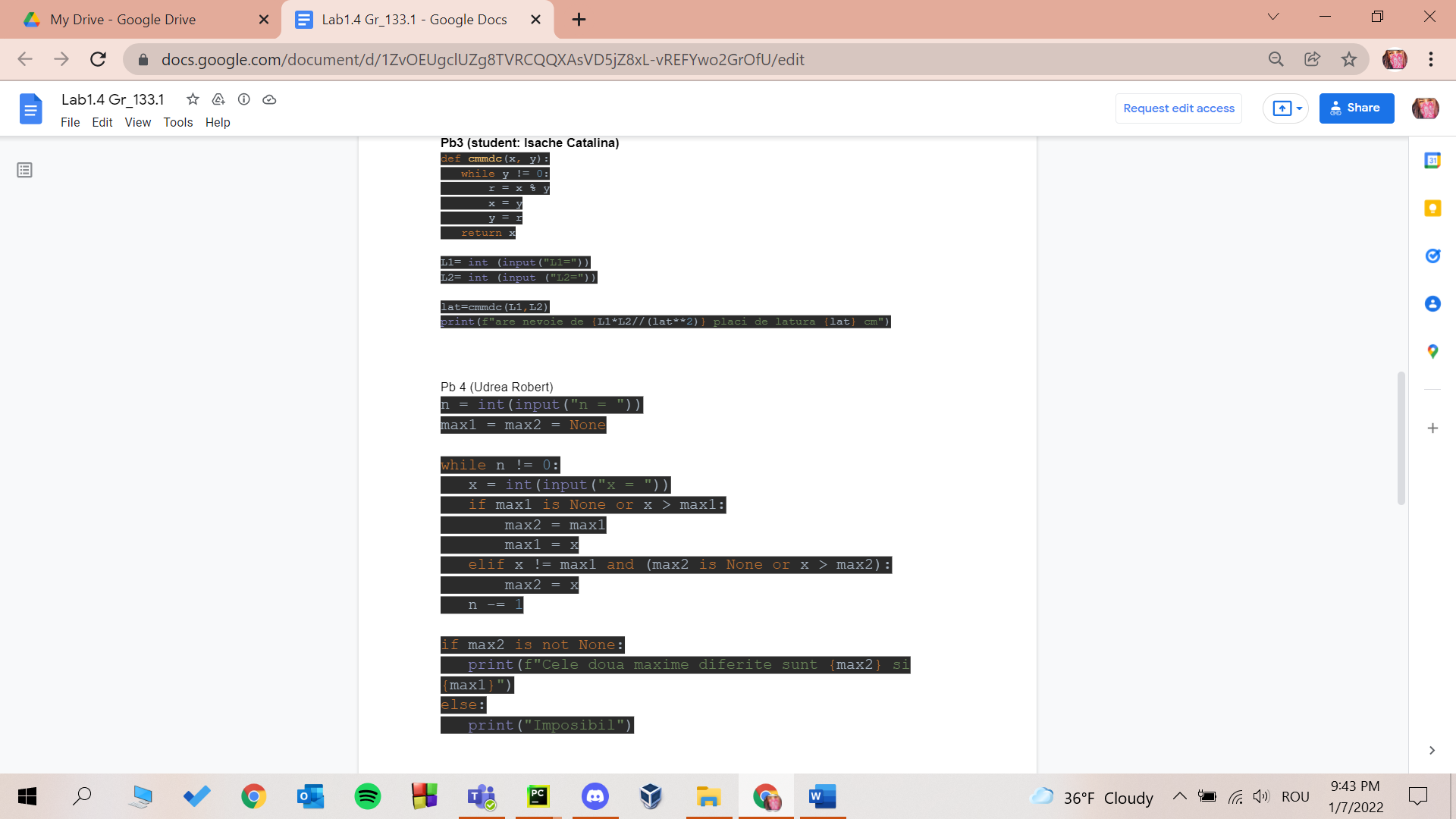
1. Se citește un număr natural n. Să se testeze dacă este palindrom.

n = int(input("n = "))  
salv = n  
inv = 0  
while n != 0:  
 inv = inv\*10 + n%10  
 n = n // 10  
if inv == salv:  
 print(f"Numarul {salv} este palindrom.")  
else:  
 print(f"Numarul {salv} nu este palindrom.")

1. Se citesc un număr natural n și un șir format din 𝑛 numere reale strict pozitive (𝑛≥2), reprezentând cursul de schimb valutar RON/EURO din 𝑛 zile consecutive (fiecare număr va fi dat pe o linie). Să se afișeze zilele între care a avut loc cea mai mare creștere a cursului valutar, precum și cuantumul acesteia (cu ≤ 2 zecimale). De exemplu, pentru 𝑛=6 zile și cursul valutar dat de șirul 4.25,4.05, 4.25, 4.48, 4.30, 4.40, cea mai mare creștere a fost de 0.23 RON, între zilele 3 și 4.



1. Un meșter trebuie să paveze întreaga pardoseală a unei bucătării cu formă dreptunghiulară de dimensiune 𝐿1×𝐿2 centimetri, cu plăci de gresie pătrate, toate cu aceeași dimensiune. Știind că meșterul nu vrea să taie nici o placă de gresie și vrea să folosească un număr minim de plăci, să se determine dimensiunea plăcilor de gresie de care are nevoie, precum și numărul lor. De exemplu, dacă 𝐿1=440 cm și 𝐿2=280 cm, atunci meșterul are nevoie de 77 de plăci de gresie, fiecare având latura de 40 cm.
2. Se citesc un număr natural n și un șir format din 𝑛𝑛 numere întregi (𝑛≥2). Să se afișeze cele mai mari două valori distincte din șir sau mesajul "Imposibil", dacă acestea nu există.



1. Pentru o ecuație de gradul doi să se citească de la tastatură coeficienții a, b, c (numere întregi). Să se rezolve ecuația.

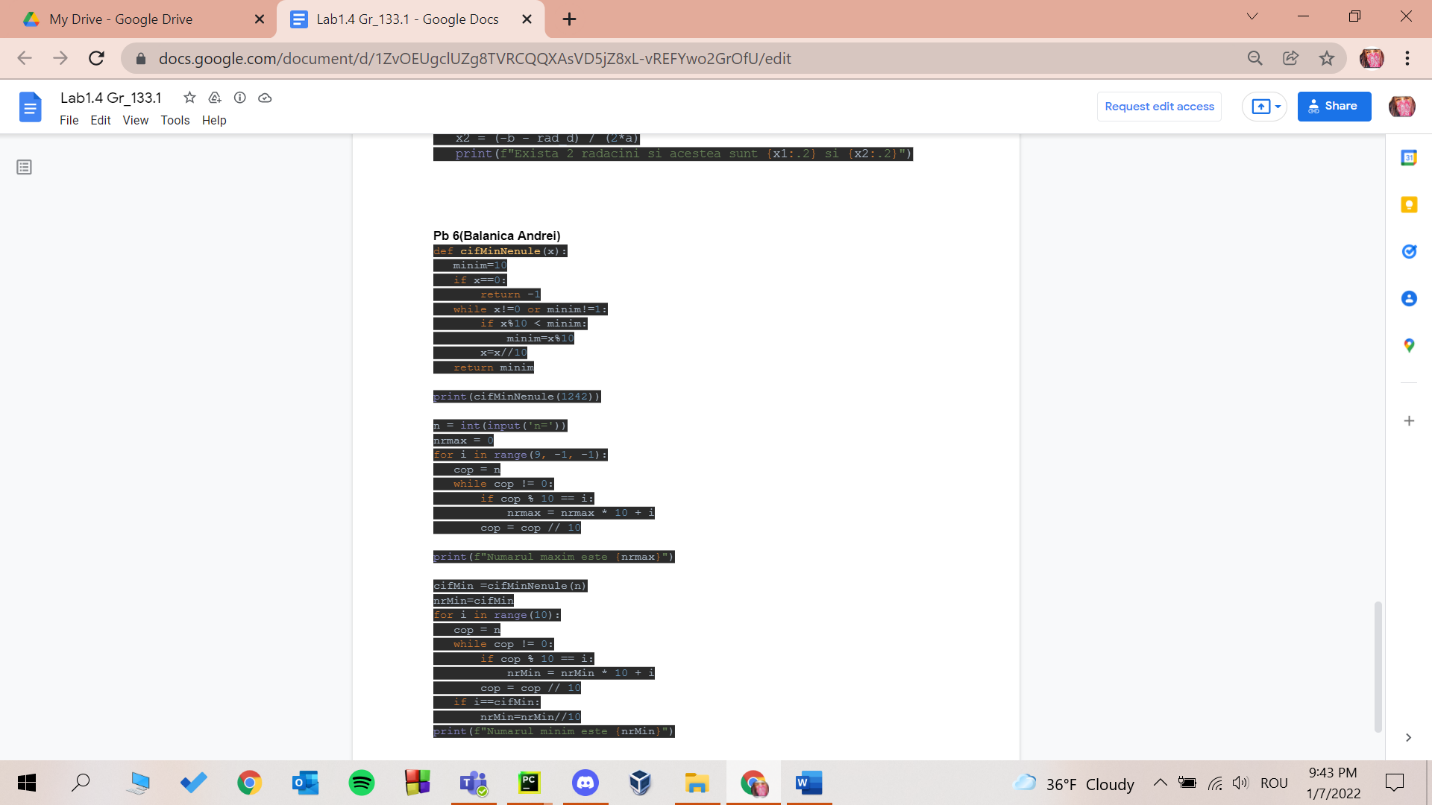
import math  
a = int(input("a = "))  
b = int(input("b = "))  
c = int(input("c = "))  
delta = b \*\* 2 - 4 \* a \* c  
if delta < 0:  
 print("Ecuatia nu are nici o radacina.")  
elif delta == 0:  
 primasolutie = -b/(2\*a)  
 print(f"Ecuatia are o singura radacina x = {primasolutie:.2}.")  
else:  
 radicaldelta = math.sqrt(delta)  
 primasolutie = (-b + radicaldelta) / (2\*a)  
 adouasolutie = (-b - radicaldelta) / (2\*a)  
 print(f"Ecuatia are doua radacini diferite x1 = {primasolutie:.2} si x2 = {adouasolutie:.2}.")

1. Se citește un număr natural n.
   1. Să se afișeze cel mai mare număr care se poate obține cu cifrele lui n.
   2. Să se afișeze cel mai mic număr care se poate obține cu cifrele lui n.

De exemplu, pentru 𝑛=812383 trebuie afișate numerele 883321 și 123388.

1. Un greiere se deplasează efectuând câte o săritură, lungimea inițială a săriturii fiind de 𝑥 cm. După fiecare 𝑛𝑛 sărituri, lungimea săriturii greierului se micșorează cu 𝑝 procente. Cunoscându-se valorile 𝑥, 𝑛, 𝑝, precum și numărul de sărituri 𝑚 pe care le face greierele, să se scrie un program care să afișeze distanța parcursă de greiere.

De exemplu, pentru 𝑥=20, 𝑛=10, 𝑝=10 și 𝑚=20 distanța parcursă de greiere este egală cu 380 cm, deoarece primele 10 sărituri efectuate au, fiecare, lungimea de 20cm, iar următoarele 10 au, fiecare, lungimea de 18 cm.



LABORATOR 2

1. Se citește un cuvânt. Să se șteargă din cuvânt toate aparițiile primei litere. Se va afișa un mesaj de forma: După ștergerea literei ‚X’ șirul obținut este „S” de lungime L folosind diferite tipuri de formatare (cu parametri poziționali și f-string-uri).

sir = input(„sir=”)  
litera=sir[0]  
sir = sir.replace(sir[0], „”)  
print(f”Dupa stergerea literei ‚{litera}’ sirul obținut este \”{sir}\” de lungime {len(sir)}”)  
print(„Dupa stergerea literei ‚{}’ sirul obținut este \”{}\” de lungime {}”.format(litera,sir,len(sir)))  
print(„Dupa stergerea literei ‚{0}’ sirul obținut este \”{1}\” de lungime {2}”.format(litera,sir,len(sir)))  
print(„Dupa stergerea literei ‚{lit}’ sirul obținut este \”{s}\” de lungime {nr}”.format(lit=litera,s=sir,nr=len(sir)))  
  
lista=[10,20,30]  
print(„primul {} al doilea {} al treilea {}”.format(\*lista)) # 3 parametri  
#print(„primul {} al doilea {} al treilea {}”.format(lista)) Eroare (un parametru)

1. Să se verifice (folosind întâi metoda find, apoi index) dacă un șir de caractere 𝑡𝑡 apare ca subșir într-un șir 𝑠𝑠, iar în caz afirmativ să se afișeze toate pozițiile la care începe 𝑡𝑡 în 𝑠𝑠.

De exemplu, șirul 𝑡𝑡 = „𝑎𝑎𝑏𝑏𝑐𝑐” apare ca subșir în șirul 𝑠𝑠 = „𝑎𝑎𝑏𝑏𝑐𝑐𝑐𝑐𝑎𝑎𝑏𝑏𝑐𝑐𝑎𝑎𝑏𝑏𝑎𝑎𝑏𝑏𝑐𝑐𝑐𝑐” începând cu pozițiile 0, 4 și 9.

Sir\_citit = input(„sir=”)  
sir\_cautat = input(„sir cautat= „)  
lungime=len(sir\_cautat)  
try:  
 pos = sir\_citit.index(sir\_cautat)  
 while True:  
 print(pos, end=” „)  
 pos = sir\_citit.index(sir\_cautat,pos+lungime)  
except:  
 print(„gata”)

1. Se citește un cuvânt s de cel mult 10 de caractere. Sa se afișeze (folosind s[i:j]) pe câte o linie cuvintele obținute succesiv din s tăind prima și ultima literă (afișate centrat pe 10 de caractere):

Exemplu:

algoritm

lgorit

gori

or

#Varianta mea

cuvant = input („cuvant = „)  
print (cuvant)  
lungime = len(cuvant)  
k = 0  
for i in range (lungime\\2):  
 print („ „\*k, cuvant[i+1:-i-1], „ „\*k)  
 k +=1

#Varianta rezolvare   
sir = input(„sir = „)  
lungime = len(sir)  
i = 0  
while i < lungime-i:  
 print(sir[i:lungime-i].center(lungime, „\*”))  
 i += 1

1. Într-o propoziție a fost efectuată, posibil de mai multe ori, aceeași greșeală de ortografie.

a) Scrieți un program care citește propoziția, șirul greșit și șirul corect, după care afișează propoziția corectă. De exemplu, în propoziția „Problemele cu șiruri de caracteger nu sunt ggerle!”greșeală constă în faptul că în loc de șirul “re” a fost scris șirul “ger”.

b) Modificați programul astfel încât să corecteze maxim 2 astfel de greșeli, iar dacă sunt mai multe să afișeze mesajul: “textul contine prea multe greseli, doar 2 au fost corectate”

prop = input(„Propozitia= „)  
gresit = input(„Sirul gresit= „)  
corect = input(„Sirul corect= „)  
#a)  
prop2 = prop.replace(gresit,corect)  
print(prop2)  
#b)  
prop2 = prop.replace(gresit,corect,2)  
print(prop2)  
nr=prop.count(gresit)  
if nr>2:  
 print(„textul contine prea multe greseli, doar 2 au fost corectate”)

1. Scrieți un program care să înlocuiască într-o propoziție toate aparițiile unui cuvânt 𝑠𝑠 cu un cuvânt 𝑡𝑡 (!cuvânt, nu subșir). Cuvintele sunt separate prin unul sau mai multe spații.

Sir = input(„sir este: „)  
s = input(„s este:”)  
t = input(„t este: „)  
lista = sir.split(„ „)  
for i in range(len(lista)):  
 if lista[i] == s:  
 lista[i] = t  
print(lista)  
sir\_final = „ „.join(lista)  
print(sir\_final)

1. Se citește un text codificat după regula: În fata fiecărui caracter este scris un număr de cel mult 2 cifre care reprezintă numărul de apariții consecutive ale acestui. Scrieți un program care decodifica textul. De exemplu textul 1G10o4l se va decodifica Goooooooooollll

b) Scrieți un program care, dat un text, îl codifică după regula de la a)

# a  
cuv = input(“Cuvantul codificat este “)  
n = len(cuv)  
l = []  
nrcif = 0  
for I in range(n):  
 if cuv[i].isdigit():  
 nrcif = nrcif + 1  
 elif cuv[i].isalpha():  
 nr = int(cuv[I – nrcif: i])  
 l += [nr \* cuv[i]]  
 nrcif = 0  
rez = “”.join(l)  
print(rez)  
# b  
s = “Goooooooooollll”  
fin = “”  
I = 0  
litera = s[0]  
nrlit = 0  
while i<len(s):  
 if s[i]==litera:  
 nrlit=nrlit+1  
 else:  
 fin+=str(nrlit)+litera  
 litera=s[i+1]  
 nrlit=1  
 i+=1  
fin+=str(nrlit)+litera  
print(fin)

1. Jurnalul electronic al Anei conține, în fiecare zi, câte o frază cu informații despre cheltuielile pe care ea le-a efectuat în ziua respectivă. Scrieți un program care să citească o frază de acest tip din jurnalul Anei și apoi să afișeze suma totală cheltuită de ea în ziua respectivă. De exemplu, pentru fraza “Astăzi am cumpărat pâine de 5 RON, pe lapte am dat 10 RON, iar de 15 RON am cumpărat niște cașcaval. De asemenea, mi-am cumpărat și niște papuci cu 50 RON!”, programul trebuie să afișeze suma totală de 80 RON. Fraza se consideră corectă, adică toate numerele care apar în ea sunt numere naturale reprezentând sume cheltuite de Ana în ziua respectivă!

Fraza = input(“Jurnalul electronic al Anei: “)  
frazanoua = fraza.split()  
s = 0  
for cuvant in frazanoua:  
 if cuvant.isdigit():  
 s += int(cuvant)  
print(f”Suma totala cheltuita este de {s} RON.”)

1. Numele Pre-Nume

Scrieți un program care citește un șir de caractere și decide dacă acesta este un nume complet corect al unei persoane. Se consideră că un nume complet este corect dacă respectă următoarele proprietăți:

• persoana poate avea cel mult doua prenume, iar dacă sunt două atunci sunt despărțite printr-o cratimă (‘-’). La fel și în cazul numelui de familie

• numele de familie sau prenume conține doar litere și cel mult o cratimă.

• orice nume de familie sau prenume este format din cel puțin 3 litere.

• orice nume sau prenume începe cu literă mare.

Exemplu de nume complete corecte: Ionescu-Cherea Mihai-Adrian, Popescu Elena-Maria, Vlad Matei și de nume incorecte: Ionescu – Cherea Mihai, Vlad Matei Alexandru

1. Cifrul lui Cezar

a) Se citește un text și un număr natural k. Să se afișeze textul cifrat cu cifrul lui Cezar, prin care fiecare literă (!doar literele) este înlocuită cu litera aflată peste 𝑘𝑘 poziții la dreapta în alfabet în mod circular (valoarea 𝑘𝑘 reprezintă cheia secretă comună pe care trebuie să o cunoască atât expeditorul, cât și destinatarul mesajului criptat).

b) Se citește un număr natural k și text criptat cu cifrul lui Cezar cu cheia k. Să se afișeze textul decriptat.

1. Se citesc doua cuvinte formate doar din litere mici. Să se verifice dacă sunt anagrame.

Varianta mea:

icuv = input(“Primul cuvant este: “)  
iicuv = input(“Al doilea cuvant este: “)  
ok = 1  
for litera in icuv:  
 if (iicuv.find(litera) == -1):  
 ok = 0  
 if icuv.count(litera) != iicuv.count(litera):  
 ok = 0  
if ok == 1:  
 print(“Cele doua cuvinte sunt anagrame.”)  
else: print(“Cele doua cuvinte nu sunt anagrame.”)

Variante Seminar 3:

# VARIANTA 1:

def anagrame (s, t):  
 if len(s) != len(t):  
 return False  
 for litera in set(s):  
 if s.count(litera) != t.count(litera):  
 return False  
 return True

sir1 = input(“Primul cuvant este: “)  
sir2 = input(“Al doilea cuvant este: “)  
print(anagrame(sir1, sir2))

# VARIANTA 2:

def anagrame (sir1, sir2):  
 if len(sir1) == len(sir2):  
 for litera in sir1:  
 if litera not in sir2:  
 return False  
 sir2 = sir2.replace(litera, “”, 1)  
 return True  
 return False  
  
  
sir1 = input(“Primul cuvant este: “)  
sir2 = input(“Al doilea cuvant este: “)  
print(anagrame(sir1, sir2))

# VARIANTA 3:

#!/usr/bin/python  
def count\_characters(word):  
 char\_frequency\_dict = {}  
 for c in word:  
 try:  
 char\_frequency\_dict[c] += 1  
 except KeyError:  
 char\_frequency\_dict[c] = 1  
 return char\_frequency\_dict  
def are\_anagrams(word1, word2):  
 if len(word1) != len(word2):  
 return False  
 chars1 = count\_characters(word1)  
 chars2 = count\_characters(word2)  
 return chars1 == chars2  
[word1, word2] = input("Two words: ").split()[:2]  
if are\_anagrams(word1, word2):  
 print(f"{word1} și {word2} sunt anagrame")  
else:  
 print(f"{word1} și {word2} nu sunt anagrame")

1. a) 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.

b) Se citește de la tastatură un text în care cuvintele sunt despărțite în silabe cu ajutorul cratimelor. Se cere să se “traducă” textul dat în limba păsărească astfel: după fiecare silabă se adaugă litera p și se repetă ultima literă din acea silabă. Afișați traducerea și cu cratime, dar și fără.

Exemplu: “a-na a-re mul-te me-re ro-sii si de-li-cioa-se.” Devine

“apa-napa apa-repe mulpl-tepe mepe-repe ropo-siipi sipi depe-lipi-cioapa-sepe.” Și

“apanapa aparepe mulpltepe meperepe roposiipi sipi depelipicioapasepe.”

Fiind dat un astfel de text în limba păsărească (cel care conține și cratime), se poate obține textul original? Dacă da, scrieți un program care decodifică un astfel de text.

SEMINAR 2

Restul problemelor sunt rezolvate in fișierul cu rezolvări.

1. Într-o propoziție, Ana a sintetizat informațiile despre cumpărăturile pe care le-a efectuat într-o anumită zi, pentru fiecare produs cumpărat ea, precizând cantitatea și prețul unitar(e.g., "Astăzi am cumpărat 5 kg de mere cu 2.30 RON kilogramul și 2 pâini a câte 5 lei bucata."). Afișați totalul cheltuielilor Anei din ziua respectivă.

sir = input("Sirul dat este: ")  
cantitate = pret = None  
suma = 0  
for cuvant in sir.split():  
 try:  
 x = float(cuvant)  
 if cantitate is None:  
 cantitate = x  
 else:  
 pret = x  
 suma += pret \* cantitate  
 cantitate = pret = None  
 except ValueError:  
 pass  
print(f"Suma cheltuita este: {suma}.")

SEMINAR 3

1. Se citesc doua cuvinte formate doar din litere mici. Să se verifice dacă sunt anagrame.

Rezolvare problema – PROBLEMA 10 (LABORATOR 2)

1. În fișierul text “numar\_lipsa.txt” se găsesc pe prima linie 𝑛−1numere naturale distincte dintre primele 𝑛numere naturale nenule. Să se afișeze numărul lipsă.

Exemplu: dacă fișierul “numar\_lipsa.txt” conține “2 1 5 4”, se va afișa numărul lipsă 3.

# VARIANTA 1:

f = open("nr\_lipsa.txt")  
s = f.read()  
f.close()  
lista = [int(x) for x in s.split()]  
n = len(lista)+1  
d = {x: False for x in range(1, n+1)}  
for x in lista:  
 d[x] = True  
for x in d:  
 if d[x] == False:  
 print(x)  
 break  
print(lista)

1. Fișierul text “numere\_comune.txt” conține numere naturale despărțite prin spații și scrise pe mai multe linii. Să se afișeze în fișierul “comune.txt” numerele care apar pe toate liniile din fișier. Exemplu: dacă fișierul “numere\_comune.txt” conține

2 1 5 1 3

1 4 2 2

2 1 1 6 8

atunci fișierul “comune.txt” va conține (numerele nu neapărat în această ordine):

1 2

f = open('numere\_comune.txt')  
g = open('comune.txt', 'w')  
  
a = set(int(x) for x in f.readline().split())  
for linie in f:  
 # b = set(int(x) for x in linie.split())  
 # a = a & b  
 b = [int(x) for x in linie.split()]  
 a = a.intersection(b)  
  
if len(a) == 0:  
 g.write('nu e niciun nr')  
else:  
 for i in a:  
 g.write(str(i) + " ")  
  
f.close()  
g.close()

LABORATOR 3

1. List comprehensions:

a) Sa se genereze lista literelor in lowercase de la a la z

lista = [chr(x) for x in range(ord('a'), ord('z')+1)]  
print(lista)

b) Pentru un numar natural n citit de la tastatura, sa se genereze lista de forma 1, -2, 3, -4, ... pana la n (cu semnul corespunzator).

n = int(input("n = "))  
lista = [(x if x % 2 == 1 else -x) for x in range(1, n+1)]  
print(lista)

n = int(input("n="))  
lista = [x\*(-1)\*\*(x+1) for x in range(1, n+1)]  
print(lista)

c) Se da o lista de numere. Sa se obtina lista cu elementele impare din lista data.

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
impare = [x for x in lista if x % 2 == 0]  
print(impare)

d) Se da o lista de numere. Sa se obtina lista cu elementele aflate pe pozitii impare in lista data.

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
pozitiiimpare = [lista[i] for i in range(len(lista)) if i % 2 != 0]  
print(pozitiiimpare)

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
pozitiiimpare = lista[1::2]  
print(pozitiiimpare)

e) Sa se obtina, pentru o lista de numere data, lista continand elementele care au aceeasi paritate cu pozitia pe care se afla. De exemplu, pentru lista [2, 4, 1, 7, 5, 1, 8, 10], lista calculata va contine elementele: 2, 7, 1, 8.

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
accpar = [lista[x] for x in range(len(lista)) if lista[x] % 2 == x % 2]  
print(accpar)

f) Sa da o lista. Sa se obtina lista cu perechiile (tupluri) de elementele de pe pozitii vecine. De exemplu pentru lista [1,2,3,4] lista rezultata ar fi [(1,2),(2,3),(3,4)]

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
ppozvec = [(lista[x], lista[x+1]) for x in range(len(lista)-1)]  
print(ppozvec)

Sa da o lista. Sa se obtina lista cu perechiile disjuncte (tupluri disjuncte) de elementele de pe pozitii vecine. De exemplu pentru lista [1,2,3,4] lista rezultata ar fi [(1, 2), (3, 4).]

lista = [int(x) for x in input("Elemente:").split()]  
print(lista)  
ppozvecn = [(lista[x], lista[x+1]) for x in range(0, len(lista)-1, 2)]  
print(ppozvecn)

g) Sa se scrie o functie care primeste ca parametru un numar n si genereaza cu ajutorul list comprehension o lista de n liste. In fiecare lista element vom avea siruri de forma "x\*y=rez". Elementul x va avea drept valoare indicele listei-element iar y va varia intre 1 si n (sirurile fiind ordonate crescator). Practic se genereaza tabla inmultirii numerelor de la 1 la n. Sa se apeleze functia si sa se afiseze rezultatul.

def tabla\_inmultirii(n):  
 matrice = [[f"{x}\*{y} = {x\*y}" for y in range(1, n+1)] for x in range(1, n+1)]  
 return matrice  
  
def afisare (matrice):  
 listamax = max([len(str(elem)) for linie in matrice for elem in linie])  
 for linie in matrice:  
 for elem in linie:  
 print(f"{elem}".rjust(listamax+1), end="\t")  
 print()  
  
n = int(input("n = "))  
m = tabla\_inmultirii(n)  
afisare(m)

h) Sa se obtina lista cu toate permutarile circulare ale unui sir dat. De exemplu, pentru sir="abcde" vom obtine lista ['abcde', 'bcdea', 'cdeab', 'deabc', 'eabcd'].

sir = input("sir = ")  
lista = [sir[i:]+sir[:i] for i in range(len(sir))]  
print(lista)

i) Sa se scrie o functie care returneaza o lista de n liste (unde n e dat ca parametru in functie), cu proprietatea ca prima lista va fi vida, a doua lista va avea un singur element egal cu 1, a treia lista va avea doua elemente egale cu 2, si asa mai departe pana la a n-a lista care va avea n-1 elemente egale cu n-1. De exemplu pentru n=4, lista de liste va fi: [[],[1],[2,2],[3,3,3]].

n = int(input("n ="))  
lista = [[x for i in range(x)] for x in range(n)]  
print(lista)

1. Sortari liste:

a) Se considera o lista de numere. Sa se sorteze folosind o functie lambda asa cum s-ar fi sortat daca numerele erau siruri.

Varianta prin definirea functiei:

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
  
def criteriu(x):  
 return str(x)  
  
listasortata = sorted(lista, key=criteriu)  
  
print(lista)  
print(listasortata)

Varianta ceruta:

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
listasortata = sorted(lista, key = lambda x : str(x))

print(lista)  
print(listasortata)

b) Se considera o lista de numere. Sa se sorteze folosind o functie lambda comparand intai ultima cifra apoi penultima, etc. (asa cum am fi sortat daca erai siruri cu literele in ordine inversa.

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
listasortata = sorted(lista, key = lambda x : str(x)[::-1])

print(lista)  
print(listasortata)

c) Se considera o lista de numere. Sa se sorteze descrescator dupa lungimea numarului.

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
listasortata = sorted(lista, key = lambda x : -len(str(x)))  
# sau listasortata = sorted(lista, key = lambda x : len(str(x)), reverse=True)  
print(lista)  
print(listasortata)

d) Se considera o lista de numere. Sa se sorteze dupa numarul de cifre distincte.

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
listasortata = sorted(lista, key=lambda x: len(set(str(x))))  
print(lista)  
print(listasortata)

e) sortare numere intai dupa lungime si apoi dupa valoare.

lista = [1, 3, 2, 5, 7, 8, 124, 2345, 65, 34, 734, 2322, 654]  
listasortata = sorted(lista, key=lambda x: (len(str(x)), x))  
print(lista)

print(listasortata)

SEMINAR 4

1. a) Scrieți o funcție care returnează o matrice triunghiulară de dimensiune 𝑛, având forma următoare:

• prima coloană conține numerele 1,2,3,...,𝑛;

• ultima linie conține numerele 𝑛,𝑛−1,...,2,1;

• restul elementelor aflate în triunghiul de sub diagonala principală se calculează ca sumă a elementelor vecine de la vest, sud și sud-vest.

Exemplu: pentru 𝑛=4 matricea cerută este 𝑀=[[1],[2,15],[3,10,15],[4,3,2,1]]

def matrice(n):  
 M = [[i+1] + [0]\*i for i in range(n-1)] + [[i for i in range (n, 0, -1)]]  
 for i in range(n-2, 0, -1):  
 for j in range(1, i+1):  
 M[i][j] = M[i][j-1] + M[i+1][j-1] + M[i+1][j]  
 return M  
  
print (matrice(4))

b) Scrieți o funcție care afișează o matrice triunghiulară, memorată sub forma unei liste de liste, în formă matriceală, cu elementele de pe fiecare coloană aliniate la dreapta.

def matrice(n):  
 M = [[i+1] + [0]\*i for i in range(n-1)] + [[i for i in range (n, 0, -1)]]  
 for i in range(n-2, 0, -1):  
 for j in range(1, i+1):  
 M[i][j] = M[i][j-1] + M[i+1][j-1] + M[i+1][j]  
 return M  
  
def printMat(M):  
 # vmax = max([max(linie) for linie in M])  
 n = len(M)  
 vmax = M[n // 2][n // 2]  
 cifmax = len(str(vmax))  
 for linie in M:  
 for x in linie:  
 # print(f"{x:{cifmax}}",end=" ")  
 print(str(x).rjust(cifmax), end=" ")  
 print()  
  
printMat(matrice(11))

1. Scrieți o funcție care primește un număr variabil de liste și o valoare 𝑥 și furnizează toate listele care conțin valoarea 𝑥.

def cautax(\*liste, nr\_cautat):  
 liste\_valide = []  
 for lista in liste:  
 if nr\_cautat in lista:  
 liste\_valide.append(lista)  
 return liste\_valide  
  
print (cautax([1, 5, 6], [4, 6, 8, 9], [3, 6, 7, 8], nr\_cautat = 8))

1. a) Pentru un student se cunosc următoarele informații: numele, grupa și o listă cu creditele obținute la toate examenele din anul respectiv. Considerând o listă de studenți, scrieți o funcție care să adauge la fiecare student situația sa școlară: promovat (True) sau nepromovat (False). Pentru a fi considerat promovat, un student trebuie să nu aibă nici un examen nepromovat (adică o valoare egală cu 0 în lista creditelor), iar suma creditelor obținute să fie mai mare sau egală decât un număr minim de credite dat.

CITIREA NOASTRA:

def citire(nume\_fisier):  
 file = open(nume\_fisier)  
 studenti = []  
 for line in file:  
 rand = line.split(',')  
 stud = (rand[0], int (rand[1]), tuple(int(x) for x in rand[2:]))  
 studenti.append(stud)  
 file.close()  
 return studenti  
  
  
studenti = citire("studenti.csv")  
print(\*studenti, sep="\n")

def parse\_students(csv\_file):  
 students = []  
 for line in csv\_file:  
 items = line.split(",")  
 student\_entry = (  
 items[0],  
 int(items[1]),  
 tuple(int(credits) for credits in items[2:]),  
 )  
 students.append(student\_entry)  
 return students

def evaluate(students):  
 for i in range(len(students)):  
 s = students[i]  
 passed\_all = True  
 total\_credits = 0  
 for credits in s[2]:  
 if credits == 0:  
 passed\_all = False  
 break  
 total\_credits += credits  
 promoted = passed\_all and total\_credits >= min\_credits  
 students[i] = s + (promoted,)

min\_credits = int(input("Minimum credits = "))  
with open("studenti.csv") as f:  
 students = parse\_students(f)  
 evaluate(students)  
 print(\*students, sep='\n')

b) Scrieți câte o funcție care să furnizeze cheia necesară sortării listei cu informații despre studenți în raport de fiecare dintre următoarele criterii:

• crescător după grupă și în fiecare grupă în ordine alfabetică;

• întâi studenții promovați, apoi cei nepromovați și în fiecare categorie în ordine alfabetică;

• descrescător după suma creditelor, iar în cazul unor sume egale în ordinea crescătoare a grupei și în ordine alfabetică în cadrul grupei;

• în ordinea crescătoare a grupelor, în cadrul fiecărei grupe mai întâi studenții promovați, iar apoi cei nepromovați, în fiecare categorie (promovat/nepromovat) în ordinea descrescătoarea sumei creditelor și, în cazul unor sume egale, în ordine alfabetică.

LABORATOR 4

1. a) Scrieți o funcție care primește ca parametru un număr variabil de nume de fișiere și returnează un dicționar cu cuvintele care apar în cel puțin unul dintre fișiere și frecvența totală cu care apare fiecare cuvânt (suma frecvențelor cu care apar în fișiere). Cuvintele pot fi pe mai multe linii și pe o linie sunt separate prin spații.

b) Se consideră fișierele cuvinte1.in si cuvinte2.in. Să se afișeze cuvintele care apar în cel puțin unul dintre fișiere ordonate crescător lexicografic.

c) Se consideră fișierul cuvinte1.in. Să se creeze o listă de perechi (cuvinte, frecvențe) cu cuvintele care apar în fișier și frecvența cu care apar, ordonată descrescător după frecvență (folosind funcția de la a)).

d) Să se determine un cuvânt care apare cel mai des în cuvinte2.in, folosind funcția de la a) și funcția max. Dacă sunt mai multe posibilități, se va afișa cuvântul cel mai mic din punct de vedere lexicografic.

# a)  
def frecvente(\*parametrii):  
 dic = {}  
 for nume in parametrii:  
 with open(nume) as f:  
 for linie in f:  
 for cuv in linie.split():  
 dic[cuv] = dic.get(cuv, 0) + 1  
 # if cuv in dic:  
 # dic[cuv] += 1  
 # else:  
 # dic[cuv] = 1  
  
 return dic  
  
d = frecvente("cuvinte1.in", "cuvinte2.in")  
# print(d)  
  
# b)  
print(sorted(d.keys()))  
  
# c)  
d1 = frecvente("cuvinte1.in")  
# print(d1)  
# print(d1.items())  
# print(sorted(d1.items(), key = lambda t: t[1], reverse=True))  
print(sorted(d1.items(), key=lambda t: -t[1]))  
  
# d)  
d2 = frecvente("cuvinte2.in")  
# print(d2)  
  
print(min(d2.items(), key=lambda t: (-t[1], t[0]))[0])  
  
# e)  
sus = jos1 = jos2 = 0  
  
for cuv in d1.keys() | d2.keys():  
 sus += d1.get(cuv, 0) \* d2.get(cuv, 0)  
 jos1 += d1.get(cuv, 0)\*\*2  
 jos2 += d2.get(cuv, 0)\*\*2  
  
res = sus / (jos1\*\*0.5 \* jos2\*\*0.5)  
  
print(f"{res: .2}")

1. În fișierul text “date.in” sunt memorate, pe linii, numele și prenumele studenților dintr-o grupă. Să se scrie un program care să genereze conturile de email ale studenților și parolele temporare, după care să le salveze în fișierul text “date.out”. Contul de email al unui student va fi de forma prenume.nume@s.unibuc.ro, iar parola temporară va fi de forma o literă mare, 3 litere mici și 4 cifre. Se va scrie o funcție care generează parola folosind funcții din modulul random <https://docs.python.org/3/library/random.html> (randint, choice, choices pentru constantele string.ascii\_uppercase, string.digits etc din modulul string <https://docs.python.org/3/library/string.html>).

|  |  |
| --- | --- |
| date.in | date.out (exemplu,parolele sunt generate aleator) |
| Bobocea Andrei  Marinescu Ciprian  Vasile Dragos | [andrei.bobocea@s.unibuc.ro,Wadf2133](mailto:andrei.bobocea@s.unibuc.ro,Wadf2133)  [ciprian.marinescu@s.unibuc.ro,Qsdd2111](mailto:ciprian.marinescu@s.unibuc.ro,Qsdd2111)  [dragos.vasile@s.unibuc.ro,Bbyt7690](mailto:dragos.vasile@s.unibuc.ro,Bbyt7690) |

import random  
import string  
#nr = random.randint(a,b)  
def parola():  
 a = random.choice(string.ascii\_uppercase)  
 b = "".join(random.choices(string.ascii\_lowercase, k=3))  
 c = "".join(random.choices(string.digits, k = 4))  
 return a+b+c  
f= open("date.in")  
g= open("date.out", "w")  
for linie in f:  
 numele, prenumele = linie.split()  
 p = parola()  
 g.write(f"{prenumele.lower()}.{numele.lower()}@s.unibuc.ro,{p}\n")  
f.close()  
g.close()

1. În fișierul "elevi.in" sunt memorate informații despre elevii unei clase; astfel, pe o linie a fișierului se dau următoarele informații despre un elev: CNP, nume (fără spații), prenume (fără spații), lista de note.

De exemplu:

2501910000034 Ionescu Ion 10 8 7 8

2402900000041 Marinica Maria 9 10 8 8 8

1412900000041 Petrescu Petrica 8 10 4 7

a) Memorați lista de elevi din fișier astfel încât să se poată răspundă cât mai eficient la întrebări de tipul celor de la subpunctele următoare (dat CNP elev, care sunt numele, notele, să se lista de note a elevului).

b) Scrieți o funcție care primește ca parametri un CNP și structura de date în care s-au memorat datele despre elevi la punctul a) și crește cu 1 prima notă a elevului cu CNP-ul primit ca parametru. Funcția returnează nota după modificare sau None dacă CNP-ul nu există. Apelați funcția pentru un CNP citit de la tastatură.

c) Scrieți o funcție care primește ca parametri un CNP, o listă de note și structura de date în care s-au memorat datele despre elevi la punctul a) și adaugă lista de note la notele elevului cu CNP-ul primit ca parametru. Funcția returnează lista de note după modificare sau None dacă CNP-ul nu există. Apelați funcția pentru un CNP citit de la tastatură si lista l\_note=[10,8].

d) Scrieți o funcție care primește ca parametri un CNP și structura de date în care s-au memorat datele despre elevi la punctul a)și șterge informațiile despre elevul cu acest CNP. Apelați funcția pentru un CNP citit de la tastatură (dacă CNP-ul nu este în listă funcția nu va modifica nimic și nu va da eroare)

e) Folosind structura de date în care s-au memorat datele despre elevi la punctul a) (nu citind din nou datele) construiți în memorie o lista de liste cu elevii din fișier, un element din lista fiind de forma [nume, prenume, lista de note], ordonată descrescător după medie și, în caz de egalitate, după nume și afișați elementele listei în fișierul „elevi.out”.

f) Scrieți o funcție care primește ca parametru structura de date în care s-au memorat datele despre elevi la punctul a a) și adaugă la informațiile asociate unui student un cod de lungime 6 generat aleator care conține 3 litere urmate de 3 cifre.

Exemplu de apel:

genereaza\_coduri(d)

print(d)

def medie(lista):  
 return sum(lista)/len(lista)

def creste(cnp,dict):  
 if cnp not in dict:  
 return None  
 if dict[cnp][-1][0]<10:  
 dict[cnp][-1][0]+=1  
 return dict[cnp][-1][0]  
  
def adauga(cnp,note,dict):  
 if cnp not in dict:  
 return None  
 #dict[cnp][-1]+=note  
 dict[cnp][-1].extend(note)  
 return dict[cnp][-1]

def sterge(cnp,d):  
 try:  
 #del dict[cnp]  
 print(d.pop(cnp,"cheie inexistenta"))  
 except:  
 print("Eroare del")

f=open("elevi.in")  
d={}

for linie in f:  
 cnp,nume,prenume,note=linie.split(maxsplit=3)  
 d[int(cnp)]=[nume,prenume,[int(x) for x in note.split()]]  
f.close()

print(d)  
cnp\_i=int(input("cnp="))  
l\_note=[10,8]  
print(creste(cnp\_i,d))  
print(d)  
  
print(adauga(cnp\_i,l\_note,d))  
print(d)  
  
print(sterge(cnp\_i,d))  
print(d)  
  
lista\_noua=[x for x in sorted(d.values(),key=lambda t: (-medie(t[-1]),t[0]))]  
print(lista\_noua)

1. Scrieți o funcție cu număr variabil de parametri care să caute un cuvânt dat în mai multe fișiere text. Funcția va scrie într-un fișier text câte o linie pentru fiecare fișier text de intrare, astfel: numele fișierului text de intrare și apoi numerele de ordine ale liniilor pe care apare cuvântul dat în acel fișier (numerotate de la 1)sau un mesaj corespunzător dacă fișierul nu conține cuvântul respectiv. Antetul funcției va fi: cautare\_cuvant(cuv, nume\_fis\_out, \*nume\_fis\_in). Se vor număra aparițiile cuvântului fără a face diferența între literă mare și literă mică.

De exemplu, prin apelul cautare\_cuvant("floare","rez.txt", "eminescu.txt", "paunescu.txt") se va căuta cuvântul “floare” în fișierele text “eminescu.txt” și “paunescu.txt”, iar rezultatul căutării va fi scris în fișierul text “rez.txt”.

def cautare\_cuvant(cuv, nume\_fis\_out, \*nume\_fis\_in):  
 cuv = cuv.lower()  
 c = 1  
 for fisier in nume\_fis\_in:  
 fis = open(fisier, "r")  
 f = fis.readlines()  
 rasp = [fisier]  
 for i in range(len(f)):  
 f[i] = f[i].lower()  
 if f[i].find(cuv) != -1:  
 rasp.append(i+1)  
 print(rasp)  
 if c == 1:  
 g = open(nume\_fis\_out, "w")  
 if len(rasp) != 1:  
 for j in range(len(rasp)):  
 g.write(str(rasp[j]))  
 g.write((" "))  
 g.write("\n")  
 g.close()  
 c = 2  
 else:  
 if len(rasp) != 1:  
 g = open(nume\_fis\_out, "a")  
 for j in range(len(rasp)):  
 g.write(str(rasp[j]))  
 g.write((" "))  
 g.write("\n")  
 g.close()  
 fis.close()  
  
  
cautare\_cuvant("floare", "rez.txt", "eminescu.txt", "paunescu.txt", "irina.txt")

SEMINAR 5

1. Problema candidatului majoritar.

Se consideră o listă v formată din n numere naturale nenule reprezentând voturile a n alegători. Să se afișeze, dacă există, câștigătorul alegerilor, adică un candidat care a obținut cel puțin [n : 2] + 1 voturi (candidatul majoritar).

Exemplu:

dacă 𝑣 = [1,5,5,1,1,5], atunci nu există niciun câștigător al alegerilor

dacă 𝑣 = [7,3,7,4,7,7], atunci candidatul 7 a câștigat alegerile

def vot\_maj(lista):  
 d = {}  
 for x in lista:  
 if x in d:  
 d[x] += 1  
 if d[x] > len(lista) // 2:  
 return x  
 else:  
 d[x] = 1  
 return None  
  
  
v1 =[1,5,5,1,1,5]  
v2 = [7,3,7,4,7,7]  
  
print(vot\_maj(v1))  
print(vot\_maj(v2))

def castigator (voturi):  
 avantaj = 0  
 castigator = None  
 for x in voturi:  
 if avantaj == 0:  
 castigator = x  
 avantaj = 1  
 elif x == castigator:  
 avantaj += 1  
 else:  
 avantaj -= 1  
 if avantaj != 0:  
 if voturi.count(castigator) > len(voturi) // 2:  
 return castigator  
 else:  
 return None  
 else:  
 return None  
  
v1 = [1,5,5,1,1,5]  
v2 = [7,3,7,4,7,7]  
  
print(castigator(v1))  
print(castigator(v2))

1. Se citește o listă de numere naturale sortată strict crescător și un număr natural 𝑆. Să se afișeze toate perechile distincte formate din valori distincte din lista dată cu proprietatea că suma lor este egală cu 𝑆.

Exemplu: pentru lista 𝐿 = [2,5,7,8,10,12,15,17,25] și 𝑆=20, trebuie afișate perechile (5,15) și (8,12).

L = [2, 5, 7, 8, 10, 12, 15, 17, 25, 27]  
S = 20  
  
for i in range(len(L)):  
 if L[i] >= S//2:  
 break  
 y = S - L[i]  
 st = i + 1  
 dr = len(L) - 1  
 while dr >= st:  
 mij = (st + dr)//2  
 if y > L[mij]:  
 st = mij + 1  
 else:  
 if y < L[mij]:  
 dr = mij -1  
 else:  
 print (L[i], y)  
 break

def perechi (L, S):  
 M = set(L)  
 for x in L:  
 if x >= S // 2:  
 break  
 if S - x in M:  
 print (x, S-x)  
  
L = [2, 5, 7, 8, 10, 12, 15, 17, 25, 27]  
S = 20  
perechi(L, S)

L = [2, 5, 7, 8, 10, 12, 15, 17, 25, 27]  
S = 20  
  
left, right = 0, len(L) - 1  
while left < right:  
 if L[left] + L[right] == S:  
 print(L[left], L[right])  
 left += 1  
 right += 1  
 elif L[left] + L[right] < S:  
 left += 1  
 else:  
 right -= 1

1. Fie 𝑛 intervale închise 𝐼1=[𝑎1,𝑏1],...,𝐼𝑛=[𝑎𝑛,𝑏𝑛]. Să se determine reuniunea intervalelor date, precum și lungimea sa.

LABORATOR 5

3. Turn de înălțime maximă format din cuburi

Se dă o mulțime de n cuburi. Fiecare cub este caracterizat prin lungimea laturii și culoare. Nu există două cuburi având aceeași dimensiune. Fișierul “cuburi.txt” conține pe prima linie un număr natural nenul n (numărul de cuburi), apoi pe următoarele n linii câte un număr natural nenul (lungimea laturii cubului) și un șir de caractere (culoarea cubului).

Să se construiască un turn de înălțime maximă astfel încât peste un cub cu latura L și culoarea K se poate așeza doar un cub cu latura mai mică strict decât L și culoare diferită de K. În fișierul “turn.txt” să se afișeze componența turnului de la bază spre vârf, pe câte un rând latura și culoarea cubului, apoi la final să se afișeze înălțimea totală a turnului.

|  |  |
| --- | --- |
| cuburi.txt | turn.txt |
| 6  7 verde  10 roșu  6 albastru  12 roșu  4 roșu  9 verde | 12 roșu  9 verde  6 albastru  4 roșu  Înălțime totala: 31 |

f = open("cuburi.txt")  
n = int(f.readline())  
lista = []  
for linie in f.readlines():  
 cub = linie.split()  
 lista.append((int(cub[0]), cub[1]))  
lista.sort(key=lambda t: -t[0])  
# lista.sort(reverse=True)  
f.close()  
  
g = open("turn.txt", "w")  
s = lista[0][0]  
cul = lista[0][1]  
# g.write(f"{lista[0][0]} {lista[0][1]}\n")  
g.write("{} {}\n".format(\*lista[0]))  
for i in range(1, len(lista)): # for cub in lista[1:]:  
 if lista[i][1] != cul:  
 cul = lista[i][1]  
 s += lista[i][0]  
 # g.write(f"{lista[i][0]} {lista[i][1]}\n")  
 g.write("{} {}\n".format(\*lista[i]))  
g.write(f"inaltime totala:{s}")  
g.close()

1. Plata unei sume folosind un număr minim de bancnote

Fie o mulțime de bancnote {B0,B1, ...,Bn} astfel încât B0 = 1 (avem mereu bancnota unitate, pentru a putea plăti orice sumă) și Bi I Bj, 0 <= i < j < n-2 (cu excepția ultimelor 2 bancnote, toate valorile se divid cu toate valorile din listă mai mici decât ele).

De exemplu se consideră bancnotele românești cu valorile {1,5,10,50,100,200,500}.

Pentru o sumă de bani S, să se determine o modalitate de a plăti suma S folosind un număr minim de bancnote.

Fișierul "bani.txt" conține pe prima linie valorile bancnotelor disponibile (se consideră că avem la dispoziție un număr infinit din fiecare bancnotă), iar pe a doua linie valoarea sumei S. În fișierul "plata.txt" să se afișeze ce bancnote cu valori diferite și câte din fiecare valoare s-au folosit pentru a achita suma S.

|  |  |
| --- | --- |
| bani.txt | plata.txt |
| 1 5 10 50 100 200 500  173 | 173 = 100\*1 + 50\*1 + 10\*2 + 1\*3 |

f = open("bani.txt")  
g = open("plata.txt", "w")  
  
lista = [int(x) for x in f.readline().split()]  
lista = sorted(lista, reverse=True)  
  
s = int(f.readline())  
g.write(str(s) + " = ")  
  
for x in lista:  
 if s == 0:  
 break  
 if x <= s:  
 numar = s // x  
 s -= numar \* x  
 # g.write(str(x)+"\*"+str(numar))  
 g.write(f"{x} \* {numar}")  
 if s > 0:  
 g.write("+")  
g.close()

1. Minimizarea timpului mediu de așteptare

Din fișierul "tis.txt" se citesc numere naturale nenule reprezentând timpii necesari pentru servirea fiecărei persoane care așteaptă la o coadă. Să se determine ordinea în care ar trebui servite persoanele de la coadă astfel încât timpul mediu de așteptare să fie minim.

Exemplu: dacă “tis.txt” conține numerele 7 15 3 7 8 3 2 10 5 4 2, atunci timpul mediu de așteptare inițial va fi: 41.73, iar timpul mediu de așteptare după sortarea listei va fi: 24.82.

INDICAȚII: Se creează o listă de tupluri care să conțină, pentru fiecare persoană, numărul său de ordine în lista inițială și timpul individual de servire pentru acea persoană. Se definește o funcție “afisare\_timpi\_servire (tis)” care primește ca parametru o listă de tupluri de tipul indicat mai sus și afișează pentru fiecare persoană, pe 3 coloane, următoarele informații: numărul de ordine al persoanei, timpul individual de servire și timpul său de așteptare. La final se afișează timpul mediu de așteptare al tuturor persoanelor. Se apelează funcția de mai sus pentru lista inițială, iar apoi pentru lista care a fost sortată … (după ce criteriu)?

|  |  |  |
| --- | --- | --- |
| Varianta inițială | | |
| Persoana | Timp individual | Timp cumulat |
| 1 | 7 | 7 |
| 2 | 15 | 22 |
| 3 | 3 | 25 |
| 4 | 7 | 32 |
| 5 | 8 | 40 |
| 6 | 3 | 43 |
| 7 | 2 | 45 |
| 8 | 10 | 55 |
| 9 | 5 | 60 |
| 10 | 4 | 64 |
| 11 | 2 | 66 |
| Timpul mediu de servire: 41.73 | | |

|  |  |  |
| --- | --- | --- |
| Varianta optimă | | |
| Persoana | Timp individual | Timp cumulat |
| 7 | 2 | 2 |
| 11 | 2 | 4 |
| 3 | 3 | 7 |
| 6 | 3 | 10 |
| 10 | 4 | 14 |
| 9 | 5 | 19 |
| 1 | 7 | 26 |
| 4 | 7 | 33 |
| 5 | 8 | 41 |
| 8 | 10 | 51 |
| 2 | 15 | 66 |
| Timpul mediu de servire: 24.82 | | |

LABORATOR 5

1. Se dă o scară cu n trepte (n citit de la tastatură). Un om aflat la baza scării (pe treapta 0) poate într-un pas să sară cate 1, 2 sau 3 trepte. In cate moduri poate ajunge pe treapta n pornind de pe treapta 0?

Exemplu: n=3 => 4 moduri (câte 1+1+1 sau 1+2 sau 2+1 sau 3 trepte sărite)

Folosim un vector cu n+1 poziții, unde:

v[i] = nr moduri in care omul poate ajunge de pe treapta 0 pe treapta i

Subproblemele cunoscute:

v[0] = 0, v[1] = 1, v[2] = 2, v[3]=4

Recurenta pentru 4 <= k <= n:

v[k] = v[k-1] + v[k-2] + v[k-3]

n = int(input("n = "))  
v = [0] \* (n + 1)  
# v[1] = 1  
# v[2] = 2  
# v[3] = 4  
v[1:4] = [1, 2, 4]  
  
for i in range(4, n + 1):  
 # v[i] = v[i - 1] + v[i - 2] + v[i - 3]  
 v[i] = sum(v[i - 3: i])  
  
print(\*v[1:])  
print(f"Ajungem pe treapta {n} in {v[n]} moduri.")

1. Conjunctura lui Collatz

Se dă o funcție f : N - N definită astfel:

f(n) = n / 2 dacă n este par și

f(n) = 3 \* n + 1 dacă n este impar

Pentru orice număr natural, dacă se aplică în mod repetat funcția f se ajunge la valoarea 1.

Pentru un număr n dat, să se afișeze după câți pași se ajunge prima dată la valoarea 1.

Exemplu: n=20 => 7 pași (20 - 10 - 5 - 16 - 8 - 4 - 2 - 1)

Vom utiliza un dicționar cu cheie = n, valoare = nr pași in care ajungem de la n la 1

Subprobleme știute: {1:0}

def f(val, n):  
 if n not in val:  
 val[n] = 1 + f(val, n // 2 if n % 2 == 0 else 3 \* n + 1)  
 return val[n]  
  
  
n = int(input("n = "))  
val = {1: 0}  
valoare = f(val, n)  
print(val)  
print(valoare)

1. Subsecvența de sumă maximă a unui șir: Se dă un șir de numere (în fișier, separate prin spații). Să se afișeze o subsecvenţă de sumă maximă a șirului (formată cu elemente consecutive)

date.in : v = 1 -2 3 -1 5 2 -6 1 3

date.out : 3 -1 5 2

Varianta 1:

current\_sum

max\_sum = -math.inf

start = 0, end = 0

current\_sum = max(current\_sum + x, x)

max\_sum = max(max\_sum, current\_sum)

Varianta 2:

Folosim un vector: s[i] = suma maxima a unei subsecvente care se termina pe pozitia i

Subproblema știută: s[0] = v[0]

Recurenta: s[k] = max{s[k-1] + v[k], v[k]}

# f = open("date.in", "r")  
# v = [int(x) for x in f.readline().split()]  
numere = "1 -2 3 -1 5 2 -6 1 3"  
v = [int(x) for x in numere.split()]  
  
s = [0] \* len(v)  
s[0] = v[0]  
  
for i in range(1, len(v)):  
 s[i] = max(s[i - 1] + v[i], v[i])  
  
print(\*s)  
print(f"Suma maxima a unei secvente: {max(s)}")  
  
i = s.index(max(s))  
res = []  
while v[i] != s[i]:  
 res.append(v[i])  
 i -= 1  
res.append(v[i])  
  
res.reverse()  
print(\*res)

1. Se dau două cuvinte s și t. Să se găsească un subșir comun de lungime maximă.

Exemplu: s = “SUBSIR”, t = “RUSTICE” => “USI”

Matrice cu (len(s)+1) linii si (len(t)+1) coloane

M[ i ][ j ] = lungimea maxima a unui subșir comun dintre prefixele de lungime i al lui s si lungime j al lui t

Subprobleme știute:

Punem 0 in matrice pe toata linia i=0 si toata coloana j=0

Recurenta, pentru 1 <= i <= len(s) si 1 <= j <= len(t):

M[i][j] = 1 + M[i-1][j-1], daca s[i-1] == t[t-1]

sau max {M[i][j-1] , M[i-1][j], daca s[i-1] == t[t-1]}

s, t = input("Cuvinte:").split()  
lungimi = [[0] \* (len(t) + 1) for i in range(0, len(s) + 1)]  
  
for i in range(1, len(s) + 1):  
 for j in range(1, len(t) + 1):  
 if s[i - 1] == t[j - 1]:  
 lungimi[i][j] = lungimi[i - 1][j - 1] + 1  
 else:  
 lungimi[i][j] = max(lungimi[i - 1][j], lungimi[i][j - 1])  
  
print("-", "-", \*t)  
af\_s = ["-", \*s]  
for lindex in range(0, len(s) + 1):  
 print(af\_s[lindex], \*lungimi[lindex])  
  
i = len(s)  
j = len(t)  
sol = []  
  
while lungimi[i][j] != 0:  
 if s[i - 1] == t[j - 1]:  
 sol.append(s[i - 1])  
 j -= 1  
 i -= 1  
 elif lungimi[i - 1][j] >= lungimi[i][j - 1]:  
 i -= 1  
 else:  
 j -= 1  
  
sol.reverse()  
sol = "".join(sol)  
print(f"Un subsir comun maximal este: {sol}")