<https://colab.research.google.com/drive/1oJCvIa2sbfdYu8p7ay6xFBl4i34Re4Na?usp=sharing#scrollTo=MGmuQHLNQRY3>

1. ***P-rime***

Doua cuvinte sunt p-rime daca au ultimele p litere egale. Scrieti o functie care primeste doua cuvinte si returneaza p maxim pentru care cuvintele sunt p-rime.

s1 = “niciodata”  
s2 = “fata”

find\_p\_rime(s1, s2) == 3

Idee:

i) inversati string-urile si parcurgeti-le simultan

ii) incrementati un contor cat timp literele de la final sunt egale

iii) literele sunt diferite sau daca s-a terminat un string - iesiti

iv) returnati contorul

s1=input("Primul cuvant:")  
s2=input("Al doilea cuvant:")  
nr=0  
x=min(len(s1),len(s2))+1  
for i in range (1,x):  
 if (s1[-i]==s2[-i]):  
 nr+=1  
 else:  
 break  
print(nr)

1. ***Limba pasareasca***

In limba pasareasca, de fiecare data cand avem o vocala intr-o propozitie vom concatena litera p si acea vocala. Exemplu:

masa     = mapasapa  
iepure   = iepepupurepe

Primind un string in limba romana, scrie o functie encrypt care converteste string-ul in limba pasareasca si o functie decrypt care converteste un string din limba pasareasca in limba romana (daca se poate). Pentru verificare folositi: s == decrypt(encrypt(s))

Idee criptare:

i) parcurgeti string-ul initial

ii) cand intalniti o vocala salvati in string-ul nou v urmat de pv

iii) returnati noul string

Idee decriptare:

i) parcurgeti string-ul

ii) cand apare o vocala, verificati daca urmatoarele litere sunt pv

**#CRIPTARE**

s=input("Cuvant:")  
sc=0  
nr=1  
for i in range (1,len(s)):  
 i=nr  
 if s[i]=='a' or s[i]=='e' or s[i]=='i' or s[i]=='o' or s[i]=='u':  
 sc=s[i]+'p'+s[i]  
 s=s[:i]+sc+s[i+1:]  
 nr=i+3  
 else:  
 nr=i+1  
print(s)

Sau

def decrypt(s):  
 decrypted = ""  
 i = 0 # parcurgem cu while  
 while i < len(s): # pentru ca trebuie sa sarim  
 c = s[i] # extragem caracterul curent  
 if is\_voyel(c): # daca este vocala  
 if (i+1 < len(s) and i+2 < len(s) and # ne asiguram ca exista inca  
 s[i+1] == 'p' and s[i+2] == c): # doua litere dupa si sunt px  
 decrypted += c # adaugam litera  
 i += 3 # sarim peste caracterul curent, p si x  
 else:  
 return False # altfel este o eroare in string  
 else:  
 decrypted += c # e consoana, nu conteaza  
 i += 1 # mergem la urmatorul caracter  
 return decrypted  
  
def decrypt(s):  
 # presupunem ca s este corect  
 for vocala in "aeiou":  
 s = s.replace(vocala + "p" + vocala, vocala)  
 return s  
  
s = "masa"  
print(encrypt(s), decrypt(encrypt(s)))

1. Poezie

Primiti o poezie, trebuie sa gasiti p cu prop. ca versurile sunt p-rime

poezie = “””A fost odată ca-n poveşti,  
A fost ca niciodată.  
Din rude mari împărăteşti,  
O prea frumoasă fată.”””

print(find\_p\_rime(poezie))

Primiti poezia fara diacritice, dar cu semne de punctuatie!

Idee:

i) scapati de semnele de punctuatie cu .replace(semn, ‘’);

ii) faceti un split pe versuri .split(“ “);

iii) pentru fiecare tip de rima, folositi functia de la problema 1 pentru a gasi p-ul;

iv) luati valoarea maxima care este respectata de toate versurile;

Bonus:

rezolvati problema care contine toata poezia (nu primiti doar o strofa,

ci toata poezia; idee: faceti split initial pe strofe .split(“\n\n”) si

folositi functia de mai sus care merge pe strofe.

# p\_rime("povesti", "imparatesti") = 4  
def p\_rime(s\_1, s\_2):  
 s\_1 = s\_1[::-1]  
 s\_2 = s\_2[::-1]  
  
 cnt = 0  
 for i in range(len(s\_1)):  
 if s\_1[i] != s\_2[i]:  
 break  
 cnt += 1  
  
 return cnt  
  
  
poezie = """A fost odata ca-n povesti,  
A fost ca niciodata.  
Din rude mari imparatesti,  
O prea frumoasa fata.  
  
A fost odata ca-n povesxti,  
A fost ca niciodata.  
Din rude mari imparatesti,  
O prea frumoasa fata."""  
  
  
def p\_strofe(strofa):  
 semne = ',.?!;'  
 for i in semne:  
 strofa = strofa.replace(i, '')  
  
 versuri = strofa.split("\n")  
  
 # imperechetata  
 p\_1 = p\_rime(versuri[0], versuri[1])  
 p\_2 = p\_rime(versuri[2], versuri[3])  
  
 min\_1 = min(p\_1, p\_2)  
  
 # imbratisata   
 p\_1 = p\_rime(versuri[0], versuri[3])  
 p\_2 = p\_rime(versuri[1], versuri[2])  
  
 min\_2 = min(p\_1, p\_2)  
  
 # incrucisata   
 p\_1 = p\_rime(versuri[0], versuri[2])  
 p\_2 = p\_rime(versuri[1], versuri[3])  
  
 min\_3 = min(p\_1, p\_2)  
  
 raspuns = max(min\_1, min\_2, min\_3)  
 return raspuns  
  
  
strofe = poezie.split('\n\n')  
minim = 100  
for strofa in strofe:  
 p = p\_strofe(strofa)  
  
 minim = min(minim, p)  
print(minim)

1. Comprimare si decomprimare

Primiti un string fara spatii in care literele se repeta des. Scrieti o functie care comprima string-ul astfel:

“aabbbbaccc”    =>    “a2b4a1c3”

Scrieti o functie de decomprimare care primeste string-ul comprimat si il intoarce la varianta initiala.

def comprimare(s):  
 # salvam caracterul curent sa stim daca se repeta  
 caracter\_crt = s[0]  
 cnt = 0  
  
 str\_comprimat = ''  
  
 for c in s: # pentru fiecare caracter din string  
 if c == caracter\_crt: # daca este caracterul curent il contorizam  
 cnt += 1  
 else: # daca nu inseamna ca s-a schimbat caracterul  
 str\_comprimat += caracter\_crt + str(cnt) # salvam comprimarea  
  
 caracter\_crt = c # modificam caracterul curent  
 cnt = 1 # aceasta a aparut o data (acum)  
  
 if cnt != 0:  
 str\_comprimat += caracter\_crt + str(cnt)  
 return str\_comprimat  
  
  
def comprimare(s):  
 # gasim indicii unde apar litere diferite  
 idx\_inceput = [0]  
 idx\_sfarsit = []  
 for i in range(len(s) - 1):  
 if s[i] != s[i + 1]:  
 idx\_sfarsit.append(i)  
 idx\_inceput.append(i + 1)  
 idx\_sfarsit.append(len(s) - 1)  
  
 comprimat = ''  
 for inc, sf in zip(idx\_inceput, idx\_sfarsit):  
 comprimat += s[inc] + str(sf - inc + 1)  
  
 return comprimat  
  
  
def decomprimare(s):  
 str\_decomprimat = ""  
  
 i = 0  
 while i < len(s):  
 # extragem litera  
 litera = s[i]  
  
 # salvam in str\_nr numarul (poate avea mai multe cifre)  
 str\_nr = ''  
 while i + 1 < len(s) and s[i + 1].isnumeric():  
 str\_nr += s[i + 1]  
 i += 1  
  
 # adaugam litera in string-ul decomprimat de str\_nr ori  
 str\_decomprimat += litera \* int(str\_nr)  
 i += 1  
  
 return str\_decomprimat  
  
  
def decomprimare(s):  
 decomprimat = ''  
  
 idx\_litere = [] # gasim pozitiile literelor din string  
 for i in range(len(s)):  
 if s[i].isalpha():  
 idx\_litere.append(i)  
 idx\_litere.append(len(s)) # adaugam o litera fictiva pentru sfarsit numarului  
  
 for i in range(len(idx\_litere) - 1): # pentru fiecare indice  
 idx\_litera = idx\_litere[i] # extragem litera  
 idx\_start\_nr = idx\_litere[i] + 1 # extragem inceputul numarului  
 idx\_end\_nr = idx\_litere[i + 1] - 1 # extragem sfarsitul numarului  
  
 decomprimat += s[idx\_litera] \* int(s[idx\_start\_nr:idx\_end\_nr + 1])  
  
 return decomprimat  
  
  
s = "aabbbbaccc"  
print(s, comprimare(s), decomprimare(comprimare(s)))

1. Rot(13)

Primiti un string s si un numar k. Trebuie sa rotiti fiecare litera din s cu k pozitii mai la dreapta.

De exemplu:

s = “bec” k = 3 =>  
rot(s, k) = “ehf”  
irot(“ehf”, k) = “bec”

s = "bec"  
k = 3  
  
def rot(s, k):  
 r = ""  
 for c in s:  
 # trebuie sa avansam cuvantul k litere la dreapta  
 # il convertim la ASCII, ii adaugam k pozitii,   
 # facem modulo 26 pt depasiri si il convertim inapoi in string  
 idx\_caracter\_nou = (ord(c) + k - ord('a')) % 26  
 r += chr(idx\_caracter\_nou + ord('a'))  
 return r  
  
def irot(s, k): # putem face ca la rot cu "-"  
 return rot(s, 26-k) # sau rotim de 26-k ori (operatia %)  
  
print(rot(s, k), irot(rot(s, k), k))