

DS n°2

Nom et classe :

I. Échauffement

L'énoncé est à rendre avec la copie. Lorsque cela est demandé, cocher la case de l'affirmation juste (parfois plusieurs réponses correctes).

1. On considère le tableau suivante : $T = [[1,4,3] , [2,8,9] , [3,6,7]]$.
Quelles sont les assertions correctes ?

☐ $T[1] = [2,8,9]$ | ☐ $T[1] = [1,4,3]$ | ☐ $T[1] = 4$ | ☐ $T[1] = 1$
☐ $T[1][1] = 1$ | ☐ $T[1][1] = 4$ | ☐ $T[1][1] = 8$ | ☐ $T[1][1] = 3$

2. Dans le code ci-dessous, souligner le(s) erreur(s) présente(nt) :

```
1 tab = [[0 for j in range(5)] for i in range(5)]
2 for i in range(5):
3     for j in range(5)
4         tab[i][j] = i + j
```

3. On considère le code suivant concernant un tableau 5×5 :

```
1 d = tab[0][0]
2 for i in range(5):
3     for j in range(5):
4         if tab[i][j] < d:
5             d = tab[i][j]
```

Ce code permet

- ☐ de trouver le maximum du tableau
☐ de trouver le minimum du tableau
☐ de compter le nombre de cases
☐ de calculer une distance d

4. Dans une boucle `while`, que se passe-t-il si la condition de la boucle est fausse dès le début ?

- ☐ La boucle s'exécute une fois
☐ La boucle s'exécute jusqu'à ce que la condition devienne vraie
☐ La boucle ne s'exécute jamais
☐ La boucle s'exécute en boucle infinie

5. Quel code permet d'annuler les termes d'une liste L existante avec une boucle `while` ?

☐

```
1 i = 0
2 while i < len(L):
3     L[i] = 0
4     i += 1
```

☐

```
1 i = 0
2 while i < len(L):
3     L.append(0)
4     i += 1
```

☐

```
1 i = 0
2 while i < len(L) - 1:
3     L[i] = 0
4     i += 1
```

☐

```
1 i = 0
2 while i > len(L):
3     L.append(0)
4     i += 1
```

6. Quelle est la valeur finale de i lors de l'exécution du code suivant :

```
1 i = 0
2 while i < 4:
3     i += 1
```

Votre réponse :

7. La recherche de l'avant dernier terme dans une liste triée de 8 éléments nécessite

- ☐ 3 opérations par dichotomie
☐ 7 opérations par parcours itératif usuel
☐ 8 opérations par dichotomie
☐ 3 opérations par parcours itératif usuel

8. Compléter la fonction suivante permettant de faire une recherche par dichotomie de l'élément x dans une liste L :

```
1 def recherche_dicho(x,L)
2 .
3 .
4 .
5 .
6 .
7 .
8 .
9 .
10 .
11 .
12 .
13 .
14 .
15 .
16 .
17 .
```

II. Pays aléatoire

1 - Génération d'un pays

On désire simuler la génération d'un pays avec des villes situées aléatoirement sur un tableau T de taille $n \times n$. Les villes seront numérotées de 1 à N et placées aléatoirement sur le tableau.

□ **Q 1** - Définir une fonction `gen_pays(n)` renvoyant un tableau 2D de taille $n \times n$ entièrement nul.

On notera T le résultat de la fonction `gen_pays(n)`.

On rappelle que la fonction `randint(start,stop)` du module `random` fournit une variable aléatoire entière comprise entre `start` et `stop` inclus.

□ **Q 2** - Rappeler la ligne de commande permettant d'importer cette fonction.

□ **Q 3** - Proposer une fonction `coord_alea(T)` qui renvoie les coordonnées aléatoires d'une ville sous la forme d'un tuple (i,j) . La ville doit pouvoir s'insérer dans un tableau T .

□ **Q 4** - Proposer une fonction `gen_villes(T,N)` modifiant sur place le tableau T en insérant de façon aléatoire N villes numérotées de 1 à N . Il faudra veiller à ce que les N villes soient présentes, c'est à dire qu'une ville ne doit pas écraser une ville de numéro inférieur.

Pour un tableau de taille 5×5 , on attendrait le résultat suivant :

```
1 >>> gen_villes(T,5)
2 >>> T
3 [[0,0,2,0,0] ,
4  [0,0,0,1,0] ,
5  [3,0,0,0,5] ,
6  [0,0,0,0,0] ,
7  [0,4,0,0,0] ]
```

□ **Q 5** - Que doit vérifier N par rapport à n pour que l'algorithme précédent se termine toujours ?

2 - Caractéristiques du pays aléatoire

□ **Q 6** - Proposer une fonction `coord_ville(T,k)` qui renvoie les coordonnées de la ville numéro k dans le tableau T .



□ **Q 7** - Définir une fonction `distance(T,k1,k2)` qui renvoie la distance (en unité arbitraire) entre les villes de numéro $k1$ et $k2$.

□ **Q 8** - Définir une fonction `nbre_villes(T)` retournant le nombre de villes dans le tableau T .

□ **Q 9** - Proposer une fonction `liste_distances(T,k)` qui renvoie la liste des distances entre la ville numéro k et toutes les autres villes. La liste retournée sera composée de la distance accompagnée du numéro de la ville. Par exemple pour un pays de taille 9×9 avec 6 villes, on obtiendrait :

```
1 >>> liste_distance(T,3)
2 [ [8.2,1] , [6.7,2] , [8.1,4] , [1.2,5] , [4.2,6] ]
```

La méthode `.sort()` permet de trier une liste. On suppose que la liste L précédente est triée par ordre de distance croissante :

```
1 >>> L.sort()
2 [ [1.2,5] , [4.2,6] , [6.7,2] , [8.1,4] , [8.2,1] ]
```

□ **Q 10** - Proposer une fonction `villes_voisines(L,d)` permettant de retourner les numéros de villes qui sont à moins d'une distance d de celle étudiée dans la liste L . On utilisera une boucle `for`.

□ **Q 11** - La liste L étant triée, on désire trouver la ville dont la distance est égale à une valeur d_0 . Un algorithme de recherche par dichotomie est beaucoup plus efficace lorsque la liste L est grande. Proposer une fonction `recherche(L,d0)` qui donne un numéro de ville située à distance $d0$ de la ville étudiée par L . La fonction devra utiliser une recherche dichotomique.

Introduction

Ce corrigé propose des solutions détaillées aux questions du DS n°2. Les réponses sont accompagnées d'explications pour permettre une meilleure compréhension des concepts abordés.

III. Partie I : Échauffement

Question 1

On considère le tableau suivant : $T = [[1,4,3],[2,8,9],[3,6,7]]$. Les assertions sont évaluées comme suit :

- $T[1] = [2,8,9]$: Vrai, car $T[1]$ correspond à la deuxième ligne du tableau T .
- $T[1] = [1,4,3]$: Faux, $T[0]$ est la première ligne.
- $T[1] = 4$: Faux, $T[1]$ est une liste.
- $T[1] = 1$: Faux, $T[1]$ est une liste.
- $T[1][1] = 1$: Faux, $T[1][1] = 8$.
- $T[1][1] = 4$: Faux, $T[1][1] = 8$.
- $T[1][1] = 8$: Vrai, l'élément situé dans la deuxième ligne et la deuxième colonne est bien 8.
- $T[1][1] = 3$: Faux, $T[1][1] = 8$.

Question 2

Dans le code suivant, les erreurs sont :

Listing 1 – Code avec erreurs

```
tab = [[0 for j in range(5)] for i in range(5)]
for i in range(5):
    for j in range(5) # Manque le ":" ici
        tab[i][j] = i + j
```

Correction : La ligne 3 doit inclure un `:` après `for j in range(5)`.

Question 3

Le code donné permet :

de trouver le **minimum** du tableau.

Explications : À chaque itération, on compare l'élément courant à la valeur minimale actuelle (d), et si l'élément est plus petit, on met à jour d . Cela correspond bien à une recherche de minimum.

Question 4

Dans une boucle **while**, si la condition est fausse dès le départ :

La boucle **ne s'exécute jamais**.

En effet, la condition d'une boucle **while** doit être vraie pour que le bloc s'exécute.

Question 5

Le code correct permettant d'annuler les termes d'une liste L est :

```
i = 0
while i < len(L):
    L[i] = 0
    i += 1
```

Explications : Ce code parcourt la liste avec un index i et remplace chaque élément par 0.

Question 6

Pour le code :

```
i = 0
while i < 4:
    i += 1
```

La valeur finale de i est 4, car la boucle s'arrête lorsque $i = 4$.

IV. Partie II : Pays aléatoire

Question 1

Définir une fonction `gen_pays(n)` qui génère un tableau $n \times n$ nul :

```
def gen_pays(n):
    return [[0 for _ in range(n)] for _ in range(n)]
```

Explications : Cette fonction utilise une compréhension de liste pour créer un tableau $n \times n$ rempli de zéros.

Question 2

La ligne de commande pour importer la fonction `randint` est :

```
from random import randint
```

Question 3

Proposer une fonction `coord_alea(T)` :

```
def coord_alea(T):
    n = len(T)
    return (randint(0, n-1), randint(0, n-1))
```

Explications : La fonction génère des coordonnées aléatoires (i, j) pour un tableau T de taille $n \times n$, en veillant à respecter les bornes.

Question 4

Proposer une fonction `gen_villes(T, N)` :

```
def gen_villes(T, N):
    n = len(T)
    villes = set()
    while len(villes) < N:
        i, j = coord_alea(T)
        if T[i][j] == 0: # Case vide
            T[i][j] = len(villes) + 1
            villes.append((i, j))
```

Explications : On utilise un ensemble pour s'assurer que les coordonnées des villes soient uniques.

Question 5

Que doit vérifier N par rapport à n pour que l'algorithme se termine toujours ?

Pour garantir que l'algorithme termine, N (le nombre de villes) doit être inférieur ou égal au nombre de cases du tableau, soit $N \leq n^2$. **Explications :** Si $N > n^2$, il n'y aurait pas assez de cases disponibles pour insérer toutes les villes, et l'algorithme resterait bloqué dans la boucle.

Question 6

Proposer une fonction `coord_ville(T, k)` qui renvoie les coordonnées de la ville numéro k dans le tableau T :

```
def coord_ville(T, k):
    for i in range(len(T)):
        for j in range(len(T[i])):
            if T[i][j] == k:
                return (i, j)
    return None
```

Explications : La fonction parcourt toutes les cases du tableau T . Si elle trouve la ville k , elle retourne ses coordonnées (i, j) . Si k n'est pas dans T , la fonction retourne `None`.

Question 7

Définir une fonction `distance(T, k1, k2)` qui renvoie la distance entre les villes k_1 et k_2 :

```
import math
```

```
def distance(T, k1, k2):
    x1, y1 = coord_ville(T, k1)
    x2, y2 = coord_ville(T, k2)
    return ((x1-x2)**2 + (y1-y2)**2)**.5
```

Explications : La distance entre deux villes est calculée à l'aide de la formule de la distance euclidienne :

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

La fonction retourne `None` si l'une des deux villes k_1 ou k_2 n'existe pas.

Question 8

Définir une fonction `nbre_villes(T)` qui retourne le nombre de villes dans le tableau T :

```
def nbre_villes(T):
    count = 0
    for i in range(len(T)):
        for j in range(len(T[i])):
            if T[i][j] != 0: # Case occupee par une ville
                count += 1
    return count
```

Explications : La fonction parcourt toutes les cases du tableau T et incrémente un compteur chaque fois qu'elle rencontre une valeur différente de 0 (indiquant une ville).

Question 9

Proposer une fonction `liste_distances(T, k)` qui renvoie la liste des distances entre la ville k et toutes les autres villes :

```
def liste_distances(T, k):
    distances = []
    for i in range(1, nbre_villes(T) + 1):
        # Parcourt les villes
        if i != k:
            d = distance(T, k, i)
            if d is not None:
                distances.append([d, i])
    return distances
```

Explications : La fonction calcule la distance entre la ville k et toutes les autres villes présentes dans T . Les distances sont stockées dans une liste sous forme de sous-listes $[distance, numero_ville]$.

Question 10

Proposer une fonction `villes_voisines(L, d)` pour retourner les villes situées à une distance inférieure à d :

```
def villes_voisines(L, d):
    voisins = []
    for distance, ville in L:
        if distance < d:
            voisins.append(ville)
    return voisins
```

Explications : La fonction parcourt la liste des distances (triées ou non) et sélectionne les villes pour lesquelles la distance est inférieure à d .

Question 11

Proposer une fonction `recherche(L, d0)` utilisant une recherche dichotomique pour trouver une ville située à une distance d_0 :

```
def recherche(L, d0):
    gauche, droite = 0, len(L) - 1
    while gauche <= droite:
        milieu = (gauche + droite) // 2
        if L[milieu][0] == d0:
            return L[milieu][1] # Retourne le numero de la ville
        elif L[milieu][0] < d0:
            gauche = milieu + 1
        else:
            droite = milieu - 1
    return None
```

Explications : - Cette fonction utilise la recherche dichotomique pour trouver une distance d_0 dans une liste triée L . - Si une ville est trouvée à la distance d_0 , son numéro est retourné. Sinon, la fonction retourne **None**. La recherche dichotomique a une complexité en $O(\log(n))$, ce qui est beaucoup plus rapide que la recherche linéaire pour de grandes listes.