Conseils :

- Prenez le temps de bien faire les choses et rendez une copie propre. Des points seront mis pour le soin (environ 1,5 points sur 20). Le fait de commenter ses fonctions pour les expliquer sera aussi évalué.
- Aucun ordre n'est imposé pour la résolution, par contre, rendez les problèmes dans l'ordre. Numérotez avec rigueur les questions que vous traitez.
- Même si vous n'avez pas sû faire une question, vous pouvez utiliser la fonction que vous étiez censé coder pour répondre aux questions suivantes.
- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- L'usage des **calculatrices est interdit**.

L'usage de fonctions de python non vues en cours telles que min/max/sum n'est pas autorisé sauf mention contraire.

I. À PROPOS DE GRAPHES

Soit un graphe $G = (S,A)$ tel que `S = {0, 1, 2, 3}` et `A = {{0,0},{1,0},{0,2},{3,1},{2,1}}`

Q1 ➢ Dessiner une représentation possible de ce graphe.

Q2 ➢ Écrire une liste d'adjacence pouvant représenter ce graphe.

Q3 ➢ Écrire la matrice d'adjacence représentant ce graphe.

Q4 ➢ Ce graphe est-il connexe ? Que signifie ce terme ?

Q5 En justifiant votre réponse, dire si la matrice suivante $\begin{pmatrix} 2 & \infty & 4 \\ \infty & 4 & 2 \end{pmatrix}$:

1. peut représenter un graphe orienté ;
2. peut représenter un graphe non orienté ;
3. ne peut pas représenter un graphe ;
4. on ne peut pas savoir ;
5. peut représenter un graphe pondéré ;
6. peut représenter un graphe non pondéré.

Q6 Écrire une fonction `matrice_vers_liste(matrice:list)->list` qui prend en argument une liste de liste représentant la matrice d'adjacence d'un graphe non pondéré et qui renvoie une représentation en terme de liste d'adjacence. A-t-on besoin de savoir si le graphe est orienté avant d'appliquer votre fonction ?

II. QUELQUES PETITES FONCTIONS

Q7 ➢ Écrire une fonction `mini(L)` qui prend en argument une liste de nombre et qui renvoie sous forme d'un tuple l'indice [1] du minimum et la valeur du minimum. Par exemple mini ([0, - 2,3]) doit renvoyer 1, -2

Q8 ➢ Écrire une fonction `fact(n)` récursive qui prend comme argument un nombre entier positif ou nul n et renvoie la valeur de $n!$.

Q9 ➢ Écrire une fonction `eleve_carre` qui prend comme argument une liste de nombres et qui renvoie une liste contentant les carrés des éléments de la première liste.
Par exemple `eleve_carre([-2,3])` renvoie `[4,9]`.

Q10 ➢ Écrire une fonction `produit_liste` qui prend comme argument deux listes de nombres que l'on supposera de même longueur et qui renvoie une liste contenant le produit terme à terme.
Par exemple `produit_liste([-2,3],[1,4])` renvoie `[-2,12]`.

Q11 ➢ Écrire une fonction `dicho(L,x)` qui prend comme argument une liste de nombre L supposée triée et un nombre x et qui renvoie $True$ si $x \in L$ et $False$ sinon. On utilisera une une recherche dichotomique. Expliquer précisément les points délicats de l'algorithme.

Q12 ➢ Écrire une fonction `faitPartie(ch1,ch2)` qui prend comme argument deux chaines de caractères et qui renvoie $True$ si $ch1$ peut se retrouver dans $ch2$ et $False$ sinon. Par exemple `faitPartie("jor","bonjour")->False` alors que `faitPartie("jour","bonjour")->True`.
Il n'est bien entendu pas autorisé d'utiliser le test "in" de python mais seulement l'égalité entre deux lettres. (On pourra utiliser le "in" de "for i in range" bien sûr). Il est conseillé de coder une fonction annexe qui teste l'égalité entre deux chaines de même longueur.

1. on ne se préoccupera pas du cas où le minimum apparait plusieurs fois.

III. Voyageur de commerce, le retour

On pourra importer la fonction exp (exponentielle) du module math ainsi que les fonctions random et randint du module random dont la documentation indique :
`random()-> Return random float in the interval [0, 1).`
`randint()-> Return random integer in range [a, b], including both end points.`

A. Introduction

Dans le devoir précédent, vous avez abordé le problème du voyageur de commerce. On se propose dans ce problème d'étudier deux stratégies alternatives à la stratégie gloutonne étudiée dans le devoir précédent. Il est à noter qu'aucune connaissance préalable n'est nécessaire (bon, par contre, rassurez vous, comme vous avez travaillé le corrigé du précédent devoir, cela va vous aider à comprendre le problème plus rapidement).

Dans ce problème, on considère n villes, numérotée de 0 à $n-1$. On cherche le cycle passant par toutes les villes de distance totale minimale, c'est-à-dire un chemin qui part d'une ville, passe par toutes les autres villes, puis revient à la première ville, tout en minimisant la distance totale parcourue. On supposera que la distance entre chaque couple de villes est enregistrée dans une liste de listes (matrice) d, variable globale, telle que $d[i][j]$ est la distance depuis la ville i vers la ville j. Dans ce problème, pour tenir compte d'éventuels sens uniques, la matrice d ne sera pas nécessairement symétrique.

Puisque l'on cherche un cycle, on peut imposer le point de départ et d'arrivée sans perte de généralité ([0,1,2,0] est le même cycle que [1,2,0,1] et a donc la même distance). Ainsi on décide arbitrairement de partir de la ville 0. Un chemin valide sera donc une liste d'entiers entre 0 et $n-1$, telle que le premier et le dernier élément de la liste sont 0, et les autres entiers de $[\![1, n-1]\!]$ apparaissent exactement une fois.

Q13 ➢ Écrire une fonction `est_valide(chemin:list,n:int)->bool` qui renvoie *True* si le chemin est valide et *False* sinon. On prendra soin d'écrire un algorithme en temps au plus quadratique avec n et on justifiera la complexité du programme brièvement.

Q14 ➢ Écrire une fonction `longueur(c:list, d:list[list[float]])->float` qui prend en argument un chemin c (supposé valide) et la matrice d, et qui renvoie en temps linéaire le coût du chemin, c'est-à-dire la distance totale parcourue. Il n'est pas nécessaire de justifier la complexité de votre programme.
Exemple : si `d = [[0,2,4],[2,0,3],[4,3,0]]`, alors `longueur([0,1,2,0],d)-> 9`.

B. Nous sommes des brutes !

On souhaite employer une méthode force brute, c'est-à-dire tester toutes les possibilités et renvoyer la meilleure. Pour cela, on souhaite écrire préalablement une fonction qui renvoie toutes les permutations possibles d'une liste d'éléments (supposés distinct deux à deux).

Q15 ➢ Écrire une fonction `perm(L:list)->list[list]` qui renvoie la liste de toutes les permutations de L. On ne se préoccupera pas de complexité et peu importe l'ordre dans lequel les permutations sont renvoyées. Il est possible d'utiliser la récursivité. Expliquer qualitativement pourquoi votre fonction réalise bien l'opération demandée.
Exemple : `perm([1, 2, 3])->[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]`

Q16 ➢ Écrire une fonction `brute(d:list[list[float]])->list` qui, étant donné une matrice d, renvoie un meilleur chemin (si plusieurs chemins ont la même longueur, peu importe celui que vous renvoyez). Par exemple : si `d = [[0,2,4,3],[2,0,3,1],[4,3,0,2],[7,2,4,0]]` alors `brute(d)-> [0, 2, 3, 1, 0]`

C. Nous sommes des physiciens !

Nous allons dans ce paragraphe utiliser une idée venant de la physique statistique (facteur de Boltzmann) : l'algorithme du recuit simulé.

On définit une température T initiale (voir utilité plus loin), on part d'un chemin arbitraire (par exemple $[0,1,...,n-1,0]$) puis on fait une modification aléatoire de ce chemin pour obtenir un deuxième chemin. Si ce nouveau chemin est plus court que le précédent, on le considère comme notre chemin actuel, on parlera d'«accepter» le nouveau chemin. Si le chemin est plus long, on ne le rejette pas nécessairement, on l'«accepte» avec une probabilité qui est d'autant plus faible que la différence de longueur est importante. La probabilité d'«accepter» le nouveau chemin est $p = \exp\left(\frac{ca-cn}{T}\right)$ où ca est le coût (c'est-à-dire la longueur) du chemin actuel et cn est le coût du nouveau chemin. On diminue ensuite la température T d'un facteur α et on recommence le processus jusqu'à ce que la température soit assez faible.

La difficulté est le choix de la modification aléatoire pour que l'algorithme soit performant. Ici, un bon choix est d'inverser l'ordre dans lequel on parcourt les villes entre un indice i et un indice j choisis aléatoirement et tels que $0 < i < j < n$.

Q17 ➢ Écrire une fonction `permut(c:list, i:int, j:int)->list` qui prend en argument un chemin c et des indices i et j et qui renvoie un chemin qui contient les mêmes éléments que c, dans le même ordre, sauf entre i et j (inclus) où l'ordre est inversé. Attention à ne pas modifier la liste c. Exemple : `permut([0, 1, 2, 3, 4, 5, 0], 2, 5))-> [0, 1, 5, 4, 3, 2, 0]`

Q18 ➢ Écrire une fonction `alea(p:float)->bool` qui prend en argument un flottant compris entre 0 et 1 et qui renvoie aléatoirement $True$ ou $False$: $True$ avec une probabilité p et $False$ avec une probabilité $1-p$.

Q19 ➢ Écrire une fonction `recuit(alpha:float, Tini:float, Tfin:float, d)` qui étant donné une température initiale $Tini$ et finale $Tfin$ et une matrice d, renvoie un chemin obtenu par l'algorithme du recuit simulé (c'est-à-dire l'algorithme décrit ci-dessus). Le paramètre *alpha* est un réel dans]0,1[par lequel on multiplie la température à chaque itération pour la faire diminuer.

D. Nous sommes des biologistes ...

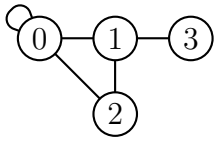
Pour votre culture générale, d'autres algorithmes appelés «colonies de fourmis» se basent plutôt sur la manière dont les fourmis trouvent des chemis optimum de façon collective et ce alors même qu'une fourmi seule se déplace un peu au hasard. Chaque fourmi dépose des substances odorantes volatiles lors de son déplacement, et lorsqu'elle choisit un chemin, le fait en partie en fonction des substance odorantes déjà présentes. Sur les chemins les plus longs, l'odeur s'atténue plus vite et donc au fur et à mesure les chemins les plus courts sont renforcés. Si vous vous ennuyez, vous pouvez réfléchir à la manière d'implémenter cette idée dans le problème qui nous intéresse ici.

I. À PROPOS DE GRAPHES

Soit un graphe $G = (S,A)$ tel que `S = {0, 1, 2, 3}` et `A = {{0,0},{1,0},{0,2},{3,1},{2,1}}`

Q1 ➢ Dessiner une représentation possible de ce graphe.

Attention, le graphe n'était pas orienté. On peut le voir car A est un ensemble d'ensemble (non ordonné) et non un ensemble de couple qui aurait été écrit `A = {(0,0),(1,0),...}`



Q2 ➢ Écrire une liste d'adjacence pouvant représenter ce graphe.

```
L = [[0,1,2],#voisins de 0
     [0,2,3],#voisins de 1
     [0,1],  #voisins de 2
     [1]    ]#voisins de 3
```

Q3 ➢ Écrire la matrice d'adjacence représentant ce graphe.

Convention choisie : 0 correspond à non lié, 1 correspond à lié (ce n'est pas la seule convention possible, on pourrait le faire avec des booléens ou ∞).

```
M = [[1,1,1,0],#voisins de 0
     [1,0,1,1],#voisins de 1
     [1,1,0,0],#voisins de 1
     [0,1,0,0]]#voisins de 3
```

Q4 ➢ Ce graphe est connexe car depuis n'importe quel sommet on peut accéder (en suivant des arêtes) à n'importe quel autre sommet.
Remarque : si vous avez considéré un graphe orienté, nous n'avons pas défini la notion de connexité. On parle de graphe fortement connexe si pour tout couple de sommet ($s1$,$s2$) on peut aller de $s1$ à $s2$ et de $s2$ à $s1$.

Q5 La matrice suivante $\begin{pmatrix} 2 & \infty & 4 \\ \infty & 4 & 2 \end{pmatrix}$ ne peut pas représenter un graphe avec les conventions du cours car ce n'est pas une matrice carrée. En effet, $M_{i,j}$ doit représenter le fait qu'il y a un lien/coût entre les sommets i et j, or i et j varie dans le même intervalle d'entiers et la matrice doit donc être carrée. Si cette matrice représentait un graphe, ce serait forcément pondéré car on voit des valeurs différentes (plus que 2, donc pas seulement «relié ou pas relié»). Pour que ce soit un graphe non orienté, il faudrait que la matrice (carré !) soit symétrique.

Q6 Écrire une fonction `matrice_vers_liste(matrice:list)->list` qui prend en argument une liste de liste représentant la matrice d'adjacence d'un graphe et qui renvoie une représentation en terme de liste d'adjacence. Fonction vue à la fin du cours.

```
def matrice_vers_liste(matrice:list[list])->list:
    n = len(matrice)
    liste = [[] for _ in range(n)] #crée une liste de n listes vides
       indépendantes ! attention à ne pas faire []*n qui crée UNE?
       liste vide (identique à [] tout court)
    for i in range(n):#ligne
        for j in range(n):#colonne
            if matrice[i][j] != 0:#ou False
                liste[i].append(j) #j est successeur de i, on le met
                   dans liste[i]
    return liste #fonctionne que le graphe soit orienté ou non
```

II. Quelques petites fonctions

Q7 ➢ Fonction vue en TP à plusieurs reprises. À savoir faire sans hésitation. On trouve un minimum local, on avance et on le met à jour si on se rend compte qu'il y a un nombre plus petit.

```
def mini(L):
  imin, m = 0, L[0] #indice,valeur
  for i in range(len(L)):
    if L[i] < m :
      imin, m = i, L[i]
  return imin, m
```

Q8 ➢ 2 points important : gérer les cas de base ($n = 0$ ici) et vérifier que les appels récursifs nous en rapprochent. Il était bien demandé une fonction récursive et c'était souligné.

```
def fact(n):
  if n == 0:
    return 1
  return n*fact(n-1)
```

Q9 ➢ Il faut savoir manipuler les listes. Plusieurs possibilités.

```
def eleve_carre(L):
  res = []
  for x in L:
    res.append(x**2)
  return res
```

```
def eleve_carre(L):
  res = [0] * len(L)
  for i in range(len(L)):
    res[i] = L[i]**2
  return res
def eleve_carre(L):
  return [x**2 for x in L]
```

Q10 ➢ Il faut savoir manipuler les listes. Plusieurs possibilités à nouveau. Par contre il est important ici d'avoir un seul indice pour parcourir les deux listes «en même temps», il n'y a donc qu'une seule boucle.
Peut se faire avec un append bien sûr.

```
def produit_liste(L1,L2):
  return [L1[i]*L2[i] for i in
     range(len(L1))]
def produit_liste(L1,L2):
  res = [0] * len(L1)
  for i in range(len(L1)):
    res[i] = L1[i]*L2[i]
  return res
```

➢ Fonction très importante avec plusieurs difficultés éventuelles. Vu en TP et en cours.

deb et *fin* représentent des indices, il faut bien faire $deb = 0$ et non $L[0]$. On continue tant qu'il reste au moins un élément dans la liste.

On prend l'indice au milieu. Ce doit être un entier d'où le //.

On compare la valeur $L[m]$ avec x (et non m avec x ou $L[x]$ qui n'a pas de sens car x n'est pas forcément un entier et qu'il ne représente pas un indice mais une valeur).

Si $L[m]$ n'est pas x, alors il ne sert à rien de garder m dans l'intervalle d'indice où l'on cherche x, d'où les $+1$ et -1. Ce n'est pas seulement inutile, cela peut poser des problèmes de terminaison de l'algorithme si on ne le fait pas. Par exemple si $L = [2,5]$ et que l'on cherche $x = 5$, alors $m = 0$ et on doit chercher à «droite» car $L[m] < x$, mais si on fait $deb = m$, on n'a en fait rien changé et on a une boucle infini. On peut aussi s'en rendre compte avec la démonstration de la terminaison faite en cours.

Enfin, si on sort de la boucle sans avoir renvoyé vrai, alors c'est que $x \notin L$.

Q11

```python
def dicho(L,x):
    deb = 0 #indice de début
    fin = len(L)-1 #et fin
    while deb <= fin:
        m = (deb+fin)//2#//2 important ici
        if L[m] == x:
            return True
        elif L[m]<x: #il faut chercher à droite de m
            deb = m+1#strictement à droite
        else:#dans ce cas il faut chercher à gauche
            fin = m-1
    return False
```

```python
def dicho(L,x):
    deb = 0
    fin = len(L)
    while deb < fin:
        m = (deb+fin)//2#//2 important ici
        if L[m] == x:
            return True
        elif L[m]<x: #il faut chercher à droite de m
            deb = m+1#strictement à droite
        else:#dans ce cas il faut chercher à gauche
            fin = m
    return False
```

Question : l'algorithme ci-dessus à droite présente 3 différences avec celui de gauche. Quelles sont-elles ? Pourquoi est-il lui aussi correcte ? (Quelle convention est prise pour fin dans les deux algorithmes précédents).

➢ Fonction vue en TP et on cours. Attention, le for du 2e range est délicat et mérite une justification détaillée. Dans la syntaxe $ch2[a:b]$, le dernier b pertinent est $b_{max} = len(ch2) = p$.

Q12 Cela revient à prendre $ch2$ jusqu'à l'indice $p-1$ inclus. On veut donc $decal_{max} + m = p$, soit $decal_{max} = p - m$. Toutefois, la convention de range est que le dernier est exclus, d'où le $+1$ dans le range.

```python
def faitPartie(ch1, ch2):
    def egal(c1,c2):
        #suppose len(c1) == len(c2)
        for i in range(len(c1)):
            if c1[i] != c2[i]:
                return False
        return True
    m = len(ch1)#longueur du "mot"
    p = len(ch2)#longueur de la "phrase"
    for decal in range(p-m+1):
        if egal(ch1,ch2[decal:decal+m]):
            return True
    return False
#ou sans fonction intermédiaire
def faitPartie2(ch1, ch2):
    m = len(ch1)#longueur du "mot"
    p = len(ch2)#longueur de la "phrase"
    for decal in range(p-m+1):
        test = True #vaut vrai si ch1[:i]==ch2[decal:decal+i]
        for i in range(m):
            if ch1[i] != ch2[decal + i]:
                test = False #faux car une lettre de différence
                break #break non indispensable
        if test:#si test vaut vrai c'est que ch1[:i]==ch2[decal:
```

```
            decal+i] encore ici, c'est à dire même pour imax
25              return True#donc les chaines sont égales
26      return False#on a jamais trouvé d'égalité
```

III. Voyageur de commerce, le retour

A. Introduction

Q13 ➢ Fonction `est_valide(chemin:list,n:int)->bool` qui renvoie $True$ si le chemin est valide et $False$ sinon.
Remarque : je ne vérifie pas que le contenu de chemin est bien des entiers et qu'ils sont entre 0 et $n-1$ directement, c'est fait indirectement car je vérifie que les entiers entre 1 et n sont présents et qu'il n'y a rien d'autre. Mais en fonction de comment on code cela, il faudrait potentiellement le faire.
Ici, pour voir s'il y a bien toutes les villes une et une seule fois, je vérifie que toutes les villes sont présentes au moins une fois, puis la vérification sur la taille de chemin m'assure qu'il n'y a pas de doublon (s'il y avait au moins un doublon, alors la taille serait trop grande puisqu'à côté de ça il n'en manque pas).

```
1  def est_valide(chemin, n):
2      if chemin[0]!=0 or chemin[-1] != 0: #pas juste chemin[0] !=
           chemin[-1]
3          return False #départ ou arrivée invalide
4      if len(chemin) != n+1:
5          return False #nombre de ville invalide
6      #vu la condition ci-dessus, juste besoin de tester si chaque
           ville de [1,n-1] est présente au moins une fois.
7      #Si c'est le cas pour toutes les villes, vu la taille de la
           liste et les bords, elle sera présente exactement 1 fois
8      def cherche(x,L):
9          """cherche si la valeur x est présente dans la liste L,
               codé nous même pour controler la complexité"""
10         for val in L:
11             if val == x:
12                 return True
13         return False
14     for ville in range(1,n):
15         if not cherche(ville,chemin):
16             return False
17     return True
```

Complexité : `cherche` est de complexité $\mathcal{O}(len(L))$ car on itère sur tous les éléments de L pour faire une opération élémentaire (test d'égalité entre deux nombres). Les 4 premières lignes se font en temps constant. La boucle `for ville in range(1,n)` s'exécute $n-1$ fois et utilise la fonction cherche de complexité $\mathcal{O}(n)$, ainsi la boucle est de complexité $(n-1)\times\mathcal{O}(n)=\mathcal{O}(n^2)$.
Remarque : il est en fait possible de faire un algorithme en temps linéaire, saurez-vous trouver comment ?!?
Remarque 2 : vous avez été nombreux à tester s'il n'y avait pas de doublon en faisant 2 boucles. Cela a souvent été mal fait, il faut bien s'assurer d'éviter de tester si $L[i]==L[j]$ avec $i==j$

sinon on aura l'impression qu'il y a un doublon alors que c'est juste le même nombre. Un moyen simple et élégant de faire cela est de faire la 2e boucle de $i+1$ à n (technique déjà vue en TP). Par exemple ci-dessous (mais il faudrait adapter pour l'utiliser ici).

```
def sans_doublons(L):
    for i in range(len(L)):
        for j in range(i+1,len(L)):
            #on ne teste que strictement à droite
            #inutile de regarder à gauche
            #car on a déjà testé
            if L[i] == L[j]:
                return False #il existe un contre exemple
    return True #jamais trouvé de doublon
```

Q14 ➢ Écrire une fonction `longueur(c:list, d:list[list[float]])->float` qui prend en argument un chemin c (supposé valide) et la matrice d.

```
def longueur(c, d):
    res = 0
    for i in range(len(c)-1): #Attention au -1
        res += d[ c[i] ][ c[i+1] ]#les indices des villes
           successives
    return res
```

B. Nous sommes des brutes !

Q15 ➢ Écrire une fonction `perm(L:list)->list[list]` qui renvoie la liste de toutes les permutations de L. On ne se préoccupera pas de complexité et peu importe l'ordre dans lequel les permutations sont renvoyées. Il est possible d'utiliser la récursivité. Expliquer qualitativement pourquoi votre fonction réalise bien l'opération demandée.

```
def perm(L):
    if len(L) <=1:#cas de base
        return [L] #une seule permutation, la liste elle même
    res = []
    #sinon, plus d'un élément. chaque élément pourrait être en 1
       ère position
    #avec après lui toutes les permutations des éléments restant
    for i in range(len(L)):
        temp = perm(L[:i] + L[i+1:])#toutes les permutations
           pour les éléments
        #d'indice != de i
        for liste in temp:
            res.append([L[i]] + liste)
        #temp contient toutes les permutations avec L[i] en
           premier élément
    return res #on a calculé temp pour tous les i, donc toutes
       les permutations puisque chaque élément peut être en
       première place
```

Q16 ➢ Écrire une fonction `brute(d:list[list[float]])->list` qui, étant donné une matrice d, renvoie un meilleur chemin.

```
def brute(d):
    n = len(d)
    liste_perm = perm(list(range(1,n)))
    c_min = [0] + liste_perm[0] + [0] #chemin le plus court
       jusqu'à présent
    l_min = longueur(c_min, d) #longueur correspondante
    for liste in liste_perm[1:]:
        c = [0] + liste + [0] #il existe des moyens plus
           efficace ici
        long = longueur(c,d)
        if long < l_min:
            l_min = long
            c_min = c
    return c_min
```

C. Nous sommes des physiciens !

Q17 ➢ Écrire une fonction `permut(c:list, i:int, j:int)->list` qui prend en argument un chemin c et des indices i et j et qui renvoie un chemin qui contient les mêmes éléments que c, dans le même ordre, sauf entre i et j (inclus) où l'ordre est inversé. Attention à ne pas modifier la liste c.

```
def permut(c,i,j):
    assert 0<i and i<j and j<len(c)-1 #non demandé par l'énoncé
    return c[:i] + c[j:i-1:-1] + c[j+1:]
```

Q18 ➢ Écrire une fonction `alea(p:float)->bool` qui prend en argument un flottant compris entre 0 et 1 et qui renvoie aléatoirement $True$ ou $False$: $True$ avec une probabilité p et $False$ avec une probabilité $1-p$.

```
def alea(p):
    return random.random()<=p
```

Q19 ➢ Écrire une fonction `recuit(alpha:float, Tini:float, Tfin:float, d)` qui étant donné une température initiale $Tini$ et finale $Tfin$ et une matrice d, renvoie un chemin obtenu par l'algorithme du recuit simulé (c'est-à-dire l'algorithme décrit ci-dessus). Le paramètre *alpha* est un réel dans]0,1[par lequel on multiplie la température à chaque itération pour la faire diminuer.

```
def recuit(alpha, Tini, Tfin, d):
    assert 0 < alpha and alpha < 1
    n = len(d)
    c = list(range(n))
    long = longueur(c, d)
    c.append(0)
    T = Tini
    while T>Tfin:
        i = random.randint(1,n-2) #n-2 car j vaut au max n-1 et i<j
        j = random.randint(i+1,n-1)
```

```
11          c2 = permut(c,i,j)
12          long2 = longueur(c2,d)
13          if long2 < long or alea(m.exp((long-long2)/T)):#utilisation
               de l'évaluation paresseuse des booléens même si dans les
               fait on pourrait juste faire l'alea, vu que ça donnerait
               un nombre plus grand que 1, mais bon, calcule une exp pour
                rien
14              c = c2
15              long = long2
16          T = alpha*T
17      return c
```