```
#### Epreuve informatique X Cachan ESPCI 2011 ###
                #Partie I
#Remarques.
#L'énoncé de 2011 est adapté à Maple ou Mathematica dont le type liste commence à l'indice 1.
#Pour l'adapter à python il suffit juste de considérer les permutations de l'ensemble {0,....,n-1} ce que je fais.
#L'énoncé propose une fonction taille(t). Elle se traduit en python par len(t)
#L'énoncé propose allouer(n) pour créer un tableau "vide".
#On peut, par exemple, initialiser une liste de 0 par [0 for i in range(n)]. Convient pour tout le sujet.



            #Question 1
#Je teste grâce à deux boucles for, si pour tout i<j, f(i) <> f(j).
#Ainsi f est injective et donc bijective car de E_n  vers lui même.

def estPermutation(t):
    n=len(t)
    reponse=True
    for i in range(n):
        for j in range(i+1,n):
            if t[i]==t[j]:
                reponse=False
    return reponse

essai=[1,4,0,2,3]
print('essai=',essai)



            #Question 2
#Elementaire. Je créé un tableau de zéros au départ (cf allouer(n)) puis je le remplis à l'aide d'une boucle for des valeurs des compositions traduites par t[u[i]].

def composer(t,u):
    n=len(t)
    t_comp=[0 for i in range(n)]
    for i in range(n):
        t_comp[i]=t[u[i]]
    return(t_comp)



            #Question 3
#Elementaire. Je créé un tableau de zéros au départ (cf allouer(n)) puis je le remplis à l'aide d'une boucle for grâce à t[i]=k ssi t^(-1)[k]=i, soit t^(-1)[t[i]]=i

def inverser(t):
    n=len(t)
    t_inv=[0 for i in range(n)]
    for i in range(n):
        t_inv[t[i]]=i
    return(t_inv)



            #Question 4.
#t^1=id donne forcément t = id.
#Pour une d'ordre n, c'est un cycle sur les n valeurs, soit t=[1,2,3,.....,n-1,0]



            #Question 5
#Elementaire. Je construis le tableau id de l'identité grâce à [i, for i in rang
```



```
e(n)].
#Puis avec k initialisé à 1, avec une boucle while, tant que t^k<>id, k=k+1
#Remarque : Ici j'utilise le test d'égalité de deux listes qui signifie l'égalité de leur contenu(et non des références des variables associées)

def ordre(t):
    n=len(t)
    id=[i for i in range(n)]
    k=1
    tk=t
    while tk!=id:
        tk=composer(tk,t)
        k=k+1
    return(k)


                        ####Partie II###



            #Question 6
#Elementaire. k initialisé à 1. Boucle while tant que t^k[i]<>i  incrémente k de + 1 et calcule le nouvel itere.

def periode(t,i):
    n=len(t)
    k=1
    tki=t[i]
    while tki!=i:
        tki=t[tki]
        k=k+1
    return(k)



            #Question 7
#Je constate que si i est donné le tableau [i,t[i],.....,t^(n-1)[i]] donne toutes les valeurs orbitales de i quitte à la répéter.
#Python accepte la commande logique "une valeur in "une liste" que je me permets d'utiliser.
#Ceci donne une procédure simple à écrire.

def estDansOrbite(t,i,j):
    n=len(t)
    #creation du tableau des valeurs orbitales de i
    t_orbite_i=[0 for i in range(n)]
    t_orbite_i[0]=i
    tki=i
    for l in range(1,n):
        tki=t[tki]
        t_orbite_i[l]=tki
    # test de j est dans l' orbite de i
    return(j in t_orbite_i)



            #Question 8
#Une façon "efficace" de caractériser une une transposition est de compter ses points non invariants.
#Si ils sont au nombre de 2 c'est une transposition.

def estTransposition(t):
    n=len(t)
    cpt=0
    for i in range(n):
        if t[i]!=i:
            cpt=cpt+1
    if cpt==2:
        return(True)
```

```
    else:
        return(False)

Trans=[0,1,4,3,2,5]
print('Trans=',Trans)



            #Question 9
#Une façon de caractériser un cycle. On compte les non invariants (donc les inva
riants par complémentaire).
#Soit cpt le cardinal des non invariants. Je prends une variable nommée "variant
" qui va contenir un non invariant quelconque
#dans mon algorithme ce sera le dernier repéré car écrasant à chaque fois le pré
cédent.
#Si la periode de variant est cpt on a un cycle.

def estCycle(t):
    n=len(t)
    cpt=0
    for i in range(n):
        if t[i]!=i:
            cpt=cpt+1
            variant=i
    if periode(t,variant)==cpt:
        return(True)
    else:
        return(False)


Cycle=[0,1,4,3,6,5,2,7]
print('Cycle=',Cycle)



                        ####Partie III####

            #Question 10
#Une méthode élémentaire qui ne répond pas à la question est d'appliquer la fonc
tion periode(t,i) à tous les éléments du tableau t.
#En effet cette méthode peut être en temps quadratique car chaque appel de perio
de(t,i) peut engendrer (au maximum) n calculs des itérés de i par t.
#Ce sera par exemple le cas pour un cycle de longueur n la taille de t.

def Q_periodes(t):
    n=len(t)
    t_p=[0 for i in range(n)]
    for k in range(n):
        t_p[k]=periode(t,k)
    return(t_p)


#Une deuxième méthode en temps linéaire, donc répondant à la question.
#Il suffit de remarquer que la periode d un élément correspond au cardinal de so
n orbite, mais aussi à la periode de tous les éléments de son orbite.
#Enfin les orbites étant disjointes la somme de leur cardinal vaut n.
#On part alors d'un tableau noté t_p de periodes initialisé toutes à nulles. On
parcourt ce tableau. Chaque fois que t_p[k]=0  donc que l'on a pas calculé la pe
riode de k :
#On calcule dans un premier temps la periode de k notée p_k.
#Dans un deuxième temps on remplit tous les t_p[j] à p_k pour les éléments j de
l'orbite de k. J'utilise pour cela la variable tjk qui me sert à calculer tous l
es itérés de k par t.
#Ces deux opérations vont alors effectuer n calculs d'itérés de t, n remplissage
s du tableau t_p ce qui assure un temps linéaire.

def periodes(t):
    n=len(t)
    t_p=[0 for i in range(n)]
```



```
    for k in range(n):
        if t_p[k]==0:
            p_k=periode(t,k)
            t_p[k]=p_k
            tjk=k
            for j in range(1,p_k):
                tjk=k
                t_p[tjk]=p_k
    return(t_p)



            #Question 11
#L'énoncé est ici très précis. Pour chaque i  dans {1,...n} on utilise p[i] sa p
eriode pour t,
# et on calcule efficacement t^k(i) = t^r(i) avec r le reste de la division eucl
idienne de k par p[i].
def itererEfficace(t,k):
    n=len(t)
    p=periodes(t)
    t_k=[0 for i in range(n)]
    for i in range(n):
        r=k%p[i]  # Operateur modulo noté % en python
        tki=i
        for j in range(r):
            tki=t[tki]
        t_k[i]=tki
    return(t_k)



            #Question 12
#Prendre n = 5. un cycle de periode 3 et une permutation de periode 2 donne une
priode globale à 6=2*3 = ppcm(2,3) dépassant 5.

            #Question 13.
#Un algorithme classique, ne veut pas dire trivial car il faut faire attention a
ux différentes affectations.
#L'énoncé ne précise pas l'arrêt de l'algorithme d'Euclide. Tant que r<>0 on con
tinue le procédé.
# Quand r = 0, pgcd(b,0) = b.
def pgcd(a,b):
    while(b!=0):
        r=a%b
        a=b
        b=r
    return(a)
# On peut remarquer que cette fonction modifie les paramètres d'entrée chose int
erdite en Maple cf orgine du sujet.



            #Question 14. Sans commentaire.
def ppcm(a,b):
    return((a*b)//pgcd(a,b))



            #Question 15.
#La phrase "minimiser le nombre d'appels" est peu cadrée. Je choisis ici une amé
lioration sans doute pas optimale.
#Tout d'abord j'utilise ppcm(a,b,c) = ppcm(ppcm(a,b),c). ensuite soit  ppcm(p[1]
,....,p[i - 1])  celui des i-1 premiers,
#Je teste si p[i] divise celui ci dans ce cas il reste inchangé sinon je calcule
 le nouveau ppcm.
def ordreEfficace(t):
    n=len(t)
    p=periodes(t)
    ordre=1
    for i in range(n):
        if ordre%p[i]!=0:
```

```
            ordre=ppcm(ordre,p[i])
    return(ordre)

#> ordreEfficace:=proc(t)
#> local n,p,ordre,i;
#> n:=nops(t);
#> p:=L_periodes(t);
#> ordre:=1:
#> for i from 1 to n do
#>    if irem(ordre,p[i])<>0 then ordre:=ppcm(ordre,p[i]) fi:
#> od:
#> ordre;
#> end;

#> ordreEfficace(Cycle);
```