Cette fiche de TP était très (trop) longue. On se contente des exercices abordés en TP

Exercice 1

On calcule la somme des diviseurs stricts, dans S puis on teste l'égalité à n

```
def parfait(n):

    S=0
    for k in range(1,n): #on teste les diviseurs de 1 à n-1
        if n%k==0:  #si k divise n
            S=S+k
    if S==n:
        return True
    else:
        return False
```

Exercice 2

1)
```
#Avec un parcours par indice

def moyenne(L):
    S = 0
    for k in range(len(L)):
        S=S+L[k]
        return S/len(L)

#Avec un parcours par élément

def moyenne(L):
        S = 0
        for e in L:
                S=S+e
        return S/len(L)
```

2)
```
#Un parcours par élément fonctionne


def minimum(L):
        m = L[0]  #initialisation du minimum
        for e in L: #le score est battu
                if e<m:
                        m=e
        return m
```

3)
```
#Un parcours par indice est plus approprié


def minimum_position(L):
        m = L[0]  #initialisation du minimum
        ind=0     #et de son indice
        for k in range(len(L)):
                if L[k]<m:  #le score est battu
                        m=L[k]
                        ind=k
        return m,ind
```

4) *#Un seul parcours suffit, M1 désigne le grand max et M2 le petit max,à chaque*

#on compare l'élément de la liste à M1 et M2, trois cas selon qu'il est plus grand

#que M2, entre M1 et M2 ou plus petit que M1

```
def maximums(L):
        if L[0]<=L[1]:
                M1,M2 = L[1],L[0]
        else:
                M1,M2 = L[0],L[1]
        for k in range(2,len(L)):
                if L[k]>M1:
                        M2=M1
                        M1=L[k]
                elif M2<L[k]<=M1:
                        M2=L[k]
        return M1,M2
```

5) *#Dès que deux éléments consécutifs sont mal rangés la liste est mal triée*

```
def est_triee(L):
        for k in range(len(L)-1):
                if L[k+1]<L[k]:
                        return False
        return True

#Que penser de :
def est_triee(L):
        for k in range(len(L)-1):
                if L[k+1]<L[k]:
                        return False
                else:
                        return True
```

#C'est faux la fonction est interrompu et renvoie True dès que deux éléments consécuti

#sont bien triés sans être certains qu'ils le soient tous

6) Un parcours suffit

```
def longueur_triee(L):
        lmax = 0 #la longueur maximale
        Lmax = 0 #la liste de longueur maximale
        Lt = [L[0]]  #liste temporaire
        for k in range(1,len(L)):
                if L[k-1]<=L[k]:  #si la liste est bien triée
                        Lt.append(L[k])
                else:  #rupture de de croissance
                        if len(Lt)>lmax: #si record battu
                                lmax = len(Lt)
                                Lmax = Lt
                                Lt=[L[k]]
        return lmax,Lmax
```

7)
```
def partitionner(L):
        L1=[] #la liste des éléments positifs
        L2=[] #la liste des éléments strictement négatifs
```

```
        for e in L:
                if e>=0:
                        L1.append(e)
                else:
                        L2.append(e)
        return L1,L2
```

Exercice 4

```
#M1 : En faisant un parcours complet on compare les caractères symétriques

def palindrome(ch):
        n=len(ch)
        for k in range(n):
                if ch[k] != ch[n-1-k]:
                        return False
        return True

#M2 : Il suffit d'un parcours jusqu'à la moitié de la chaîne

def palindrome(ch):
        n=len(ch)
        for k in range(n//2):
                if ch[k] != ch[n-1-k]:
                        return False
        return True
```

Exercice 5

1)
```
def compte_lettres(ch):
        D = {} #le dictionnaire des occurences
        for lettre in ch:
                if lettre in D:
                        D[lettre]+=1
                else:
                        D[lettre]=1
        return D
```

2)
```
def majoritaire(ch):

        D = compte_lettres(ch) #on crée le dictionnaire des occurences
        #Puis un algo de calcul de maximum sur un dictionnaire
        M = 0 #nombre maximal
        lettreM = []  #liste des lettres plus présentes
        for lettre in D:
                if D[lettre]>M:     #record battu
                        M = D[lettre]
                        lettreM = [lettre]
                elif D[lettre] == M:  #record égalé
                        lettreM.append(lettre)
        return M,lettreM
```

3)
```
def longueur_mots(ch):
        D = {}  #le dictionnaire des longueurs de mots
        cpt = 0 #le compteur de lettres du mot en cours
        n=len(ch)
```

```
            for k in range(n):
                    if ch[k]!='␣':
                            cpt+=1
                    else:
                            if cpt in D:
                                    D[cpt]+=1
                            else:
                                    D[cpt]=1
                            cpt=0
            return D
```

Exercice 6

```
#On vérifie que chaque image est présente une et une seule fois
#D[cle] est l'image de cle
#on contruit le dictionnaire reciproque D[cle] -> cle

def est_bijectif(D):
        recip = {}   #le dictionnaire de la reciproque
        for cle in D:
                if D[cle] in recip: #si l'image est deja vue
                        return False#l'application n'est pas bijective
                else:
                        recip[D[cle]] = cle  #sinon on associe à D[cle] son antecedent cle
        return recip
```