**TP 4**

**Resume sur les operations des ensembles :**

#listes

a=[1,2,3,4,5,6,7]

a.append(8)  #---> a=[1,2,3,4,5,6,7,8]

a.extend([9,10])  #---->a=[1,2,3,4,5,6,7,8,9,10]

a.insert([0,0])  #----> a=[0,1,2,3,4,5,6,7,8,9,10]

a.remove(10)  #----->a=[0,1,2,3,4,5,6,7,8,9]

a.sort()    #----->a=[0,1,2,3,4,5,6,7,8,9] #trie

a.reverse()   #-------> #ma9louba

a.count(1) #------->1 #9adeh min marra mawjouda

#dict

dic={'Nom' : 'Ben Abdallah', 'Prenom' : 'Abdallah'}

monDictionnaire={'Nom' : 'Ben Abdallah', 'Prenom' : 'Abdallah', 'Age' : 20}

dic.items()  #traja3 (clé,valeur).

monDictionnaire.items() #------>dict\_items([('Nom', 'Ben Abdallah'), ('Prenom', 'Abdallah'), ('Age', 20)])

dic.keys()  #Renvoie le cle

monDictionnaire.keys()  #----> dict\_keys(['Nom', 'Prenom', 'Age'])

dic.values()  #Renvoie les valeurs

monDictionnaire.values()  #---->dict\_values(['Ben Abdallah', 'Abdallah', 20])

dic.pop()  #Renvoie la valeur et supprime la paire clé/valeur(KeyError si la clé est absente).

                #i7otou fi blasa okhra so iwali inexistant

resPop=monDictionnaire.pop('Age')  #---->{'Nom': 'Ben Abdallah', 'Prenom': 'Abdallah'}

#ensembles

ens={"a","b","c"}

ens1={}

ens2={}

ens.add()  #Ajoute un élément à l’ensemb.

ens.remove()  #Retire un élément à un ensemble(KeyError si élément absent).

ens.clear()  #supprime tout les élément d’un ensemble.

ens1.difference(ens2)  #Renvoie l’ensemble des éléments de ens1 qui n’appartiennent pas à ens2.

ens1.issuperset(ens2)  #Renvoie True si ens1 contient ens2    E appartient

ens1.issubset(ens2)  #Renvoie True si ens1 est inclu dans ens2  C inclu

ens1.union(ens2)  #Renvoie l’union des deux ensembles.  U union

ens1.intersection(ens2)  #Renvoie l’intersection des deux ensembles ens1 et ens2

len() #longueur

max()    #   min()       sum()

MaListe=[1,3,4,2.3,3.4]

print("Pour la liste {} max={} min={} sum={}",format(MaListe,max(MaListe),min(MaListe),sum(MaListe)))

#------>Pour la liste [1, 3, 4, 2.3, 3.4] max=4 min=1 sum=13.700000000000001

print(list(range(0,10,1))) #--->[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#tuples

Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])

Months={"Jan","Feb","Mar"}

Dates={21,22,17}

print(Days)

print(Months)

print(Dates)

#string.strip([chars]) #----> #supprime chars mil string

myTuple = ("John", "Peter", "Vicky")

x = "#".join(myTuple)

print(x) # equivalent lil extend fil liste

l.upper()   #majuscule

l.lower()   #miniscule

l.pop()     #---->next one

**Exercice 1:**

#Les collections:

#liste l=[ 1,2] g0 d-1 mutables

#tuples t=(1,2) non mutables

#ensembles e={} pas doublons , mutables

#dictionnaires clé valeurs keys values ,mutables , cles immutables

#d={'k1':v1 , 'k2':v2 }

#d=dict([('k1',v1),('k2',v2)])

Joueur = ('A','B')

Score={'A':0,'B':0}

Tour=input('Qui va commencer ? A/B \t')

while(Tour.upper() not in 'AB'):

    Tour=input('Qui va commencer ? A/B \t')

print(Tour)

while(abs(Score['A']-Score['B'])<2):

    C=0

    while(C<20):

        print("C'est le tour du joueur",Tour.upper())

        x=input('Saisir un entier 1 ou 2 ou 3\t')

        while(x not in '123'):

            x=input('Saisir un entier 1 ou 2 ou 3\t')

        C+=int(x)

        print('Compteur =',C)

        if(Tour.upper()=='A'):

            Tour='B'

        else:

            Tour='A'

    Score[Tour.upper()]+=1

    ProchainTour=Joueur-{Tour.upper()}

    Tour=ProchainTour.pop()

if(Score['A']==max([Score.value()]):

    print('Le gagnant est A')

else:

    print('Le gagnant est B')

**Exercice 2:**

ch="""Les fonctions nous permettent de regrouper des linges de code dans un mini-programme appelé sous-programme.

Ensuite, chaque fois que nous avons besoin de l’utiliser, nous «appelons» simplement cette fonction ; appeler une fonction signifie l’utiliser dans un autre code.

Généralement, une fonction prend une entrée et produit une sortie. Python offre deux catégories de fonction : les fonctions ordinaires et les fonctions lambda."""

alpha={'k', 'x', 'c', 'é', 'v', 'è', 'a', 'w', 'à', 'p', 'd', 'ù', 'g', 'm', 'b', 'l', 'h', 'o', 'r', 'q', 'u', 'i', 't', 'f', 'ç', 'n', 'î', 'ê', 'y', 's', 'j', 'z', 'ï', 'e'}

ponct={',', '?', '.', ':', ';','!'}

# 1. Ecrivez le scripty python permettant d'afficher les caractères de ponctuation utilisés dans \*\*ch\*\*.

e1=(set(ch.lower())-alpha)&ponct

print(e1)

# 2. Ecrivez le scripty python permettant d'afficher les lettres utilisées dans \*\*ch\*\*. Ne différenciez pas les lettres majuscules des minuscules.

e2=set(ch.lower())&alpha

print(e2)

# 3. Ecrivez le scripty python permettant d'afficher les caractères utilisés dans \*\*ch\*\* qui sont ni de ponctuation ni des lettres .

e3=set(ch.lower())-(alpha|ponct)

print(e3)

# 4. Afficher l'ensemble de mots utilisés dans ch et le nombre d'apparition de chaque mot.

l=ch.split()

print(l)

#supprimer la ponctuation à la fin de chaque mot str.strip([chars])  str.join(iterable)

#print (''.join(ponct)) #!?.:,;

#'?fghj;'.strip(''.join(ponct))

ll=[mot.strip(''.join(ponct)) for mot in l]

print(ll)

for i in range(ll.count('')):

    ll.remove('')

print(ll)

# 4.2 le nombre d'apparition de chaque mot.

ensMots=set(ll)

d=dict()

for m in ensMots:

    d[m]=ll.count(m)

print(d)

**Exercice 3:**

Tavion=('avion1','avion2','avion3','avion4')

Tcapavion=(50,120,250,150)

Tville=('Tunis','Sfax')

#Q1

#exp DictVitesse = {'avion1' :300,'avion2' :450,'avion3' :550,'avion4' :450 }

DictVitesse=dict()

for i in range (len(Tcapavion)):

    if(Tcapavion[i]<100):

        DictVitesse(Tavion[i])=300

    elif(Tcapavion[i]<200):

        DictVitesse(Tavion[i])=450

    elif(Tcapavion[i]>=200):

        DictVitesse(Tavion[i])=550

print(DictVitesse)

#Q2

#exp DictVol = {'TunSfa0' : ['Tunis','Sfax'] , 'SfaTun1' : ['Sfax','Tunis'] }

DictVol=dict()

for i in range (len(Tville)):

    for j in range (len(Tville)):

        if Tville[i]!=Tville[j]:

            code=Tville[i][:3]+Tville[j][:3]+str(i)

            DictVol[code]= [Tville[i],Tville[j]]

print(DictVol)

#Q3

#exp DictVol = {'TunSfa0' : ['Tunis','Sfax', 350 , 'avion1'] , 'SfaTun1' : ['Sfax','Tunis', 350 , 'avion2'] }

for i in DictVol.keys():

    print("depart : ",DictVol[i][0],"arrivée : ",DictVol[i][1])

    DictVol[i].append(int(input("donner la distance entre deux les villes :")))

    DictVol[i].append(input("donner le nom de l'avion :"))

print(DictVol)

#Q4

#exp DictVitesse = {'avion1' :300,'avion2' :450,'avion3' :550,'avion4' :450 }

#exp DictVol = {'TunSfa0' : ['Tunis','Sfax', 350 , 'avion1'] , 'SfaTun1' : ['Sfax','Tunis', 350 , 'avion2'] }

#DictDuree={'TunSfa0' : [1,04] , 'SfaTun1' : [0,46]}

DictDuree=dict()

for c in DictVol.keys():

    duree=60\*DictVol[c][2]//DictVitesse[DictVol[c][3]]

    DictDuree[c]=[duree//60,duree%60]

print(DictDuree)

#Q5

codeVol=input("donner le code du vol")

Tdate=tuple()

hv=int(input("l'heure de depart H="))

mv=int(input("la minute de depart M="))

Tdate=(hv,mv)

#

h=Tdate[0]+DictDuree[codeVol][0]

m=Tdate[1]+DictDuree[codeVol][1]

if(m>=60):

    h+=m//60

    m=m%60

if(h>=24):

    h=h-24

    print("arrivée le lendemain à",h,"heures",m,"minutes")

else:

    print("arrivée à",h,"heures",m,"minutes")

**TP 5**

**Exercice 1:**

def tri\_selection(tab):

   for i in range(len(tab)):

      # Trouver le min

       min = i

       for j in range(i+1, len(tab)):

           if tab[min] > tab[j]:

               min = j

       tmp = tab[i]

       tab[i] = tab[min]

       tab[min] = tmp

   return tab

# Programme principale pour tester le code ci-dessus

tab = [98, 22, 15, 32, 2, 74, 63, 70]

tri\_selection(tab)

print ("Le tableau trié est:")

for i in range(len(tab)):

    print ("%d" %tab[i])

########################################################

def triSelect(l):

    n=len(l)

    for i in range(n):

        minListe=min(l[i-n:])

        posMinListe=l.index(minListe,i,n)

        l[i],l[posMinListe]=l[posMinListe],l[i]

ll=[]

#Boucle infinie pour remplir la liste

while True:

    d=input("-->")

    if d==' ':

        break

    else:

        ll.append(int(d))

triSelect(ll)

print(ll)

**Exercice 2:**

#Q1: le dictionnaire sera {'a':2, 'b': 3, 'interval':(1,3)}

def segment (a,b,extr1,extr2):

    return ({'a':a,'b':b,'interval':(min(extr1,extr2),max(extr1,extr2))})

#Q2 "lambda" [parameter\_list] ":" expression

#retourne une liste de couple de valeurs entières; coordonnées (x, y) des points apparent tq y=ax+b

f=lambda d:[(x,d['a']\*x+d['b']) for x in range(d['interval'][0],d['interval'][1]+1)]

#Q3: deux droites sont parallèles si les valeurs absolues de leurs coefficients |a| sont égaux.

def parallel(seg1,seg2):

    return abs(seg1['a'])==abs(seg2['a'])

#Q4:le pourcentage en terme de nombre de points apparents communs entre deux segments.

#La formule est : nombre de points apparents communs divisé par la somme des nombres de points apparents des deux segments. Sinon, elle renvoie 0.

def confondus(seg1,seg2):

    if parallel(seg1,seg2):

        pourcentage= len(set(f(seg1))&set(f(seg2)))\*100/(len(set(f(seg1)))+len(set(f(seg2))))

        return round(pourcentage,2)

#Q5: renvoie le point apparent d'intersection entre deux segments s'il existe, sinon renvoie 0.

def InterPointApparent(seg1,seg2):

    res=set(set(f(seg1))&set(f(seg2))

    if (res==set()):

        return 0

    return res

#Q6:

#     A. Calculez le déterminant \*\*det = (xA − xB)(yC − yD) − (xC − xD)(yA − yB)\*\*

#     B. Calculez \*\*t1 = ((xC − xB)(yC − yD) − (xC − xD)(yC − yB))/det\*\*

#     C. Calculez \*\*t2 = ((xA − xB)(yC − yB) − (xC − xB)(yA − yB))/det\*\*

#     D. Les deux segments se coupent ssi t1 et t2 appartiennent simultanément à [0, 1].

#exple l1=[(-3, -7), (-2, -4), (-1, -1), (0, 2), (1, 5), (2, 8), (3, 11)]

#[A,B]=[(xA,yA),(xB,yB),,,,,]    xA=l1[i][0] yA=l1[i][1] xB=[i+1][0] yB=l1[i+1][1]

#[C,D]=[(xC,yC),,,,,,,(xD,yD)]    xC=l2[0][0] yC=[0][1]    xD=[-1][0]  yD=[-1][1]

def interPointDisc(seg1,seg2):

    l1=f(seg1)

    l2=f(seg2)

    res=[]

    for i in range(len(l1)-1):

        det = (l1[i][0] - l1[i+1][0])\*(l2[0][1] - l2[-1][1]) - (l2[0][0] - l2[-1][0])\*(l1[i][1] - l1[i+1][1])

        t1 = ((l2[0][0] - l1[i+1][0])\*(l2[0][1] - l2[-1][1]) - (l2[0][0] - l2[-1][0])\*(l2[0][1] - l1[i+1][1]))/det

        t2 = ((l1[i][0] - l1[i+1][0])\*(l2[0][1] - l1[i+1][1]) - (l2[0][0] - l1[i+1][0])\*(l1[i][1] - l1[i+1][1]))/det

        if 0<t1<1 and 0<t2<1:

            res.append((l1[i],l1[i+1]))

            break

    if res!=[]:

        for i in range(len(l2)-1):

            det = (l2[i][0] - l2[i+1][0])\*(l1[0][1] - l1[-1][1]) - (l1[0][0] - l1[-1][0])\*(l2[i][1] - l2[i+1][1])

            t1 = ((l1[0][0] - l2[i+1][0])\*(l1[0][1] - l1[-1][1]) - (l1[0][0] - l1[-1][0])\*(l1[0][1] - l2[i+1][1]))/det

            t2 = ((l2[i][0] - l2[i+1][0])\*(l1[0][1] - l2[i+1][1]) - (l1[0][0] - l2[i+1][0])\*(l2[i][1] - l2[i+1][1]))/det

            if 0<t1<1 and 0<t2<1:

                res.append((l2[i],l2[i+1]))

                break

        return res

    return 0

def etude2Segments(seg1,seg2):

    if parallel(seg1,seg2):

        temp=PourConf(seg1,seg2)

        if temp==0:

            print("Les deux segment sont strictement parallèles")

        else:

            print("Les deux segment sont confondus de {}%",format(temp))

    else :

        temp=InterPointApparent(seg1,seg2)

        if temp!=0:

            print("L'intersection des deux segments est :",temp)

        else:

            temp=interPointDisc(seg1,seg2)

                if temp!=0:

                    print("L'intersection des deux segments se trouve entre les points :",temp)

                else:

                    print("Les deux segments ne se coupent pas et ne sont pas parallèle")

# Programme principale pour tester le code ci-dessus

seg1=Segment(3,2,-3,14)

seg2=Segment(-2,1,-5,14)

seg3=Segment(3,2,0,4)

seg4=Segment(1,2,-3,14)

etude2Segments(seg1,seg3)

etude2Segments(seg1,seg2)

etude2Segments(seg3,seg2)

etude2Segments(seg1,seg4)

**Exercice 3:**

#find(chaine[, start[, end]]) renvoie la première occurence de chaine

#1. toutes les occurences de chaine ds texte

def findAll1(chaine,texte):

    l=[]

    i=0

    n=len(texte)

    while i<n:

        pos=texte.find(chaine,i)

        if pos==-1:

            break

        l.append(pos)

        i=pos+1

    if l==[]:

        return -1

    return l

#2.

def findAll(chaine,texte, start=0, end=-1):

    l=[]

    i=start

    n=len(texte[:end])

    while i<n:

        pos=texte.find(chaine,i,end)

        if pos==-1:

            break

        l.append(pos)

        i=pos+1

    if l==[]:

        return -1

    return l

print(findAll('a',"abcdajhoaajzbdkjzhfak"))

print(findAll(texte="abcdajhoaajzbdkjzhfak",chaine='x'))

print(findAll('a',"abcdajhoaajzbdkjzhfak",9))

print(findAll('a',"abcdajhoaajzbdkjzhfak",9,15))

f=findAll

print(f('a',"abcdajhoaajzbdkjzhfak"))

#3.

def isCorCapPonc (text):

    n=len(text)

    #1. La première lettre du texte est en majuscule.

    if (text[0].islower()):

        return False

    #2. On ne met pas d'espace avant les symboles de ponctuations ".", ",", "!" et "?"

    tmp='.,!?'

    for e in tmp:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(text[i-1]==" "):

                    return False

    #3. On doit mettre une espace après les symboles de ponctuations ".", ",", "!", "?", ":" et ";"

    tmp2='.,!?:;'

    for e in tmp2:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-1 and text[i+1] not in " \n"):

                    return False

    #4. La lettre juste après les symboles de ponctuations ".", "!" et "?" doit être en majuscule.

    tmp3='.!?'

    for e in tmp3:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-2 and text[i+2].islower()):

                    return False

    #5. On doit mettre une espace avant ":" et ";"

    tmp4=':;'

    for e in tmp4:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(text[i-1]!=" "):

                    return False

    #6. La lettre juste après les symboles de ponctuations ",", ":" et ";" doit être en minuscule.

    tmp5=',:;'

    for e in tmp5:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-2 and text[i+2].isupper()):

                    return False

    return True

#Q4.

def locErreur (text):

    d={'A':[],'B':[],'C':[],'D':[],'E':[],'F':[]}

    n=len(text)

    #1. La première lettre du texte est en majuscule.

    if (text[0].islower()):

        d['A'].append(0)

    #2. On ne met pas d'espace avant les symboles de ponctuations ".", ",", "!" et "?"

    tmp='.,!?'

    for e in tmp:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(text[i-1]==" "):

                    d['B'].append(i-1)

    #3. On doit mettre une espace après les symboles de ponctuations ".", ",", "!", "?", ":" et ";"

    tmp2='.,!?:;'

    for e in tmp2:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-1 and text[i+1] not in " \n"):

                    d['C'].append(i+1)

    #4. La lettre juste après les symboles de ponctuations ".", "!" et "?" doit être en majuscule.

    tmp3='.!?'

    for e in tmp3:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-2 and text[i+2].islower()):

                    d['D'].append(i+2)

    #5. On doit mettre une espace avant ":" et ";"

    tmp4=':;'

    for e in tmp4:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(text[i-1]!=" "):

                    d['E'].append(i)

    #6. La lettre juste après les symboles de ponctuations ",", ":" et ";" doit être en minuscule.

    tmp5=',:;'

    for e in tmp5:

        l=findAll(e,text)

        if(l!=-1):

            for i in l:

                if(i<n-2 and text[i+2].isupper()):

                    d['F'].append(i+2)

    for r in d:

        if d[r]==[]:

            d[r]='verifiée'

        else:

            d[r].sort()

    return d

#Q5

#5 exple d={'A': [0], 'B': [36, 48, 101, 166], 'C': [103, 168, 200], 'D': [104, 169], 'E': [199], 'F': [51]}

def CorCapPonc (text,d):

    listText=list(text)

    #1. La première lettre du texte est en majuscule.

    if d['A']!='verifiée':

        listText[0]=listText[0].upper()

    #2. On ne met pas d'espace avant les symboles de ponctuations ".", ",", "!" et "?"

    if(d['B']!='verifiée'):

        supp=0

        for i in d['B']:

            del listText[i-supp]

            supp+=1

    #3. On doit mettre une espace après les symboles de ponctuations ".", ",", "!", "?", ":" et ";"

    textApresModif=''.join(listText)

    dicApresModif=locErreur(textApresModif)

    listText=list(textApresModif)

    if(dicApresModif['C']!='verifiée'):

        add=0

        for i in dicApresModif['C']:

            listText.insert(i+add," ")

            add+=1

    #4. La lettre juste après les symboles de ponctuations ".", "!" et "?" doit être en majuscule.

    textApresModif=''.join(listText)

    dicApresModif=locErreur(textApresModif)

    listText=list(textApresModif)

    if(dicApresModif['D']!='verifiée'):

        for i in dicApresModif['D']:

            listText[i]=listText[i].upper()

    #5. On doit mettre une espace avant ":" et ";"

    if(dicApresModif['E']!='verifiée'):

        add=0

        for i in dicApresModif['E']:

            listText.insert(i+add," ")

            add+=1

    #6. La lettre juste après les symboles de ponctuations ",", ":" et ";" doit être en minuscule.

    textApresModif=''.join(listText)

    dicApresModif=locErreur(textApresModif)

    listText=list(textApresModif)

    if(dicApresModif['F']!='verifiée'):

        for i in dicApresModif['F']:

            listText[i]=listText[i].lower()

    res=''.join(listText)

    return res

#Q6.

ch="nous souhaitons tester ce paragraphe , pour cela , Nous avons commis plusieurs fautes de ponctuations .voyons alors notre correcteur est t'il capable de les corrigées ?bien-sûr au niveau de détection:Il détecte certaines erreurs et d'autres aprés la corrections."

print(ch)

print("----------------------------------------------")

print(isCorCapPonc(ch))

print("----------------------------------------------")

erreurs=locErreur(ch)

print(erreurs)

print("----------------------------------------------")

ch\_Corrigee=CorCapPonc(ch,erreurs)

print(ch\_Corrigee)

print("----------------------------------------------")

print(isCorCapPonc(ch\_Corrigee))