Une attention particulière sera prêtée à la présentation et à la rigueur dans la rédaction.
En particulier on prendra soin de bien marquer les indentations (décalages).
Certains codes ou tableaux doivent être complétés sur l'énoncé, il faut donc le rendre avec la copie.

Exercice 1

Quelques rappels sur la manipulation de chaînes de caractères en Python :

```
>>> chaine ='abcd'
>>> eniahc = chaine[::-1]
>>> eniahc
'dcba'
>>> vide = ''
>>> vide = vide + 'a'
>>> vide
'a'
```

On considère désormais le script ci-dessous :

```
alphabet = 'abcde'
n = len(alphabet)

for lettre in alphabet[::-1]:
    ligne = ''
    for i in range(n):
        ligne = ligne + lettre
    print(ligne)
```

script 1

1. Parmi les trois affichages proposés ci-dessous, quel est celui fourni par ce script ?

Affichage 1	**Affichage 2**	**Affichage 3**
edcba	aaaaa	eeeee
edcba	bbbbb	ddddd
edcba	ccccc	ccccc
edcba	ddddd	bbbbb
edcba	eeeee	aaaaa

2. En permutant deux lignes du script, on peut obtenir un autre des affichages proposés.
Quelles lignes suffit-il de permuter et quelle affichage obtient-on alors ?

Exercice 2

On considère la suite (u_n) définie pour tout entier $n \geqslant 0$ par $\begin{cases} u_0 = 1 \\ u_n = u_{n-1} + 2n + 1 \text{pour tout entier } n \geqslant 1 \end{cases}$.

Compléter la fonction `suite(n)` ci-après pour qu'elle retourne le terme d'indice `n` de la suite (u_n) où `n` est un entier positif.

```
def suite(n):
    u = 1
    for k in range(1, .....):
        u = ..........
    return u
```

script 2

Exercice 3

Quelques rappels sur les tests et expressions booléennes en Python :

```
>>> 2 <= 3 and 3 <= 5
True
>>> 2 <= 7 or 7 <= 5
True
```

On veut écrire une fonction `verification(t, binf, bsup)` qui doit retourner :

- `None` si `binf > bsup;`
- `True` si tous les éléments `e` d'un tableau de nombres `t` vérifient `binf <= e` et `e <= bsup;`
- et `False` sinon.

```
def verification(t, binf, bsup):
    if binf > bsup:
        return None
    for e in t:
        if binf <= e and e <= bsup:
            return True
    return False
```

script 3

On donne ci-dessus la solution proposée par un élève. Elle est incorrecte comme le prouve l'exécution ci-dessous :

```
>>> verification([4,1,1,1],2,5)
True
```

Écrire une fonction `verification(t, binf, bsup)` qui satisfait la spécification ci-dessus.

Exercice 4

On suppose l'existence d'un module `robot` qui permet de déplacer un robot sur une grille avec les commandes suivantes :

- ☞ `haut()` pour demander au robot de se déplacer d'une case vers le haut ;
- ☞ `bas()` pour demander au robot de se déplacer d'une case vers le bas ;
- ☞ `droite()` pour demander au robot de se déplacer d'une case vers la droite ;
- ☞ `gauche()` pour demander au robot de se déplacer d'une case vers la gauche.

```
##############
#            #
# ########## #
# ########## #
# ########## #
# ########## #
# ########## #
# ########## #
# ########## #
# ########## #
# ########## #
#D           #
##############
```

On donne ci-dessus une grille de 14 lignes et 14 colonnes avec des caractères # pour matérialiser des murs et des espaces pour les cases où peut passer le robot. Ces cases forment un chemin.
Compléter le script ci-dessous, pour que le robot fasse 730 fois le tour du chemin représenté ci-dessus, *en tournant dans le sens des aiguilles d'une montre* à partir de la case libre marquée par `D`. On utilisera les fonctions du module `robot` et les structures de contrôle habituelles de Python (boucles, structures conditionnelles ...).

```
1 from robot import *
```

script 3

Exercice 5

Quelques rappels sur la manipulation de listes (ou tableaux) en Python :

```
>>> t = [7,4]
>>> t.insert(1, 5)
>>> t
[7, 5, 4]
>>> t.insert(len(t), 6)
>>> t
[7, 5, 4, 6]
```

1. Compléter le code de la fonction `place(t, n)` qui retourne la place à laquelle on peut insérer un entier `n` dans un tableau d'entiers `t` déjà classé dans l'ordre croissant, pour que le tableau modifié soit encore dans l'ordre croissant.

```
def place(t, n):
    i = 0
    while i < len(t) and t[i] .... n:
        ....
    return i
```

script 4

Voici quelques exemples d'exécution de cette fonction :

```
>>> place([1,2,3], 4)
3
>>> place([1,2,2,3], 2)
1
>>> place([1,2,2,3], 0)
0
```

script 4

2. On considère la fonction `mystere(t)` qui prend en argument un tableau d'entiers `t` et qui retourne un nouveau tableau d'entiers :

```
def mystere(t):
    new = [t[0]]
    for i in range(1, len(t)):
        j = place(new, t[i])
        new.insert(j, t[i])
    return new
```

script 5

a. Pour expliquer ce qu'il se passe lors de l'appel `mystere([6, 2, 5, 4])`, compléter le tableau d'état des variables ci-après (noter les valeurs de `j` et `new` à la fin de chaque tour de boucle) :

i	j	new
Initialisation	Rien	6
1	.....	.....
2	.....	.....
3	.....	.....

b. Si vous deviez écrire la documentation de la fonction `mystere(t)` pour expliquer son rôle, que diriez-vous ?

Exercice 6

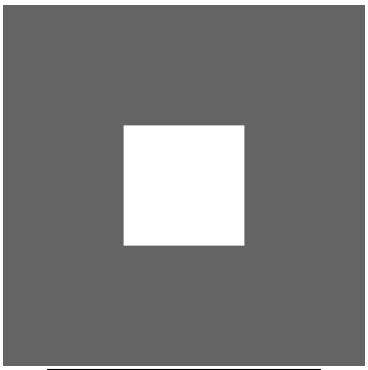
Une institutrice propose un atelier découpage pour ses élèves.

Étape 1 l'élève partage d'abord la feuille en 9 carrés et découpe le carré central ;

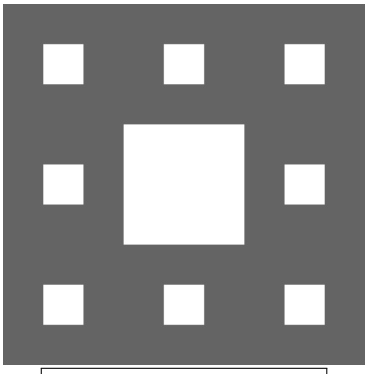
Étape 2 l'élève partage alors les 8 carrés restant en 9 carrés égaux et découpe le carré central ;

Étapes suivantes l'élève répète le même procédé ...

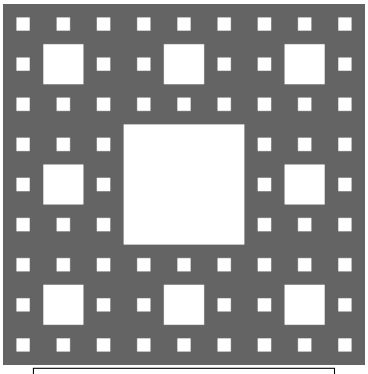
Étape 0 : 0 découpe



Étape 1 : 1 découpe



Étape 2 : 9 découpes



Étape 3 : 73 découpes

Écrire une fonction `nombredecoupe(n)` qui retourne le nombre de découpes au bout de `n` étapes où `n` est un entier positif.