exercice001

Exercice

Cet exercice porte sur l'algorithmique et la programmation en Python. Il aborde les notions de tableaux de tableaux et d'algorithmes de parcours de tableaux.

Partie A : Représentation d'un labyrinthe

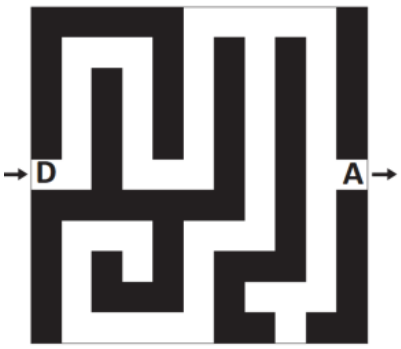
On modélise un labyrinthe par un tableau à deux dimensions à n lignes et m colonnes avec n et m des entiers strictement positifs.

Les lignes sont numérotées de 0 à $n-1$ et les colonnes de 0 à $m-1$. La case en haut à gauche est repérée par $(0,0)$ et la case en bas à droite par $(n-1, m-1)$.

Dans ce tableau :

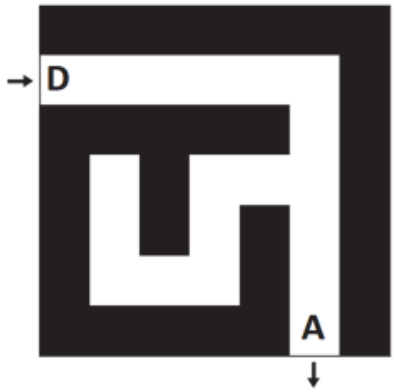
- 0 représente une case vide, hors case de départ et arrivée,
- 1 représente un mur,
- 2 représente le départ du labyrinthe,
- 3 représente l'arrivée du labyrinthe.

Ainsi, en Python, le labyrinthe ci-dessous est représentée par le tableau de tableaux `lab1`.




```
>>> lab1 = [[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1],
... [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
... [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
... [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
... [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
... [2, 0, 1, 0, 0, 0, 1, 0, 1, 0, 3],
... [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
... [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
... [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
... [1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
... [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1]]
```

1. Le labyrinthe ci-dessous est censé être représenté par le tableau de tableaux `lab2`. Cependant, dans ce tableau, un mur se trouve à la place du départ du labyrinthe. Donner une instruction permettant de placer le départ au bon endroit dans `lab2`.




```
>>> lab2 = [[1, 1, 1, 1, 1, 1, 1],
... [2, 0, 0, 0, 0, 0, 1],
... [1, 1, 1, 1, 1, 0, 1],
... [1, 0, 1, 0, 0, 0, 1],
... [1, 0, 1, 0, 1, 0, 1],
... [1, 0, 0, 0, 1, 0, 1],
... [1, 1, 1, 1, 1, 3, 1]]
```

2. Écrire une fonction `est_valide(i, j, n, m)` qui renvoie `True` si le couple `(i, j)` correspond à des coordonnées valides pour un labyrinthe de taille (n, m), et `False` sinon.

   On donne ci-dessous des exemples d'appels.

   ```
   >>> est_valide(5, 2, 10, 10)
   True
   >>> est_valide(-3, 4, 10, 10)
   False
   ```

3. On suppose que le départ d'un labyrinthe est toujours indiqué, mais on ne fait aucune supposition sur son emplacement. Écrire une fonction `depart(lab)` de sorte qu'elle renvoie, sous la forme d'un tuple, les coordonnées du départ d'un labyrinthe (représenté par le paramètre `lab`).

   Par exemple, on doit avoir :

   ```
   >>> depart(lab1)
   (5, 0)
   >>> depart(lab2)
   (1, 0)
   ```

4. Écrire une fonction `nb_cases_vides(lab)` qui renvoie le nombre de cases vides d'un labyrinthe (comprenant donc l'arrivée et le départ).

   Par exemple,

   ```
   >>> nb_cases_vides(lab1)
   58
   >>> nb_cases_vides(lab2)
   19
   ```

Partie B : Recherche d'une solution dans un labyrinthe

On suppose dans cette partie que les labyrinthes possèdent un unique chemin allant du départ à l'arrivée sans repasser par la même case. Dans la suite, c'est ce chemin que l'on appellera solution du labyrinthe.

Pour déterminer la solution d'un labyrinthe, on parcourt les cases vides de proche en proche. Lors d'un tel parcours, afin d'éviter de tourner en rond, on choisit de marquer les cases visitées. Pour cela, on remplace la valeur d'une case visitée dans le tableau représentant le labyrinthe par la valeur 4.

1. On dit que deux cases d'un labyrinthe sont voisines si elles ont un côté commun. Écrire une fonction `voisines(i, j, lab)` qui prend en arguments deux entiers `i` et `j` représentant les coordonnées d'une case et un tableau `lab` qui représente un labyrinthe. Cette fonction renvoie la liste des coordonnées des cases voisines de la case de coordonnées `(i, j)` qui sont valides, non visitées et qui ne sont pas des murs. L'ordre des éléments de cette liste n'importe pas.

   Ainsi,

```
>>> voisines(1, 1, [[1, 1, 1], [4, 0, 0], [1, 0, 1]])
[(2, 1), (1, 2)]
>>> voisines(0, 1, [[4, 0, 1], [1, 0, 0], [1, 0, 1]])
[(1, 1)]
```

Que renvoie l'appel `voisines(1, 2, [[1, 1, 4], [0, 0, 0], [1, 1, 0]])` ?

2. On souhaite stocker la solution dans une liste chemin. Cette liste contiendra les coordonnées des cases de la solution, dans l'ordre. Pour cela, on procède de la façon suivante.

   - Initialement :
     - déterminer les coordonnées du départ : c'est la première case à visiter ;
     - ajouter les coordonnées de la case départ à la liste chemin.
   - Tant que l'arrivée n'a pas été atteinte :
     - on marque la case visitée avec la valeur 4 ;
     - si la case visitée possède une case voisine libre, la première case de la liste renvoyée par la fonction voisines devient la prochaine case à visiter et on ajoute à la liste chemin ;
     - sinon, il s'agit d'une impasse. On supprime alors la dernière case dans la liste chemin. La prochaine case à visiter est celle qui est désormais en dernière position de la liste chemin.

   a. Le tableau de tableaux `lab3` ci-dessous représente un labyrinthe.

```
>>> lab3 = [[1, 1, 1, 1, 1, 1],
...         [2, 0, 0, 0, 0, 3],
...         [1, 0, 1, 0, 1, 1],
...         [1, 1, 1, 0, 0, 1]]
```

La suite d'instructions ci-dessous simule le début des modifications subies par la liste chemin lorsque l'on applique la méthode présentée.

```
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1, 1))
chemin.append((2, 1))
chemin.pop()
chemin.append((1,2))
chemin.append((1,3))
chemin.append((2,3))
```

Compléter cette suite d'instructions jusqu'à ce que la liste chemin représente la solution. Rappel : la méthode pop supprime le dernier élément d'une liste et renvoie cet élément.

b. Écrire une fonction `solution(lab)` donnée ci-dessous de sorte qu'elle renvoie le chemin solution du labyrinthe représenté par le paramètre `lab`. On pourra pour cela utiliser la fonction `voisines`.

Par exemple,

```
>>> solution(lab2)
[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5),
 (4, 5), (5, 5), (6, 5)]
```

exercice002

Exercice

On cherche à rendre une certaine somme entière en utilisant le moins de pièces possibles parmi un système monétaire donné, dont les valeurs sont stockées dans une liste (triée par ordre croissant de valeur). Pour notre système monétaire usuel, les valeurs sont celles de la liste [1, 2, 5, 10, 20, 50, 100]. On peut utiliser plusieurs fois la même valeur.

Le but est d'écrire une fonction nommée `rendu` dont le premier paramètre est un entier positif non nul `somme_a_rendre`, le second une liste des pièces du système monétaire rangée dans l'ordre croissant et qui retourne une liste correspondant aux nombres de pièces ou billets de chaque valeur à rendre.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de plus grande valeur, puis le nombre maximal de billets de valeur juste inférieure au précédent et ainsi de suite jusqu'à la pièce de plus petite valeur.

Exemples :

```
>>> rendu(13, [1, 2, 5])
[1, 1, 2]
>>> rendu(64, [1, 2, 5])
[0, 2, 12]
>>> rendu(89, [1, 2, 5])
[0, 2, 17]
```

Le dernier résultat signifie que pour rendre 89€ avec des pièces de 1€ et 2€ et des billets de 5 à disposition, il faut 17 billets de 5€ et 2 pièces de 2€.

exercice003

Exercice

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self, v, s) :
        self.valeur = v
        self.suivant = s
```

Compléter la classe `File` suivante où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self, element) :
        nouveau_maillon = Maillon(... , self.dernier_file)
        self.dernier_file = ...

    def est_vide(self) :
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...
```

```
    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = ...
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = ...
            maillon.suivant = None
            return resultat
        return None
```

On pourra tester le fonctionnement de la classe en utilisant les commandes suivantes dans la console Python :

```
>>> F = File()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
```

```
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
```

```
>>> F.defile()
5
>>> F.affiche()
7
>>> F.defile()
7
>>> F.affiche()
```

exercice004

Exercice

Cet exercice porte sur les arbres binaires et leurs algorithmes associés

La fédération de badminton souhaite gérer ses compétitions à l'aide d'un logiciel. Pour ce faire, une structure arbre de compétition a été définie récursivement de la façon suivante : un arbre de compétition est soit l'arbre vide, noté ∅, soit un triplet composé d'une chaîne de caractères appelée valeur, d'un arbre de compétition appelé sous-arbre gauche et d'un arbre de compétition appelé sous-arbre droit.

On représente graphiquement un arbre de compétition de la façon suivante :

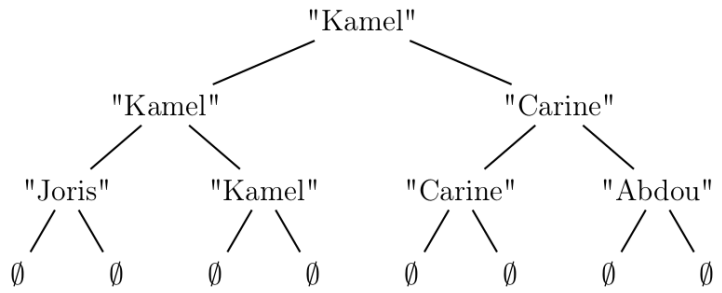


Figure 1: Arbre complet

Pour alléger la représentation d'un arbre de compétition, on ne notera pas les arbres vides, l'arbre précédent sera donc représenté par l'arbre A suivant :

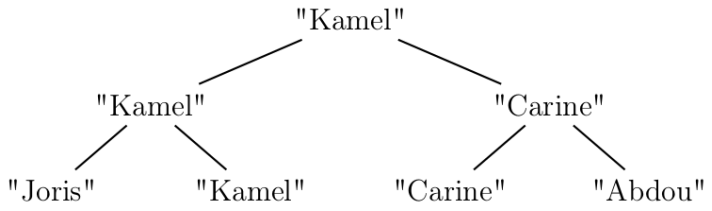


Figure 2: Arbre A

Cet arbre se lit de la façon suivante :

- 4 participants se sont affrontés : Joris, Kamel, Carine et Abdou. Leurs noms apparaissent en bas de l'arbre, ce sont les valeurs de feuilles de l'arbre.
- Au premier tour, Kamel a battu Joris et Carine a battu Abdou.
- En finale, Kamel a battu Carine, il est donc le vainqueur de la compétition.

Pour s'assurer que chaque finaliste ait joué le même nombre de matchs, un arbre de compétition a toutes ces feuilles à la même hauteur.

Les quatre fonctions suivantes pourront être utilisées :

- La fonction `racine` qui prend en paramètre un arbre de compétition `arb` et renvoie la valeur de la racine.

  Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `racine(A)` vaut "Kamel".

- La fonction `gauche` qui prend en paramètre un arbre de compétition `arb` et renvoie son sous-arbre gauche.

  Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `gauche(A)` vaut l'arbre représenté graphiquement ci-après :

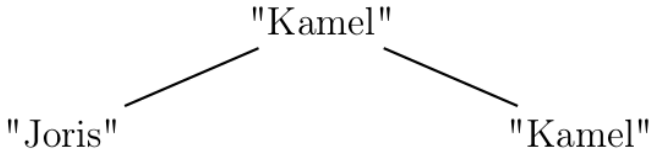


Figure 3: Arbre gauche de A

- La fonction `droit` qui prend en argument un arbre de compétition `arb` et renvoie son sous-arbre droit. Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `droit(A)` vaut l'arbre représenté graphiquement ci-dessous :

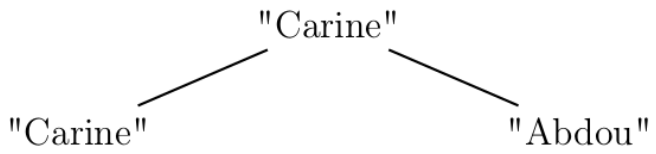


Figure 4: Arbre droit de A

- La fonction `est_vide` qui prend en argument un arbre de compétition et renvoie `True` si l'arbre est vide et `False` sinon.

  Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `est_vide(A)` vaut `False`.

Pour toutes les questions de l'exercice, on suppose que tous les joueurs d'une même compétition ont un prénom différent.

1. On considère l'arbre de compétition B suivant :

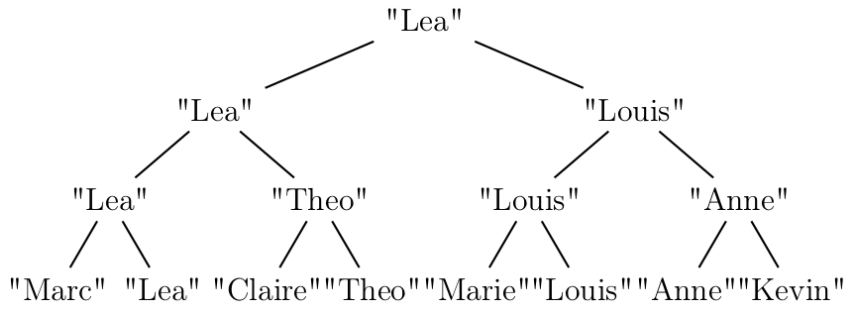


Figure 5: Arbre B

   Indiquer la racine de cet arbre puis donner l'ensemble des valeurs des feuilles de cet arbre.

2. Proposer une fonction Python `vainqueur` prenant pour argument un arbre de compétition `arb` ayant au moins un joueur. Cette fonction doit renvoyer la chaîne de caractères constituée du nom du vainqueur du tournoi. Exemple :

```
>>> vainqueur(B)
'Lea'
```

3. Proposer une fonction Python `finale` prenant pour argument un arbre de compétition `arb` ayant au moins deux joueurs. Cette fonction doit renvoyer le tableau des deux chaînes de caractères qui sont les deux compétiteurs finalistes. Exemple :

```
>>> finale(B)
['Lea', 'Louis']
```

4. Proposer une fonction Python `occurrences` ayant pour paramètre un arbre de compétition `arb` et le nom d'un joueur nom et qui renvoie le nombre d'occurrences (d'apparitions) du joueur nom dans l'arbre de compétition `arb`. Exemple :

```
>>> occurrences(B, "Anne")
2
>>> occurrences(B, "Lea")
4
>>> occurrences(B, "Paul")
0
```

5. Proposer une fonction Python `a_gagne` prenant pour paramètres un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le booléen `True` si le joueur `nom` a gagné au moins un match dans la compétition représenté par l'arbre de compétition arb. Exemple :

```
>>> a_gagne(B,'Louis')
True
>>> a_gagne(B,'Claire')
False
```

6. On souhaite programmer une fonction Python `nombre_matchs` qui prend pour arguments un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le nombre de matchs joués par le joueur `nom` dans la compétition représentée par l'arbre de compétition `arb`. Exemple :

```
>>> nombre_matchs(B,"Lea")
3
>>> nombre_matchs(B,"Marc")
1
```

   a. Expliquer pourquoi les instructions suivantes renvoient une valeur erronée. On pourra pour cela identifier le noeud de l'arbre qui provoque une erreur.

```
def nombre_matchs ( arb , nom ):
    """ arbre_competition , str -> int """
    return occurrences ( arb , nom )
```

b. Proposer une correction pour la fonction `nombre_matchs`.

7. Recopier et compléter la fonction `liste_joueurs` qui prend pour argument un arbre de compétition `arb` et qui renvoie un tableau contenant les participants au tournoi, chaque nom ne devant figurer qu'une seule fois dans le tableau.

   L'opération + à la dernière ligne permet de concaténer deux tableaux.

   Exemple : Si `L1 = [4, 6, 2]` et `L2 = [3, 5, 1]`, l'instruction `L1 + L2` va renvoyer le tableau `[4, 6, 2, 3, 5, 1]`.

```
def liste_joueurs ( arb ):
    """ arbre_competition -> tableau """
    if est_vide ( arb ):
        return ...
    elif ... and ... :
        return [ racine ( arb )]
    else :
        return ...+ liste_joueurs ( droit ( arb ))
```

exercice005

Exercice

Programmer la fonction `recherche`, prenant en paramètre un tableau non vide `tab` (type list) d'entiers et un entier `n`, et qui renvoie l'indice de la dernière occurrence de l'élément cherché. Si l'élément n'est pas présent, la fonction renvoie la longueur du tableau.

Exemples :

```
>>> recherche([5, 3],1)
2
>>> recherche([2,4],2)
0
>>> recherche([2,3,5,2,4],2)
3
```

exercice006

Exercice

On souhaite programmer une fonction donnant la distance la plus courte entre un point de départ et une liste de points. Les points sont tous à coordonnées entières.

Les points sont donnés sous la forme d'un tuple de deux entiers. La liste des points à traiter est donc un tableau de tuples. On rappelle que la distance entre deux points du plan de coordonnées $(x;y)$ et $(x';y')$ est donnée par la formule : $d = \sqrt{(x-x')^2 + (y-y')^2}$.

On importe pour cela la fonction racine carrée (`sqrt`) du module `math` de Python. On dispose d'une fonction `distance` et d'une fonction `plus_courte_distance` :

```
from math import sqrt

# import de la fonction racine carrée
def distance(point1, point2):
    """ Calcule et renvoie la distance entre deux points. """
    return sqrt((...)**2 + (...)**2)

def plus_courte_distance(tab, depart):
    """ Renvoie le point du tableau tab se trouvant à la plus
    courte distance du point depart."""
    point = tab[0]
    min_dist = ...
    for i in range (1, ...):
        if distance(tab[i], depart)...:
            point = ...
            min_dist = ...
    return point
```

Voila un exemple de ce que doivent retourner les fonctions précédentes :

```
>>> distance((1, 0), (5, 3))
5.0
```

```
>>> plus_courte_distance([(7, 9), (2, 5), (5, 2)], (0, 0))
(2, 5)
```

Recopier sous Python (sans les commentaires) ces deux fonctions puis compléter leur code et ajouter une ou des déclarations (assert) à la fonction distance permettant de vérifier la ou les pré-conditions.

exercice007

Exercice

Programmer la fonction `moyenne` prenant en paramètre un tableau d'entiers `tab` (type `list`) qui renvoie la moyenne de ses éléments si le tableau est non vide et affiche `'erreur'` si le tableau est vide.

Exemples :

```
>>> moyenne([5, 3, 8])
5.333333333333333
>>> moyenne([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
5.5
>>> moyenne([])
'erreur'
```

exercice008

Exercice

On considère un tableau d'entiers `tab` (type `list` dont les éléments sont des 0 ou des 1). On se propose de trier ce tableau selon l'algorithme suivant : à chaque étape du tri, le tableau est constitué de trois zones consécutives, la première ne contenant que des 0, la seconde n'étant pas triée et la dernière ne contenant que des 1.

`Zone de 0 |  Zone non triée |  Zone de 1`

Tant que la zone non triée n'est pas réduite à un seul élément, on regarde son premier élément :

- si cet élément vaut 0, on considère qu'il appartient désormais à la zone ne contenant que des 0 ;
- si cet élément vaut 1, il est échangé avec le dernier élément de la zone non triée et on considère alors qu'il appartient à la zone ne contenant que des 1.

Dans tous les cas, à chaque étape du tri, la longueur de la zone non triée diminue de 1.
Recopier sous Python en la complétant la fonction tri suivante :

```
def tri(tab):
      #i est le premier indice de la zone non triee, j le dernier indice.
    #Au debut, la zone non triee est le tableau entier.
    i = ...
    j = ...
    point = tab[0]
    while i < j :
        if tab[i] == 0:
                i = ...
            else:
                valeur = tab[j]
                tab[j] = ...
                ...
                j = ...
    ...
```

Exemple :

```
>>> tri([0,1,0,1,0,1,0,1,0])
[0, 0, 0, 0, 0, 1, 1, 1, 1]
```

exercice009

Exercice

Recopier et compléter sous Python la fonction suivante en respectant la spécification. On ne recopiera pas les commentaires.

```
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    return ...
```

Exemples :

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 27)
False
```

exercice010

Exercice

On modélise la représentation binaire d'un entier non signé par un tableau d'entiers dont les éléments sont 0 ou 1. Par exemple, le tableau `[1, 0, 1, 0, 0, 1, 1]` représente l'écriture binaire de l'entier dont l'écriture décimale est

```
2**6 + 2**4 + 2**1 + 2**0 = 83
```

À l'aide d'un parcours séquentiel, écrire la fonction convertir répondant aux spécifications suivantes :

```
def convertir(T):
    """
    T est un tableau d'entiers, dont les éléments sont 0 ou 1 et
    représentant un entier écrit en binaire.
    Renvoie l'entier positif dont la représentation binaire
    est donnée par le tableau T
    """

>>> convertir([1, 0, 1, 0, 0, 1, 1])
83
>>> convertir([1, 0, 0, 0, 0, 0, 1, 0])
130
```

exercice011

Exercice

La fonction `tri_insertion` suivante prend en argument une liste `L` et trie cette liste en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

```
def tri_insertion(L):
    n = len(L)
    # cas du tableau vide
    if ...:
        return L
    for j in range(1, n):
        e = L[j]
        i = j
        # À l'étape j, le sous-tableau L[0,j-1] est trié
        # et on insère L[j] dans ce sous-tableau en déterminant
        # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].
        while i > 0 and L[i-1] > ...:
            i = ...
        # si i != j, on décale le sous tableau L[i,j-1] d'un cran
        # vers la droite et on place L[j] en position i
        if i != j:
            for k in range(j,i,...):
                L[k] = L[...]
            L[i] = ...
    return L
```

Exemples :

```
>>> tri_insertion([2, 5, -1, 7, 0, 28])
[-1, 0, 2, 5, 7, 28]
>>> tri_insertion([10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

exercice012

Exercice

Cet exercice traite principalement du thème «architectures matérielles, systèmes d'exploitation et réseaux. Cet exercice mobilise des connaissances sur le routage et sur l'évolution des architectures des machines.

Partie A : Routage dans un réseau informatique

1. Expliquer pourquoi le protocole TCP-IP prévoit un découpage en paquets et une encapsulation des fichiers transférés d'un ordinateur à un autre via Internet.
2. On souhaite modéliser un réseau informatique par un graphe pondéré pour identifier le chemin optimal pour un paquet.
    a. Préciser ce que représentent les sommets et les arêtes du graphe.
    b. Préciser si le protocole RIP utilise le nombre de sauts ou le délai de réception comme poids des arêtes.

Partie B : Système d'exploitation

Un système d'exploitation doit assurer la gestion des processus et des ressources.

1. Dans ce contexte, expliquer et illustrer par un exemple ce qu'est une situation d'interblocage (deadlock).

2. Citer des mécanismes permettant d'éviter ces situations.

Partie C : Architectures matérielles

Architecture Von Neumann

«L'architecture dite architecture de Von Neumann est un modèle pour un ordinateur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul. De telles machines sont aussi connues sous le nom d'ordinateur à programme enregistré.»

source : Wikipédia

Elle décompose l'ordinateur en 4 éléments : l'unité de contrôle (appelé aussi unité de commande), l'unité arithmétique et logique (UAL), la mémoire et les entrées-sorties. Les deux premiers éléments sont rassemblés dans le processeur (CPU en anglais pour Control Processing Unit).

1. Recopier et compléter le schéma de cette architecture ci-dessous en faisant apparaître les communications entre les différents éléments.

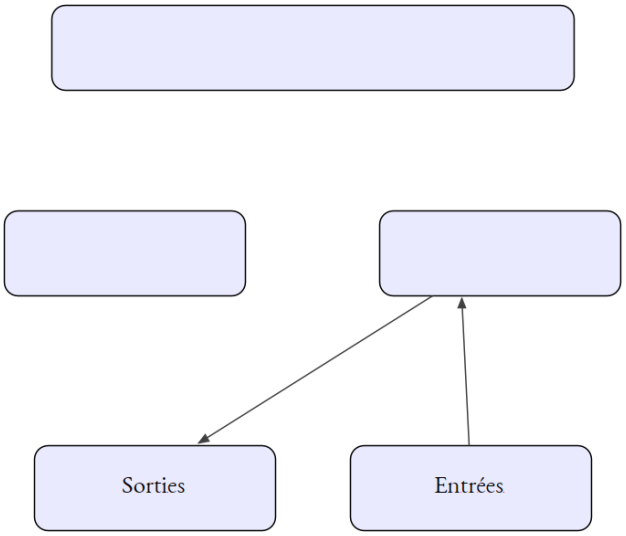


FIG. **1 :** Von Neumann

2. Dans quel(s) élément(s) sont situés le « compteur de programme » (CP ou IP en anglais pour Instruction Pointer) et le « registre d'instruction » (RI ou IR en anglais pour Instruction Register). Préciser leurs rôles.

Architecture de Harvard

« L'architecture de type Harvard est une conception qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. […] L'architecture Harvard est souvent utilisée dans les processeurs numériques de signal (DSP) et les microcontrôleurs. »

source : Wikipédia

3. Recopier et compléter le schéma de cette architecture ci-dessous et faire apparaître les communications entre les différents éléments.

4. Expliquer ce qu'est une mémoire morte et une mémoire vive. Expliquer brièvement pourquoi, dans les microcontrôleurs, la mémoire programme est une mémoire morte.

Partie D : Système sur puce

« Un "système sur une puce", souvent désigné dans la littérature scientifique par le terme anglais "system on a chip" (d'où son abréviation SoC), est un système complet embarqué sur une seule puce ("circuit intégré"), pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue. »

source : Wikipédia

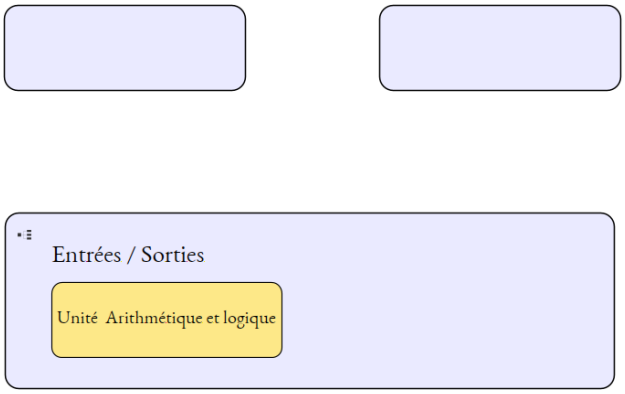


FIG. **2 :** Harvard

1. Citer un des avantages d'avoir plusieurs processeurs.
2. Expliquer pourquoi les systèmes sur puces intègrent en général des bus ayant des vitesses de transmission différentes.
3. Citer un des avantages d'un circuit imprimé de petite taille.
4. Citer un des inconvénients de cette miniaturisation.

exercice013

Exercice

Cet exercice porte sur les arbres binaires de recherche.

Dans cet exercice, les arbres binaires de recherche ne peuvent pas comporter plusieurs fois la même clé. De plus, un arbre binaire de recherche limité à un nœud a une hauteur de 1.

On considère l'arbre binaire de recherche représenté ci-dessous (figure 1), où val représente un entier :

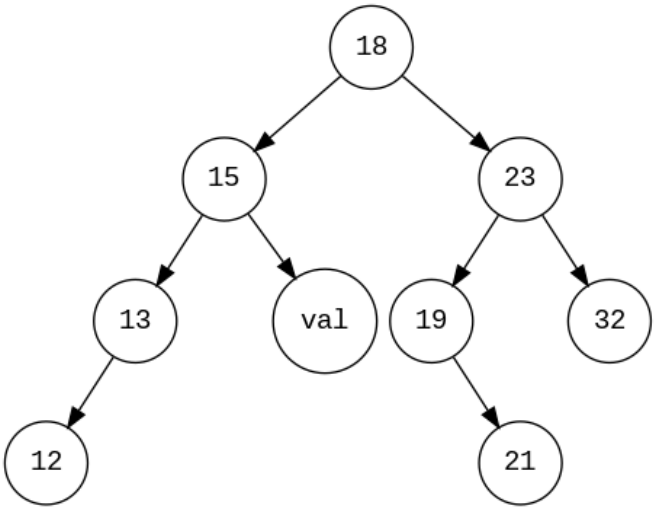


FIG. **1 :** figure 1

1. Donner le nombre de feuilles de cet arbre et préciser leur valeur (étiquette).
    a. Donner le sous arbre-gauche du nœud 23.
    b. Donner la hauteur et la taille de l'arbre.
    c. Donner les valeurs entières possibles de `val` pour cet arbre binaire de recherche.

On suppose, pour la suite de cet exercice, que `val` est égal à 16.

2. On rappelle qu'un parcours infixe depuis un nœud consiste, dans l'ordre, à faire un parcours infixe sur le sous arbre-gauche, afficher le nœud puis faire un parcours infixe sur le sous-arbre droit.

Dans le cas d'un parcours suffixe, on fait un parcours suffixe sur le sous-arbre gauche puis un parcours suffixe sur le sous-arbre droit, avant d'afficher le nœud.

a. Donner les valeurs d'affichage des nœuds dans le cas du parcours infixe de l'arbre.
b. Donner les valeurs d'affichage des nœuds dans le cas du parcours suffixe de l'arbre.

3. On considère la classe Noeud définie de la façon suivante en Python :

```
class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere :
            if v == n.v:
                # Bloc 1
                est_insere = True
                # Fin Bloc 1
            elif v < n.v:
                # Bloc 2
                if n.ag != None:
                    n = n.ag
                else:
                    n.ag = Noeud(v)
                    est_insere = True
                # Fin Bloc 2
            else:
                # Bloc 3
                if n.ad != None:
                    n = n.ad
                else:
                    n.ad = Noeud(v)
                    est_insere = True
                # Fin Bloc 3

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)
```

a. Représenter l'arbre construit suite à l'exécution de l'instruction suivante :

```
racine = Noeud(18)
racine.insere_tout([12, 13, 15, 16, 19, 21, 32, 23])
```

b. Écrire les deux instructions permettant de construire l'arbre de la figure 1. On rappelle que le nombre `val` est égal à 16.

c. On considère l'arbre tel qu'il est présenté sur la figure 1. Déterminer l'ordre d'exécution des blocs (repérés de 1 à 3) suite à l'application de la méthode `insere(19)` au nœud racine de cet arbre.

4. Écrire une méthode `recherche(self, v)` qui prend en argument un entier `v` et renvoie la valeur `True` si cet entier est une étiquette de l'arbre, `False` sinon.

exercice014

Exercice

Cet exercice porte sur les bases de données et le langage SQL.

L'énoncé de cet exercice utilise les mots du langage SQL suivants : `SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES, UPDATE, SET, DELETE, COUNT, AND,OR.`

Pour la gestion des réservations clients, on dispose d'une base de données nommée « `gare` » dont le schéma relationnel est le suivant :

```
Train (numT, provenance, destination, horaireArrivee, horaireDepart)
Reservation (numR, nomClient, prenomClient, prix, #numT)
```

Les attributs `numT` et `numR` sont respectivement les clés primaires des tables `Train` et `Reservation`. L'attribut précédé de # est une clé étrangère. La clé étrangère `Reservation.numT` fait référence à la clé primaire `Train.numT`.

Les attributs `horaireDepart` et `horaireArrivee` sont de type `TIME` et s'écrivent selon le format `"hh:mm"`, où `hh` représente les heures et `mm` les minutes.

1. Quel nom générique donne-t-on aux logiciels qui assurent, entre autres, la persistance des données, l'efficacité de traitement des requêtes et la sécurisation des accès pour les bases de données ?

2. a. On considère les requêtes SQL suivantes :

   ```
   DELETE FROM Train WHERE numT = 1241 ;
   DELETE FROM Reservation WHERE numT = 1241 ;
   ```

   Sachant que le train n°1241 a été enregistré dans la table `Train` et que des réservations pour ce train ont été enregistrées dans la table `Reservation`, expliquer pourquoi cette suite d'instructions renvoie une erreur.

   b. Citer un cas pour lequel l'insertion d'un enregistrement dans la table `Reservation` n'est pas possible.

3. Écrire des requêtes SQL correspondant à chacune des instructions suivantes :

   a. Donner tous les numéros des trains dont la destination est « Lyon ».

b. Ajouter une réservation n°1307 de 33 € pour M. Alan Turing dans le train n°654.
c. Suite à un changement, l'horaire d'arrivée du train n°7869 est programmé à 08 h 11. Mettre à jour la base de données en conséquence.

4. Que permet de déterminer la requête suivante ?

```
SELECT COUNT(*) FROM Reservation
WHERE nomClient = "Hopper" AND prenomClient = "Grace";
```

5. Écrire la requête qui renvoie les destinations et les prix des réservations effectuées par Grace Hopper.

exercice015

Exercice

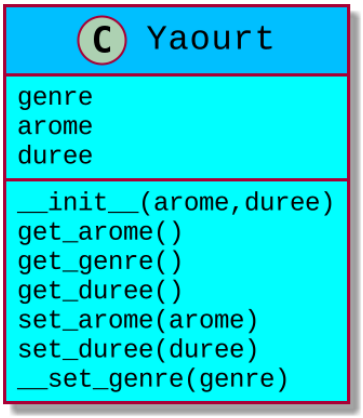
Principaux thèmes abordés : structure de données (programmation objet) et langages et programmation (spécification).

Une entreprise fabrique des yaourts qui peuvent être soit nature (sans arôme), soit aromatisés (fraise, abricot ou vanille).

Pour pouvoir traiter informatiquement les spécificités de ce produit, on va donc créer une classe Yaourt qui possèdera un certain nombre d'attributs :

- Son genre : nature ou aromatisé
- Son arôme : fraise, abricot, vanille ou aucun
- Sa date de durabilité minimale (DDM) exprimée par un entier compris entre 1 et 365 (on ne gère pas les années bissextiles). Par exemple, si la DDM est égale à 15, la date de durabilité minimale est le 15 janvier.

On va créer également des méthodes permettant d'interagir avec l'objet `Yaourt` pour attribuer un arôme ou récupérer un genre par exemple. On peut représenter cette classe par le tableau de spécifications ci-dessous :



1. On a commencé à implémenter la classe `Yaourt`. Le code ci-dessous sera à compléter.

```python
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
        - son arome
        - son genre
        - sa durée de durabilité minimale"""

    def __init__(self,arome,duree):
        ####
        # À compléter :Q.1.a
        ###
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'

    def get_arome(self):
        # À rédiger
        pass

    def get_duree(self):
        return self.__duree

    def get_genre(self):
        return self.__genre

    def set_duree(self,duree):
        if duree > 0 :
            self.__duree = duree

    def set_arome(self, arome):
        # À rédiger
        pass

    def __set_genre(self,arome):
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'
```

a. Quelles sont les assertions à prévoir pour vérifier que l'arôme et la durée correspondent bien à des valeurs acceptables. Il faudra aussi expliciter les commentaires qui seront renvoyés. Pour rappel :
   - L'arôme doit prendre comme valeur `'fraise'`, `'abricot'`, `'vanille'` ou `'aucun'`.
   - Sa date de durabilité minimale (DDM) est une valeur positive.

b. Pour créer un yaourt, on exécutera la commande suivante :

```
>>> Mon_Yaourt = Yaourt('fraise',24)
```

   Quelle valeur sera affectée à l'attribut genre associé à `Mon_Yaourt` ?

c. Écrire en python une fonction `get_arome(self)`, renvoyant l'arôme du yaourt créé.

2. On appelle mutateur une méthode permettant de modifier un ou plusieurs attributs d'un objet. Sur votre copie, écrire en Python le mutateur `set_arome(self,arome)` permettant de modifier l'arôme du yaourt. On veillera à garder une cohérence entre l'arôme et le genre.
3. On veut créer une pile contenant le stock de yaourts. Pour cela il faut tout d'abord créer une pile vide :

```
>>> def creer_pile():
...     pile = [ ]
...     return pile
```

   a. Créer une fonction `empiler(p, Yaourt)` qui renvoie la pile `p` après avoir ajouté un objet de type `Yaourt` à la fin.
   b. Créer une fonction `depiler(p)` qui renvoie l'objet à dépiler.
   c. Créer une fonction `est_vide(p)` qui renvoie `True` si la pile est vide et `False` sinon.
   d. Qu'affiche le bloc de commandes suivantes ci-dessous ?

```
>>> mon_Yaourt1 = Yaourt('aucun',18)
>>> mon_Yaourt2 = Yaourt('fraise',24)
>>> ma_pile = creer_pile()
>>> empiler(ma_pile, mon_Yaourt1)
>>> empiler(ma_pile, mon_Yaourt2)
>>> print(depiler(ma_pile).get_duree())
>>> print(est_vide(ma_pile))
```

exercice017

Exercice

Écrire une fonction `recherche` qui prend en paramètres `caractere`, un caractère, et `mot`, une chaîne de caractères, et qui renvoie le nombre d'occurrences de `caractere` dans `mot`, c'est-à-dire le nombre de fois où `caractere` apparaît dans `mot`.

Exemples :

```
>>> recherche('e', "sciences")
2
>>> recherche('i',"mississippi")
4
>>> recherche('a',"mississippi")
0
```

exercice017

Exercice

On s'intéresse à un algorithme récursif qui permet de rendre la monnaie à partir d'une liste donnée de valeurs de pièces et de billets.

Le système monétaire est donné sous forme d'une liste `pieces=[100, 50, 20, 10, 5, 2, 1]`. (on supposera qu'il n'y a pas de limitation quant à leur nombre).

On cherche à donner la liste de pièces à rendre pour une somme donnée en argument. Compléter le code Python ci-dessous de la fonction `rendu_glouton` qui implémente cet algorithme et renvoie la liste des pièces à rendre.

```
pieces = [100,50,20,10,5,2,1]

def rendu_glouton(arendre, solution=[], i=0):
    if arendre == 0:
        return ...
    p = pieces[i]
    if p <= ... :
        solution.append(...)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, ...)
```

On devra obtenir :

```
>>> rendu_glouton(68, [], 0)
[50, 10, 5, 2, 1]
>>> rendu_glouton(291, [], 0)
[100, 100, 50, 20, 20, 1]
```

exercice018

Exercice

Écrire une fonction `rechercheMinMax` qui prend en paramètre un tableau de nombres non triés `tab`, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés `min` et `max`. Les tableaux seront représentés sous forme de liste Python.

Exemples :

```
>>> tableau = [0, 1, 4, 2, -2, 9, 3, 1, 7, 1]
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': -2, 'max': 9}
>>> tableau = []
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': None, 'max': None}
```

exercice019

Exercice

On dispose d'un programme permettant de créer un objet de type `PaquetDeCarte`, selon les éléments indiqués dans le code ci-dessous. Compléter ce code aux endroits indiqués par ???, puis ajouter des assertions dans l'initialiseur de `Carte`.

```
class Carte:
    def __init__(self, c, v):
        """Initialise Couleur (entre 1 a 4), et Valeur (entre 1 a 13)"""
        self.Couleur = c
        self.Valeur = v

    def getNom(self):
        """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame, Roi"""
        if ( self.Valeur > 1 and self.Valeur < 11):
            return str(self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
        else:
            return "As"

    def getCouleur(self):
        """Renvoie la couleur de la Carte (parmi pique, coeur, carreau, trefle"""
        return ['pique', 'coeur', 'carreau', 'trefle' ][self.Couleur - 1]

class PaquetDeCarte:
    def __init__(self):
```

```
        self.contenu = []

    def remplir(self):
        """Remplit le paquet de cartes"""
        for couleur in range(1, ???):
            for valeur in range(1, ???):
                ???

    def getCarteAt(self, pos):
    """Renvoie la Carte qui se trouve à la position donnee"""
        if 0 <= pos < ??? :
            return ???
```

Exemple :

```
>>> unPaquet = PaquetDeCarte()
>>> unPaquet.remplir()
>>> uneCarte = unPaquet.getCarteAt(20)
>>> print(uneCarte.getNom() + " de " + uneCarte.getCouleur())
8 de coeur
```

exercice020

Exercice

Écrire une fonction `maxi` qui prend en paramètre une liste `tab` de nombres entiers et renvoie un couple donnant le plus grand élément de cette liste, ainsi que l'indice de la première apparition de ce maximum dans la liste.

Exemple :

```
>>> maxi([1,5,6,9,1,2,3,7,9,8])
(9, 3)
```

exercice021

Exercice

La fonction `recherche` prend en paramètres deux chaines de caractères `gene` et `seq_adn` et renvoie `True` si on retrouve `gene` dans `seq_adn` et `False` sinon. Compléter le code Python ci-dessous pour qu'il implémente la fonction `recherche`.

```
def recherche(gene, seq_adn):
    n = len(seq_adn)
    g = len(gene)
    i = ...
    trouve = False
    while i < ... and trouve == ... :
        j = 0
        while j < g and gene[j] == seq_adn[i+j]:
            ...
        if j == g:
            trouve = True
        ...
    return trouve
```

Exemples :

```
>>> recherche("AATC", "GTACAAATCTTGCC")
True
>>> recherche("AGTC", "GTACAAATCTTGCC")
False
```

exercice022

Exercice

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la première occurrence de `elt` dans `tab` si `elt` est dans `tab` et `-1` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(50, [1, 50, 1])
1
>>> recherche(15, [8, 9, 10, 15])
3
```

exercice023

Exercice

On considère la fonction `insere` ci-dessous qui prend en argument un entier `a` et un tableau `tab` d'entiers triés par ordre croissant. Cette fonction insère la valeur `a` dans le tableau et renvoie le nouveau tableau. Les tableaux seront représentés sous la forme de listes python.

```
def insere(a, tab):
    l = list(tab) #l contient les mêmes éléments que tab
    l.append(a)
    i = ...
    while a < ... and i >= 0:
        l[i+1] = ...
        l[i] = a
        i = ...
    return l
```

Compléter la fonction `insere` ci-dessus.
Exemples :

```
>>> insere(3,[1,2,4,5])
[1, 2, 3, 4, 5]
>>> insere(10,[1,2,7,12,14,25])
[1, 2, 7, 10, 12, 14, 25]
>>> insere(1,[2,3,4])
[1, 2, 3, 4]
```

exercice024

Exercice

Écrire une fonction python appelée `nb_repetitions` qui prend en paramètres un élément `elt` et une liste `tab` et renvoie le nombre de fois où l'élément apparaît dans la liste.

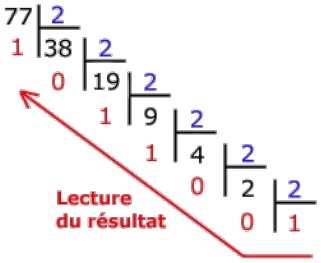
Exemples :

```
>>> nb_repetitions(5, [2, 5, 3, 5, 6, 9, 5])
3
>>> nb_repetitions('A', ['B', 'A', 'B', 'A', 'R'])
2
>>> nb_repetitions(12, [1, '!', 7, 21, 36, 44])
0
```

exercice025

Exercice

Pour rappel, la conversion d'un nombre entier positif en binaire peut s'effectuer à l'aide des divisions successives comme illustré ici :



Voici une fonction Python basée sur la méthode des divisions successives permettant de convertir un nombre entier positif en binaire :

```
def binaire(a):
    bin_a = str(...)
    a = a // 2
    while a ... :
        bin_a = ...(a%2) + ...
        a = ...
    return bin_a
```

Compléter la fonction binaire.
Exemples :

```
>>> binaire(0)
'0'
>>> binaire(77)
'1001101'
```

exercice026

Exercice

Cet exercice utilise des piles qui seront représentées en Python par des listes (type `list`).

On rappelle que l'expression `T1 = list(T)` fait une copie de `T` indépendante de `T`, que l'expression `x = T.pop()` enlève le sommet de la pile `T` et le place dans la variable `x` et, enfin, que l'expression `T.append(v)` place la valeur `v` au sommet de la pile `T`.

Compléter le code Python de la fonction `positif` ci-dessous qui prend une pile `T` de nombres entiers en paramètre et qui renvoie la pile des entiers positifs dans le même ordre, sans modifier la variable `T`.

```
def positif(T):
    T2 = ...(T)
    T3 = ...
    while T2 != []:
        x = ...
        if ... >= 0:
            T3.append(...)
    T2 = []
    while T3 != ...:
        x = T3.pop()
        ...
    print('T = ',T)
    return T2
```

Exemple :

```
>>> positif([-1, 0, 5, -3, 4, -6, 10, 9, -8])
T = [-1, 0, 5, -3, 4, -6, 10, 9, -8]
[0, 5, 4, 10, 9]
```

exercice027

Exercice

Pour cet exercice :

- On appelle «mot» une chaîne de caractères composée avec des caractères choisis parmi les 26 lettres minuscules ou majuscules de l'alphabet,
- On appelle «phrase» une chaîne de caractères :
  - composée avec un ou plusieurs «mots» séparés entre eux par un seul caractère espace `' '`,
  - se finissant :
    - soit par un point `'.'` qui est alors collé au dernier mot,
    - soit par un point d'exclamation `'!'` ou d'interrogation `'?'` qui est alors séparé du dernier mot par un seul caractère espace `' '`.

Après avoir remarqué le lien entre le nombre de mots et le nombre de caractères espace dans une phrase, programmer une fonction `nombre_de_mots` qui prend en paramètre une phrase et renvoie le nombre de mots présents dans cette phrase.

Exemples :

```
>>> nombre_de_mots('Le point d exclamation est separe !')
6
>>> nombre_de_mots('Il y a un seul espace entre les mots.')
9
```

exercice028

Exercice

On a relevé les valeurs moyennes annuelles des températures à Paris pour la période allant de 2013 à 2019. Les résultats ont été récupérés sous la forme de deux listes : l'une pour les températures, l'autre pour les années :

```
t_moy = [14.9, 13.3, 13.1, 12.5, 13.0, 13.6, 13.7]
annees = [2013, 2014, 2015, 2016, 2017, 2018, 2019]
```

Écrire la fonction `mini` qui prend en paramètres le tableau `releve` des relevés et le tableau `date` des dates et qui renvoie la plus petite valeur relevée au cours de la période et l'année correspondante.

Exemple :

```
>>> mini(t_moy, annees)
(12.5, 2016)
```

exercice029

Exercice

Un mot palindrome peut se lire de la même façon de gauche à droite ou de droite à gauche : *bob*, *radar*, et *non* sont des mots palindromes.

De même certains nombres sont eux aussi des palindromes : 33, 121, 345543.

L'objectif de cet exercice est d'obtenir un programme Python permettant de tester si un nombre est un nombre palindrome.

Pour remplir cette tâche, on vous demande de compléter le code des trois fonctions ci- dessous sachant que la fonction `est_nbre_palindrome` s'appuiera sur la fonction `est_palindrome` qui elle-même s'appuiera sur la fonction `inverse_chaine`.

La fonction `inverse_chaine` inverse l'ordre des caractères d'une chaîne de caractères `chaine` et renvoie la chaîne inversée.

La fonction `est_palindrome` teste si une chaine de caractères `chaine` est un palindrome. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

La fonction `est_nbre_palindrome` teste si un nombre `nbre` est un palindrome. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

Compléter le code des trois fonctions ci-dessous.

```
def inverse_chaine(chaine):
    result = ...
    for caractere in chaine:
        result = ...
    return result

def est_palindrome(chaine):
    inverse = inverse_chaine(chaine)
    return ...

def est_nbre_palindrome(nbre):
    chaine = ...
    return est_palindrome(chaine)
```

Exemples :

```
>>> inverse_chaine('bac')
'cab'
>>> est_palindrome('NSI')
False
>>> est_palindrome('ISN-NSI')
True
>>> est_nbre_palindrome(214312)
False
>>> est_nbre_palindrome(213312)
True
```

exercice030

Exercice

Écrire la fonction `maxliste`, prenant en paramètre un tableau non vide de nombres `tab` (type `list`) et renvoyant le plus grand élément de ce tableau.

Exemples :

```
>>> maxliste([98, 12, 104, 23, 131, 9])
131
>>> maxliste([-27, 24, -3, 15])
24
```

exercice031

Exercice

On dispose de chaînes de caractères contenant uniquement des parenthèses ouvrantes et fermantes.

Un parenthésage est correct si :

- le nombre de parenthèses ouvrantes de la chaîne est égal au nombre de parenthèses fermantes.
- en parcourant la chaîne de gauche à droite, le nombre de parenthèses déjà ouvertes doit être, à tout moment, supérieur ou égal au nombre de parenthèses déjà fermées.

Ainsi, `(((()))(()))` est un parenthésage correct.

Les parenthésages `())(()` et `(())(()` sont, eux, incorrects.

On dispose du code de la classe `Pile` suivant :

```
class Pile:
    """ Classe définissant une pile """
    def __init__(self, valeurs=[]):
        self.valeurs = valeurs

    def est_vide(self):
        """Renvoie True si la pile est vide, False sinon"""
        return self.valeurs == []

    def empiler(self, c):
        """Place l'élément c au sommet de la pile"""
        self.valeurs.append(c)

    def depiler(self):
        """Supprime l'élément placé au sommet de la pile,
        à condition qu'elle soit non vide"""
        if self.est_vide() == False:
            self.valeurs.pop()
```

On souhaite programmer une fonction `parenthesage` qui prend en paramètre une chaîne `ch` de parenthèses et renvoie `True` si la chaîne est bien parenthésée et `False` sinon. Cette fonction utilise une pile et suit le principe suivant : en parcourant la chaîne de gauche à droite, si on trouve une parenthèse ouvrante, on l'empile au sommet de la pile et si on trouve une parenthèse fermante, on dépile (si possible !) la parenthèse ouvrante stockée au sommet de la pile.

La chaîne est alors bien parenthésée si, à la fin du parcours, la pile est vide.

Elle est, par contre, mal parenthésée :

- si dans le parcours, on trouve une parenthèse fermante, alors que la pile est vide ;
- ou si, à la fin du parcours, la pile n'est pas vide.

```
def parenthesage(ch):
    """Renvoie True si la chaîne ch est bien parenthésée et False sinon"""
    p = Pile()
    for c in ch:
        if c == ...:
            p.empiler(c)
        elif c == ...:
            if p.est_vide():
                return ...
            else:
                ...
    return p.est_vide()

assert parenthesage("((()())(()))") == True
assert parenthesage("())(()") == False
assert parenthesage("(())(()") == False
```

Compléter le code de la fonction `parenthesage`.

exercice032

Exercice

Écrire en langage Python une fonction `recherche` prenant comme paramètres une variable `a` de type numérique (`float` ou `int`) et un tableau `t` (type `list`) et qui renvoie le nombre d'occurrences de `a` dans `t`.

Exemples :

```
>>> recherche(5,[])
0
>>> recherche(5,[-2, 3, 4, 8])
0
>>> recherche(5,[-2, 3, 1, 5, 3, 7, 4])
1
>>> recherche(5,[-2, 5, 3, 5, 4, 5])
3
```

exercice033

Exercice

La fonction `rendu_monnaie_centimes` prend en paramètres deux nombres entiers positifs `s_due` et`s_versee` et elle permet de procéder au rendu de monnaie de la différence `s_versee - s_due` pour des achats effectués avec le système de pièces de la zone Euro. On utilise pour cela un algorithme qui commence par rendre le maximum de pièces de plus grandes valeurs et ainsi de suite. La fonction renvoie la liste des pièces qui composent le rendu.

Toutes les sommes sont exprimées en centimes d'euros. Les valeurs possibles pour les pièces sont donc `[1, 2, 5, 10, 20, 50, 100, 200]`.

Ainsi, l'instruction `rendu_monnaie_centimes(452, 500)` renverra `[20, 20, 5, 2, 1]`.

En effet, la somme à rendre est de 48 centimes soit `20 + 20 + 5 + 2 + 1`. Le code de la fonction est donné ci-dessous :

```
def rendu_monnaie_centimes(s_due, s_versee):
    pieces = [1, 2, 5, 10, 20, 50, 100, 200]
    rendu = ...
    a_rendre = ...
    i = len(pieces) - 1
    while a_rendre > ... :
        if pieces[i] <= a_rendre :
            rendu.append(...)
            a_rendre = ...
        else :
            i = ...
    return rendu
```

Compléter ce code pour qu'il donne :

```
>>> rendu_monnaie_centimes(700,700)
[]
>>> rendu_monnaie_centimes(112,500)
[200, 100, 50, 20, 10, 5, 2, 1]
```

exercice034

Exercice

On s'intéresse au problème du rendu de monnaie. On suppose qu'on dispose d'un nombre infini de billets de 5 euros, de pièces de 2 euros et de pièces de 1 euro. Le but est d'écrire une fonction nommée `rendu` dont le paramètre est un entier positif non nul `somme_a_rendre` et qui retourne une liste de trois entiers `n1`, `n2` et `n3` qui correspondent aux nombres de billets de 5 euros (`n1`) de pièces de 2 euros (`n2`) et de pièces de 1 euro (`n3`) à rendre afin que le total rendu soit égal à `somme_a_rendre`.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de 5 euros, puis celui des pièces de 2 euros et enfin celui des pièces de 1 euros.

Exemples :

```
>>> rendu(13)
[2,1,1]
>>> rendu(64)
[12,2,0]
>>> rendu(89)
[17,2,0]
```

exercice035

Exercice

Écrire une fonction `recherche` qui prend en paramètre un tableau de nombres entiers `tab`, et qui renvoie la liste (éventuellement vide) des couples d'entiers consécutifs successifs qu'il peut y avoir dans `tab`.

Exemples :

```
>>> recherche([1, 4, 3, 5])
[]
>>> recherche([1, 4, 5, 3])
[(4, 5)]
>>> recherche([7, 1, 2, 5, 3, 4])
[(1, 2), (3, 4)]
>>> recherche([5, 1, 2, 3, 8, -5, -4, 7])
[(1, 2), (2, 3), (-5, -4)]
```

exercice036

Exercice

Soit une image binaire représentée dans un tableau à 2 dimensions. Les éléments `M[i][j]`, appelés pixels, sont égaux soit à `0` soit à `1`.

Une composante d'une image est un sous-ensemble de l'image constitué uniquement de `1` et de `0` qui sont côte à côte, soit horizontalement soit verticalement.

Par exemple, les composantes de

`M =`

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

sont

`M =`

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

On souhaite, à partir d'un pixel égal à `1` dans une image `M`, donner la valeur `val` à tous les pixels de la composante à laquelle appartient ce pixel.

La fonction `propager` prend pour paramètre une image `M`, deux entiers `i` et `j` et une valeur entière `val`. Elle met à la valeur `val` tous les pixels de la composante du pixel `M[i][j]` s'il vaut `1` et ne fait rien s'il vaut `0`.

Par exemple, `propager(M,2,1,3)` donne

`M =`

0	0	1	0
0	3	0	1
3	**3**	3	0
0	3	3	0

Compléter le code récursif de la fonction `propager` donné ci-dessous :

```
def propager(M, i, j, val):
    if M[i][j]== ...:
        return

    M[i][j] = val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == ...):
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((...) < len(M) and M[i+1][j] == 1):
        propager(M, ..., j, val)

    # l'élément à gauche fait partie de la composante
    if ((...) >= 0 and M[i][j-1] == 1):
        propager(M, i, ..., val)

    # l'élément à droite fait partie de la composante
    if ((...) < len(M) and M[i][j+1] == 1):
        propager(M, i, ..., val)
```

Exemple :

```
>>> M = [[0,0,1,0],[0,1,0,1],[1,1,1,0],[0,1,1,0]]
>>> propager(M,2,1,3)
>>> M
[[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```

exercice037

Exercice

Écrire une fonction `occurrence_max` prenant en paramètres une chaîne de caractères `chaine` et qui renvoie le caractère le plus fréquent de la chaîne. La chaine ne contient que des lettres en minuscules sans accent. On pourra s'aider du tableau

```
alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
            'r','s','t','u','v','w','x','y','z']
```

et du tableau `occurrence` de 26 éléments où l'on mettra dans `occurrence[i]` le nombre d'apparitions de `alphabet[i]` dans la chaine.
Puis on calculera l'indice `k` d'un maximum du tableau `occurrence` et on affichera `alphabet[k]`.

Exemple :

```
>>> ch = 'je suis en terminale et je passe le bac et je souhaite poursuivre
         des etudes pour devenir expert en informatique'
>>> occurrence_max(ch)
'e'
```

exercice038

Exercice

On considère une image en 256 niveaux de gris que l'on représente par une grille de nombres, c'est-à-dire une liste composée de sous-listes toutes de longueurs identiques. La largeur de l'image est donc la longueur d'une sous-liste et la hauteur de l'image est le nombre de sous-listes.

Chaque sous-liste représente une ligne de l'image et chaque élément des sous-listes est un entier compris entre 0 et 255, représentant l'intensité lumineuse du pixel.

Compléter le programme ci-dessous :

```
def nbLig(image):
    '''renvoie le nombre de lignes de l'image'''
    return ...

def nbCol(image):
    '''renvoie la largeur de l'image'''
    return ...

def negatif(image):
    '''renvoie le négatif de l'image sous la forme d'une liste de listes'''
    # on créé une image de 0 aux mêmes dimensions que le paramètre image
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]
    for i in range(len(image)):
        for j in range(...):
            L[i][j] = ...
    return L

def binaire(image, seuil):
    '''renvoie une image binarisée de l'image sous la forme
    d'une liste de listes contenant des 0 si la valeur
    du pixel est strictement inférieure au seuil
    et 1 sinon'''
```

```
    # on crée une image de 0 aux mêmes dimensions que le paramètre image
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]
    for i in range(len(image)):
        for j in range(...):
            if image[i][j] < ... :
                L[i][j] = ...
            else:
                L[i][j] = ...
    return L
```

Exemples :

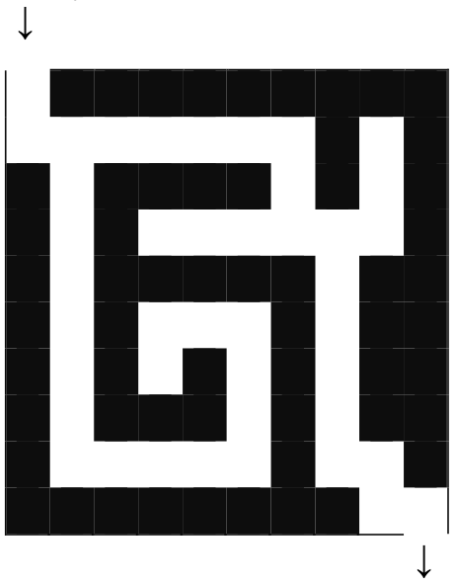
```
>>> img = [[20, 34, 254, 145, 6], [23, 124, 237, 225, 69],
...        [197, 174, 207, 25, 87], [255, 0, 24, 197, 189]]
>>> nbLig(img)
4
>>> nbCol(img)
5
>>> negatif(img)
[[235, 221, 1, 110, 249], [232, 131, 18, 30, 186],
[58, 81, 48, 230, 168], [0, 255, 231, 58, 66]]
>>> binaire(negatif(img),120)
[[1, 1, 0, 0, 1], [1, 1, 0, 0, 1], [0, 0, 0, 1, 1], [0, 1, 1, 0, 0]]
```

exercice039

Exercice

L'objectif de cet exercice est de mettre en place une modélisation d'un jeu de labyrinthe en langage Python.

On décide de représenter un labyrinthe par un tableau carré de taille `n`, dans lequel les cases seront des 0 si l'on peut s'y déplacer et des 1 s'il s'agit d'un mur. Voici un exemple de représentation d'un labyrinthe :



```
>>> laby=[[0,1,1,1,1,1,1,1,1,1],
...       [0,0,0,0,0,0,0,1,0,1],
...       [1,0,1,1,1,1,0,1,0,1],
...       [1,0,1,0,0,0,0,0,0,1],
...       [1,0,1,1,1,1,1,0,1,1],
...       [1,0,1,0,0,0,1,0,1,1],
...       [1,0,1,0,1,0,1,0,1,1],
...       [1,0,1,1,1,0,1,0,1,1],
...       [1,0,0,0,0,0,1,0,0,1],
...       [1,1,1,1,1,1,1,1,0,0]]
```

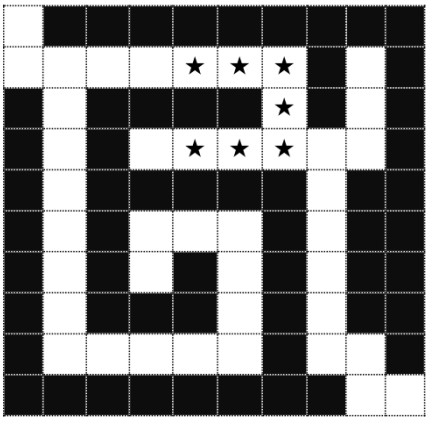
L'entrée du labyrinthe se situe à la première case du tableau (celle en haut à gauche) et la sortie du labyrinthe se trouve à la dernière case (celle en bas à droite).

1. Proposer, en langage Python, une fonction `mur()`, prenant en paramètre un tableau représentant un labyrinthe et deux entiers `i` et `j` compris entre 0 et $n-1$ et qui renvoie un booléen indiquant la présence ou non d'un mur.

   Par exemple :

   ```
   >>> mur(laby, 2, 3)
   True
   >>> mur(laby, 1, 8)
   False
   ```

Un parcours dans le labyrinthe va être représenté par une liste de cases. Il s'agit de couples `(i,j)` où `i` et `j` correspondent respectivement aux numéros de ligne et de colonne des cases successivement visitées au long du parcours. Ainsi, la liste suivante
`[(1,4),(1,5),(1,6),(2,6),(3,6),(3,5),(3,4)]`
correspond au parcours repéré par des étoiles ci-contre :



La liste `[(0,0),(1,0),(1,1),(5,1),(6,1)]` ne peut correspondre au parcours d'un labyrinthe car toutes les cases parcourues successivement ne sont pas adjacentes.

2. On considère la fonction voisine ci-dessous, écrite en langage Python, qui prend en paramètres deux cases données sous forme de couple.

```
def voisine(case1, case2) :
    l1, c1 = case1
    l2, c2 = case2
    d = (l1-l2)**2 + (c1-c2)**2
    return (d == 1)
```

   a) Après avoir remarqué que les quantités `l1-l2` et `c1-c2` sont des entiers, expliquer pourquoi la fonction voisine indique si deux cases données sous forme de tuples `(l,c)` sont adjacentes.

   b) En déduire une fonction `adjacentes(l)` qui reçoit une liste de cases `l` et renvoie un booléen indiquant si la liste des cases forme une chaîne de cases adjacentes.

Un parcours sera qualifié de **compatible** avec le labyrinthe lorsqu'il s'agit d'une succession de cases adjacentes accessibles (non murées). On donne la fonction `teste(cases, laby)` qui indique si le chemin `cases` est un chemin possible compatible avec le labyrinthe `laby` :

```
def teste(cases, laby) :
    if not adjacentes(cases) :
        return False
    possible = True
    i = 0
    while i < len(cases) and possible:
        if mur(laby, cases[i][0], cases[i][1]) :
            possible = False
        i = i + 1
    return possible
```

3. Justifier que la boucle de la fonction précédente se termine.

4. En déduire une fonction `echappe(cases, laby)` qui indique par un booléen si le chemin `cases` permet d'aller de l'entrée à la sortie du labyrinthe `laby`.

exercice040

Exercice

Chaque soir, les auditeurs d'une radio votent en ligne pour leur artiste favori. Ces votes sont stockés dans un tableau.

Exemple :

```
urne = ['A', 'A', 'A', 'B', 'C', 'B', 'C', 'B', 'C', 'B']
```

La fonction depouille doit permettre de compter le nombre de votes exprimés pour chaque artiste. Elle prend en paramètre un tableau et renvoie le résultat dans un dictionnaire dont les clés sont les noms des artistes et les valeurs le nombre de votes en leur faveur.

La fonction vainqueur doit désigner le nom du ou des gagnants. Elle prend en paramètre un dictionnaire dont la structure est celle du dictionnaire renvoyé par la fonction depouille et renvoie un tableau. Ce tableau peut donc contenir plusieurs éléments s'il y a des artistes ex- aequo. Compléter les fonctions depouille et vainqueur ci-après pour qu'elles renvoient les résultats attendus.

```
urne = ['A', 'A', 'A','B', 'C', 'B', 'C','B', 'C', 'B']

def depouille(urne):
    resultat = ...
    for bulletin in urne:
        if ...:
            resultat[bulletin] = resultat[bulletin] + 1
        else:
            ...
    return resultat

def vainqueur(election):
    vainqueur = ''
    nmax = 0
    for candidat in election:
        if ... > ... :
```

```
            nmax = ...
            vainqueur = candidat
    liste_finale = [nom for nom in election if election[nom] == ...]
    return ...
```

Exemples d'utilisation :

```
>>> election = depouille(urne)
>>> election
{'A': 3, 'B': 4, 'C': 3}
>>> vainqueur(election)
['B']
```

exercice041

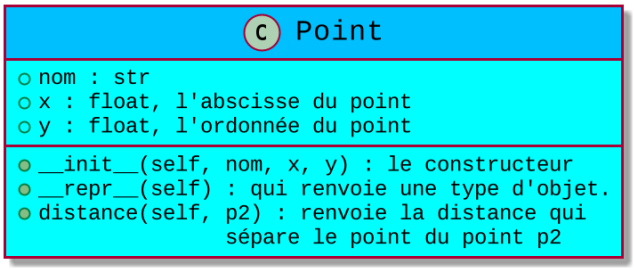
Exercice

L'objectif est d'implémenter une classe `Point` et une fonction `milieu()` qui pourraient être utilisées pour développer un logiciel de géométrie.

Nous aurons besoin de la racine carrée pour calculer des longueurs.

```
>>> from math import sqrt
```

1. Implémenter la classe `Point` correspondant au schéma suivant :




```
>>> A = Point("A", 1, 2)
>>> A.nom
'A'
>>> A.x
1
>>> A.y
2
```

On rappelle que la dunder méthode `__repr__` renvoie un texte qui décrit l'objet. Ici, on souhaite obtenir le résultat suivant (attention aux espaces !) :

```
>>> A
Point A(1, 2)
```

On rappelle que la formule de la distance séparant les points A et B est :

$$AB = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

```
>>> B = Point("B", 4, 6)
>>> A.distance(B)
5.0
>>> B.distance(A)
5.0
```

2. Écrire une fonction `milieu(p1, p2, nom)` qui prend comme paramètres deux points `p1` et `p2` et qui renvoie un nouveau point appelé `nom`, milieu du segment `[p1p2]`.

On rappelle que le milieu I du segment AB a pour coordonnées $x_I = \dfrac{x_A + x_B}{2}$ et $y_I = \dfrac{y_A + y_B}{2}$.

```
>>> C = milieu(A, B, "C")
>>> C
Point C(2.5, 4)
```

exercice042

Exercice

L'objet de ce TP est d'écrire un programme qui permet de prévoir si un texte est écrit en français ou en anglais.

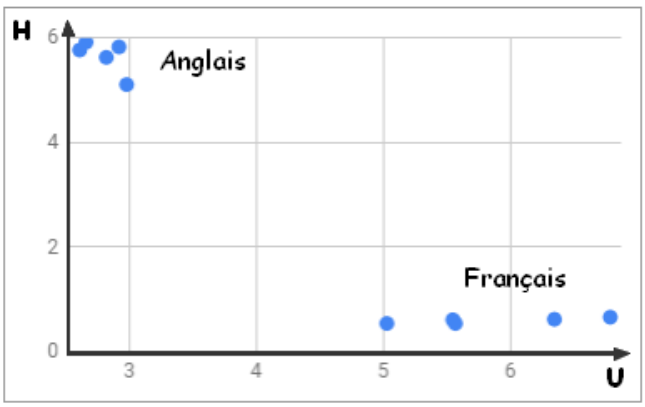
Voici les statistiques concernant la fréquence d'apparition en pourcentage des lettres *U* et *H*, dans 10 textes écrits soit en anglais, soit en français.

Texte	U	H	Langue
1	5.61	0.5	"F"
2	2.99	5.09	"A"
3	5.28	0.52	"F"
4	2.57	5.87	"A"
5	6.91	0.59	"F"
6	2.68	5.56	"A"
7	5.06	0.63	"F"
8	2.91	5.28	"A"
9	3.09	4.90	"A"
10	5.75	0.6	"F"

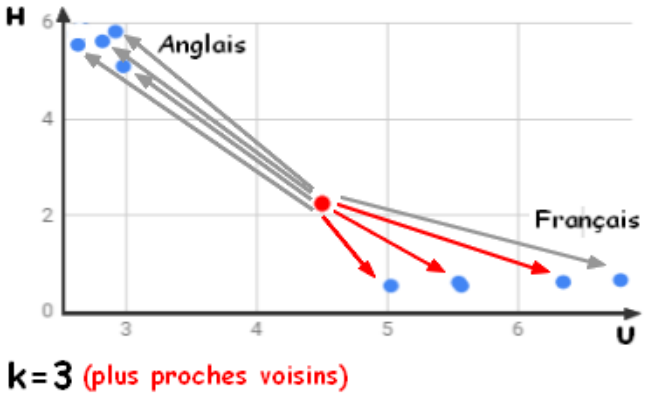
Ce tableau sera stocké dans la variable `STATS` (sans la première ligne contenant le nom des colonnes).

```
>>> STATS = [[1, 5.61, 0.5, "F"],
...          [2, 2.99, 5.09, "A"],
...          [3, 5.28, 0.52, "F"],
...          [4, 2.57, 5.87, "A"],
...          [5, 6.91, 0.59, "F"],
...          [6, 2.68, 5.56, "A"],
...          [7, 5.06, 0.63, "F"],
...          [8, 2.91, 5.28, "A"],
...          [9, 3.09, 4.9, "A"],
...          [10, 5.75, 0.6, "F"]]
```

La représentation graphique suivante avec en abscisse la fréquence du *U* et en ordonnée la fréquence du *H* montre que ces deux lettres sont assez discriminantes.



L'objectif du reste du TP est d'établir la fréquence de U et de H dans le texte dont on veut prédire la langue d'écriture pour pouvoir chercher quels sont les k plus proches voisins parmi les 10 textes dont on connait les statistiques.



1. Écrire une fonction `distance(x_1, y_1, x_2, y_2)` qui calcule la distance euclidienne entre les points de coordonnées `(x_1, y_1)` et `(x_2, y_2)`.

   On rappelle que la formule de la distance séparant les points A et B est :

   $$AB = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

   ```
   >>> distance(1, 1, 4, 5)
   5.0
   >>> distance(1, 1, 1, 0)
   1.0
   ```

2. Écrire une fonction `frequence_lettre(lettre, texte)` qui renvoie la fréquence d'apparition de la `lettre` dans le `texte`. La fréquence renvoyée doit être en pourcentage.

   ```
   >>> t = "J'adore jouer avec mon ballon !"
   >>> isclose(frequence_lettre('a', t), 9.67741935483871, rel_tol=1e-10)
   True
   >>> frequence_lettre('y', t)
   0.0
   >>> isclose(frequence_lettre('j', t), 6.451612903225806, rel_tol=1e-10)
   True
   ```

3. Écrire une fonction `liste_distances(x, y, liste_donnees)` qui renvoie la liste des distances séparant le point `(x ; y)` de tous les points présents dans la `liste_donnees`.

   ```
   >>> liste_distances(0, 0, STATS)
       [[1, 5.632237566012287, 'F'], [2, 5.90323640048406, 'A'],
         ..., [10, 5.7812195945146385, 'F']]
   ```

4. Écrire une fonction `prediction(liste, k)` qui renvoie la valeur de l'étiquette la plus présente parmi les k premières valeurs de la `liste`. *La liste doit être triée.*

   Ici l'étiquette vaut `'A'`, `'F'` ou `'?'` si on ne peut savoir.

```
>>> liste_triee = [[1, 1, 'F'],
...                [2, 2,'F'],
...                [3, 3, 'A'],
...                [4, 4, 'A'],
...                [5, 5, 'F']]
>>> prediction(liste_triee, 1)
'F'
>>> prediction(liste_triee, 2)
'F'
>>> prediction(liste_triee, 4)
'?'
```

5. Écrire une fonction `predire_langue(texte, k)` qui tente de prédire la langue dans laquelle est écrit le `texte` en se basant sur les k plus proches voisins. La fonction doit renvoyer l'une des phrases suivantes :

   - « Je pense que le texte est écrit en français. »
   - « Je pense que le texte est écrit en anglais. »
   - « Je ne sais pas dans quelle langue est écrit le texte. »

   Petit coup de pouce

   `sorted(ld, key=lambda valeur: valeur[1])` renvoie une copie triée dans l'ordre croissant de la liste de listes `ld` suivant les valeurs de la deuxième colonne.

```
>>> T = ("Earlier releases have been removed because"
...      "we can only support the current versions of the software. "
...      "To update old code, read the changes page. "
...      "Changes for each release can be found in revisions.txt. "
...      "If you have problems with the current release, please file a bug "
...      "so that we can fix it. Older releases can also be built "
...      "from the source. Read More about the releases and their numbering. "
...      "To use Android Mode, Processing 3 or later is required.")
>>> predire_langue(T, 1)
'Je pense que le texte est écrit en anglais.'
>>> predire_langue(T, 8)
'Je pense que le texte est écrit en anglais.'
>>> predire_langue(T, 10)
'Je ne sais pas dans quelle langue est écrit le texte.'
```

   Peut-on considérer l'algorithme des k plus proches voisins comme fiable ?

```
>>> T = "L'hyperbole est un truc de hippie !"
>>> predire_langue(T, 9)
'Je pense que le texte est écrit en anglais.'
>>> predire_langue(T, 1)
'Je pense que le texte est écrit en anglais.'
```

exercice043

Exercice

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

L'algorithme traite le cas du tableau vide.

L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.

On distingue les trois cas qui renvoient `False` en renvoyant `False, 1`, `False, 2` et `False, 3`.

Compléter l'algorithme de dichotomie donné ci-après.

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if ...................:
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or .................:
        return False, 2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ........
        if x == tab[m]:
            return ..........
        if x > tab[m]:
            debut = m + 1
        else:
```

```
            fin = ..........
    return ............
```

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],27)
(False, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)
```

exercice044

Exercice

La fonction `tri_insertion` suivante prend en argument une liste `L` et trie cette liste en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

```
def tri_insertion(L):
    n = len(L)

    # cas du tableau vide
    if ......................:
        return L
    for j in range(1,n):
        e = L[j]
        i = j

        # A l'étape j, le sous-tableau L[0,j-1] est trié
        # et on insère L[j] dans ce sous-tableau en déterminant
        # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].

        while i > 0 and L[i-1] > .................:
            i = ...................

        # si i != j, on décale le sous tableau L[i,j-1] d'un cran
        # vers la droite et on place L[j] en position i

        if i != j:
            for k in range(j, i, ......):
                L[k] = L[......]
            L[i] = ......
    return L
```

Exemples :

```
>>> tri_insertion([2, 5, -1, 7, 0, 28])
[-1, 0, 2, 5, 7, 28]

>>> tri_insertion([10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

exercice045

Exercice

Dans cet exercice, on étudie une méthode de chiffrement de chaines de caractères alphabétiques. Pour des raisons historiques, cette méthode de chiffrement est appelée « code de César ». On considère que les messages ne contiennent que les lettres capitales de l'alphabet `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"` et la méthode de chiffrement utilise un nombre entier fixé appelé la clé de chiffrement.

1. Soit la classe `CodeCesar` définie ci-dessous :

```
class CodeCesar:
    def __init__(self, cle):
        self.cle = cle

    def decale(self, lettre):
        indice_1 = indice_capitale(lettre)
        indice_2 = indice_1 + self.cle
        if indice_2 >= 26:
            indice_2 = indice_2 - 26
        if indice_2 < 0:
            indice_2 = indice_2 + 26
        nouvelle_lettre = lettre_capitale(indice_2)
        return nouvelle_lettre
```

On dispose aussi des fonctions `indice_capitale(indice)` et `lettre_capitale(lettre)` qui renvoient la lettre majuscule correspondant à un indice donné et l'indice (la position d'une lettre) dans l'alphabet latin usuel.
Représenter le résultat d'exécution du code Python suivant :

```
>>> code = CodeCesar(3)
>>> code.decale('A')
...
>>> code.decale('X')
...
```

2. La méthode de chiffrement du « code de César » consiste à décaler les lettres du message dans l'alphabet d'un nombre de rangs fixé par la clé. Par exemple, avec la clé 3, toutes les lettres sont décalées de 3 rangs vers la droite : le A devient le D, le B devient le E, etc.
   Ajouter une méthode `chiffre(self, texte)` dans la classe `CodeCesar` définie à la question précédente, qui reçoit en paramètre une chaîne de caractères (le message à chiffrer) et qui renvoie une chaîne de caractères (le message chiffré).
   Cette méthode `chiffre(self, texte)` doit chiffrer la chaîne `texte` avec la clé de l'objet de la classe `CodeCesar` qui a été instanciée.
   Exemple :

   ```
   >>> code = CodeCesar(3)
   >>> code.chiffre("NSI")
   'QVL'
   ```

3. Écrire une fonction `chiffre_texte(clef, texte)` qui :

   - prend en argument la clef de chiffrement et le message à chiffrer ;
   - instancie un objet de la classe `CodeCesar` ;
   - renvoie le texte chiffré.

4. On ajoute la méthode `transforme(texte)` à la classe `CodeCesar` :

   ```
   def transforme(self, texte):
       self.cle = -self.cle
       message = self.cryptage(texte)
       self.cle = -self.cle
       return message
   ```

   On exécute la ligne suivante dans une console
   `CodeCesar(10).transforme("PSX")`.
   Que va-t-il s'afficher ? Expliquer votre réponse.

exercice046

Exercice

La fonction `tri_bulles` prend en paramètre une liste `T` d'entiers non triés et renvoie la liste triée par ordre croissant. Compléter le code Python ci-dessous qui implémente la fonction `tri_bulles`.

```
def tri_bulles(T):
    n = len(T)
    for i in range(...,...,-1):
        for j in range(i):
            if T[j] > T[...]:
                ... = T[j]
                T[j] = T[...]
                T[j+1] = temp
    return T
```

exercice047

Exercice

On considère un tableau de nombres de n lignes et p colonnes.

Les lignes sont numérotées de 0 à $n-1$ et les colonnes sont numérotées de 0 à $p-1$. La case en haut à gauche est repérée par $(0,0)$ et la case en bas à droite par $(n-1,p-1)$. On appelle chemin une succession de cases allant de la case $(0,0)$ à la case $(n-1,p-1)$, en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.

On appelle somme d'un chemin la somme des entiers situés sur ce chemin.

Par exemple, pour le tableau T suivant :

4	**1**	**1**	3
2	0	**2**	1
3	1	**5**	**1**

- Un chemin est $(0,0),(0,1),(0,2),(1,2),(2,2),(2,3)$ (en gras sur le tableau) ;
- La somme du chemin précédent est 14.
- $(0,0),(0,2),(2,2),(2,3)$ n'est pas un chemin.

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case $(0,0)$ à la case $(n-1,p-1)$.

1. On considère tous les chemins allant de la case $(0,0)$ à la case $(2,3)$ du tableau `T` donné en exemple.
   a. Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il ?
   b. La longueur d'un chemin est égal au nombre de cases de ce chemin. Justifier que tous les chemins allant de $(0,0)$ à $(2,3)$ ont une longueur égale à 6.
2. En listant tous les chemins possibles allant de $(0,0)$ à $(2,3)$ du tableau `T`, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.
3. On veut créer le tableau `T'` où chaque élément `T'[i][j]` est la somme maximale pour tous les chemins possibles allant de $(0,0)$ à (i,j).

a. Compléter et recopier sur votre copie le tableau `T'` donné ci-dessous associé au tableau

`T =`

4	1	1	3
2	0	2	1
3	1	5	1

`T' =`

4	5	6	?
6	?	8	10
9	10	?	16

b. Justifier que si j est différent de 0, alors `T'[0][j] = T[0][j] + T'[0][j-1]`.

4. Justifier que si i et j sont différents de 0, alors :

   `T'[i][j] = T[i][j] + max(T'[i-1][j], T'[i][j-1])`

5. On veut créer la fonction récursive `somme_max` ayant pour paramètres un tableau `T`, un entier `i` et un entier `j`. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case $(0,0)$ à la case (i,j).

   a) Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction `somme_max` ? Que renvoie-t-on dans ce cas ?
   b) À l'aide de la question précédente, écrire en Python la fonction récursive `somme_max`.
   c) Quel appel de fonction doit-on faire pour résoudre le problème initial ?

exercice048

Exercice

Une coopérative de livres distribue des manuels scolaires aux élèves d'un lycée. Pour gérer ses stocks, elle utilise une base de données. Nous utiliserons un extrait de cette base composée uniquement de 2 tables :

- `collections` avec les attributs :
  - `isbn` : un nombre (`INTEGER`) unique qui servira de clé primaire ;
  - `editeur` : le nom de l'éditeur du manuel (`TEXT`) ;
  - `titre` : le titre de l'ouvrage (`TEXT`, pas toujours original) ;
  - `dateachat` : la date d'achat du manuel au format `TEXT` ;
  - `refmatiere` : un entier (`INTEGER`) qui est une clé étrangère vers la table `matieres` ;
  - `refniveau` : le niveau dans lequel est utilisé le livre au format `TEXT` ;
  - `etat` : `TEXT` qui décrit l'état d'une collection. Elle peut être *active* si elle est utilisée ou *obsolète* si elle ne l'est plus.
- `matieres` avec les attributs :
  - `refmatiere` : un nombre (`INTEGER`), clé primaire de la table ;
  - `nom` : le nom de la dicipline qui utilise le manuel (`TEXT`).

Tout est contenu dans le fichier `cooperative.sqlite` que vous pourrez parcourir avec le logiciel `DB Browser for Sqlite`. Vous pourrez y écrire vos requètes pour répondre aux questions posées.

Une fois que vous pensez que votre requète est correcte, vous la collerez dans la variable texte qui vous est proposée dans la cellule. **Attention**,

- les attributs doivent impérativement être **dans l'ordre de la question** ;
- n'utiliser que **des simples quotes '** et non des " ;
- surtout vous ne mettez **pas de point-virgule**.
- **Ne changez pas le nom de la variable !**

Question 1 : quels sont l'isbn, l'éditeur et le titre des collections du niveau Terminale ?
Placer votre requète dans la variable `prop_Q1`.
Question 2 : quels sont l'isbn, l'éditeur et le titre des collections obsolètes du niveau Terminale ?
Placer votre requète dans la variable `prop_Q2`.
Question 3 : quels sont les éditeurs ayant fourni des manuels au lycée ?
Placer votre requète dans la variable `prop_Q3`.
Question 4 : quels sont les isbn et les titres des collections d'espagnol ?
Placer votre requète dans la variable `prop_Q4`.
Question 5 : quels sont les éditeurs ayant fourni les collections d'histoire ?
Placer votre requète dans la variable `prop_Q5`.
Question 6 : dans quelles matières utilise t-on les livres de l'éditeur *Nathan* ?
Placer votre requète dans la variable `prop_Q6`.
Question 7 : combien chaque éditeur fournit-il de collections ?
Le résultat devra renvoyer l'éditeur et la quantité de collections fournies. Cette dernière colonne devra s'appelée *effectif*. Les résultats seront classés dans l'ordre décroissant, c'est-à-dire que l'éditeur qui fournit le plus de collections doit apparaître en premier. Placer votre requète dans la variable `prop_Q7`.
Question 8 : combien l'éditeur *Hatier* fournit-il de collections ? Le résultat sera donné dans une colonne qui s'appelle `effectif`. Placer votre requète dans la variable `prop_Q8`.
Question 9 : Quels sont les isbn, les éditeurs et les titres des collections achetées en *2005* ?
Ici la date est fournie sous forme de texte. Pour extraire l'année, je vous invite à utiliser la fonction *substr*.
Cette fonction peut s'utiliser dans une requête `SQL` en utilisant une syntaxe comme celle-ci :

```
SELECT SUBSTR(nom_colonne, 3, 10) FROM tableau
```

Dans cet exemple, le contenu de la colonne `nom_colonne` sera tronqué à partir du 4ème caractère sur 10 caractères.
Placer votre requète dans la variable `prop_Q9`.
Question 10 : quelles sont les matières correspondant à un enseignement de *spécialité* ?
Vous pouvez utiliser le mot clé `LIKE`.

- `LIKE '%a'` : le caractère "%" est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un "a".
- `LIKE 'a%'` : ce modèle permet de rechercher toutes les lignes de "colonne" qui commence par un "a".
- `LIKE '%a%'` : ce modèle est utilisé pour rechercher tous les enregistrement qui utilisent le caractère "a".
- `LIKE 'pa%on'` : ce modèle permet de rechercher les chaines qui commence par "pa" et qui se terminent par "on", comme "pantalon" ou "pardon".

Placer votre requète dans la variable `prop_Q10`.

exercice049

Exercice

Soit `T` un tableau non vide d'entiers triés dans l'ordre croissant et `n` un entier. La fonction `chercher`, donnée ci-dessous, doit renvoyer un indice où la valeur `n` apparaît éventuellement dans `T`, et `None` sinon.

Les paramètres de la fonction sont :

- `T`, le tableau dans lequel s'effectue la recherche ;
- `n`, l'entier à chercher dans le tableau ;
- `i`, l'indice de début de la partie du tableau où s'effectue la recherche ;
- `j`, l'indice de fin de la partie du tableau où s'effectue la recherche.

La fonction `chercher` est une fonction récursive basée sur le principe « diviser pour régner ».

Le code de la fonction commence par vérifier que les paramètres `i` et `j` sont corrects. Si ce n'est pas le cas, cela déclenche une `AssertionError`.

Recopier et compléter le code de la fonction `chercher` proposée ci-dessous :

```
def chercher(T, n, i, j):
    ???
    if i > j :
        return ???
    m = ???
    if T[m] < ??? :
        return chercher(T, n, ??? , ???)
    elif ??? :
        return chercher(T, n, ??? , ??? )
    else :
        return ???
```

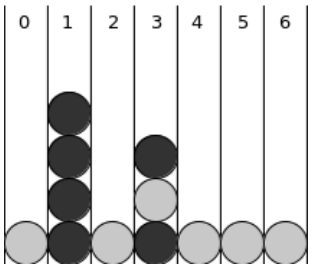
```
>>> chercher([1, 5, 6, 6, 9, 12], 7, 0, 10)
Traceback (most recent call last):
...
AssertionError
```

```
>>> chercher([1, 5, 6, 6, 9, 12], 7, 0, 5)
>>> chercher([1, 5, 6, 6, 9, 12], 9, 0, 5)
4
>>> chercher([1, 5, 6, 6, 9, 12], 6, 0, 5)
2
```

exercice050

Exercice

On souhaite développer un jeu de puissance 4. Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes.



La grille correspondant à la zone de jeu sera implémentée par une liste de listes. Les espaces vides seront représentés par des 0. Les jetons du joueur seront représentés par de 1 et ceux de l'ordinateur par des 2. Ainsi, la situation de l'image précédente est représentée par la liste suivante :

```
>>> g = [[0, 0, 0, 0, 0, 0, 0],
...      [0, 0, 0, 0, 0, 0, 0],
...      [0, 1, 0, 0, 0, 0, 0],
...      [0, 1, 0, 1, 0, 0, 0],
...      [0, 1, 0, 2, 0, 0, 0],
...      [2, 1, 2, 1, 2, 2, 2]]
```

1. Écrire une fonction `derniere_ligne_vide(t, colonne)` qui renvoie l'indice de la première ligne vide dans la `colonne`. Si la colonne est pleine, la fonction doit renvoyer -1.

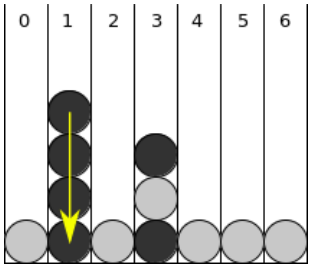
   ```
   >>> derniere_ligne_vide(g, 0)
   4
   >>> derniere_ligne_vide(g, 1)
   1
   ```

2. Écrire une fonction `ajouter_pion_colonne(t, colonne, joueur)` qui ajoute, si possible, le pion du `joueur` à la `colonne` (un entier compris entre 0 et 6) de la grille `t`. La fonction renverra la ligne à laquelle a été ajouté le pion. Si la colonne est pleine, la fonction renvoie -1.

```
>>> g[4][0]
0
>>> ajouter_pion_colonne(g, 0, 2)
4
>>> g[4][0]
2

>>> ajouter_pion_colonne(g, 1, 2)
1
>>> g[1][1]
2
>>> ajouter_pion_colonne(g, 1, 1)
0
>>> g[0][1]
1
>>> ajouter_pion_colonne(g, 1, 2)
-1
```

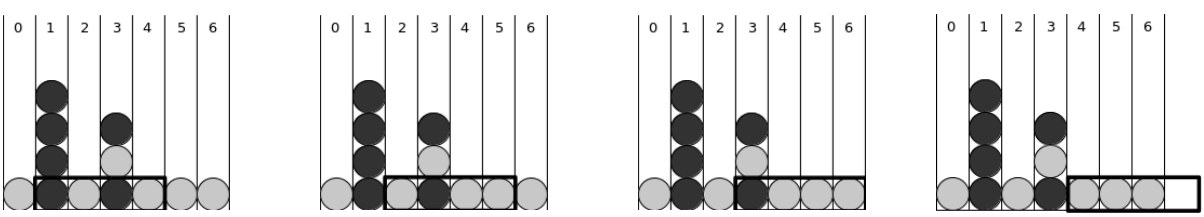
3. Une fois que le joueur a joué, il faut vérifier si ce jeton crée un alignement de 4 jetons.
   Écrire une fonction `alignement_vertical(t, ligne, colonne, joueur)` qui teste si le `joueur` a réussi un alignement vertical avec le jeton qu'il vient de placer dans la grille `t` à l'emplacement `(colonne,ligne)`. La fonction doit renvoyer un booléen.



   Dans ce cas précis, on doit avoir :

```
>>> alignement_vertical(g, 2, 1, 1)
True
```

4. Écrire une fonction `alignement_horizontal(t, ligne, colonne, joueur)` qui teste si le `joueur` a réussi un alignement horizontal avec le jeton qu'il vient de placer dans la grille `t` à l'emplacement `(colonne,ligne)`. La fonction doit renvoyer un booléen.
   Si le dernier jeton ajouté est celui de la colonne 4, voila les situations qu'il faut tester.



Dans ce cas précis,

```
>>> alignement_horizontal(g, 5, 4, 2)
False
```

exercice051

Exercice

Cet exercice est composé de 3 parties indépendantes. La partie 1 porte sur quelques concepts fondamentaux sur les arbres binaires, la partie 2 s'intéresse à la distance entre deux nœuds d'un arbre binaire et la partie 3 présente les arbre_somme.

Partie 1

On implémente les arbres binaires en utilisant la programmation objet avec la classe Nœud ci-dessous :

```
class Noeud:
    def __init__(self, racine=None):
        self.r = racine
        self.gauche = None
        self.droit = None
```

1. Combien d'attributs possèdent la classe Nœud ?

2. Représenter l'arbre obtenu avec les instructions suivantes :

```
>>> a = Noeud(4)
>>> a.gauche = Noeud(5)
>>> a.droit = Noeud(14)
>>> a.gauche.droit = Noeud(8)
```

3. On considère l'arbre de la Figure 1 :

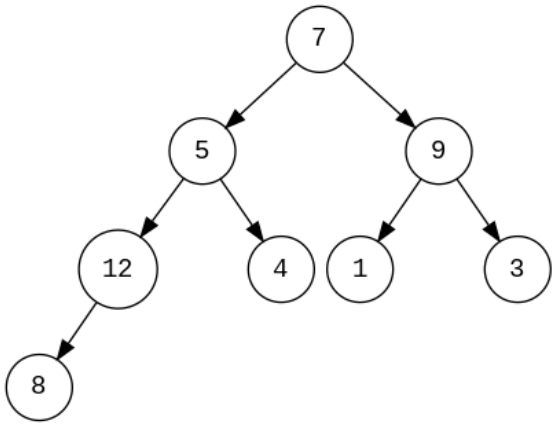


Figure 1

a) Quelle est la profondeur du nœud d'étiquette 12 ?

b) Quelle est la taille de l'arbre ?

c) Dans quel ordre sont parcourus les nœuds lors d'un parcours préfixe en profondeur ? On visite le nœud gauche avant le nœud droit.

Partie 2

Dans toute cette partie, on prendra pour exemple l'arbre de la Figure 1.

On appelle distance entre deux nœuds, le nombre minimal de branches qui sépare les deux nœuds. On notera d la distance.

Ainsi, pour l'arbre de la Figure 1, on a `d(12, 4) = 2` et `d(8, 9) = 4`.

1. Quelle est la distance entre les nœuds d'étiquette 5 et d'étiquette 1 ?

2. Soient `n1` et `n2`, deux nœuds d'un arbre binaire, on admettra la relation suivante :

$$d(n1,n2) = d(n1,r) + d(n2,r) - 2 \times d(r, ppac)$$

où `r` représente la racine de l'arbre et `ppac` le plus petit ancêtre commun aux nœuds `n1` et `n2`.

a) Écrire une fonction `contient(arbre, x)` qui renvoie `True` si le nœud de valeur `x` est dans l'arbre, `False` sinon.

b) Compléter une fonction `chemin(arbre, n)` qui renvoie la liste des nœuds compris entre la racine et le nœud compris.

```
def chemin(arbre, n):
    """
    Renvoie la liste des nœuds compris entre la racine et le nœud compris.
    """
    if arbre is None:
        return .....
```

```
    else:
        if arbre.r == n:
            return [n]
        elif contient(arbre.gauche, n):
            return [arbre.r] + ..............
        elif ......................:
            return ..........................
        else:
            return []
```

c) Pour trouver le plus petit ancêtre commun à deux nœuds `n1` et `n2`, on procède de la manière suivante :

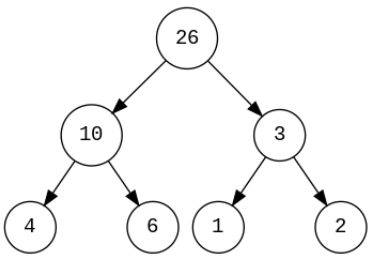
- On détermine le chemin entre la racine et le nœud `n1` ;
- On détermine le chemin entre la racine le le nœud `n2` ;
- On compare les deux listes ci-dessus et le premier élément différent entre les deux listes est l'élément qui suit le plus petit ancêtre commun. Écrire une fonction `ppac(arbre, n1,n2)` qui renvoie le plus petit ancêtre commun aux nœud `n1` et `n2`.

3. Déduire des fonctions précédentes, une fonction `distance(arbre,n1,n2)` qui renvoie la distance entre deux nœuds.

   (Remarque : On aura remarqué que la longueur de la liste du chemin entre la racine et le nœud renvoie le nombre de nœud et non la distance. On admettra que la formule donnée en début de partie appliquée aux chemins renvoie effectivement la distance)

Partie 3

On appelle `arbre_somme` un arbre binaire dont la valeur d'un nœud est égale à la somme des nœuds de ses sous-arbres gauche et droit.
Ainsi, l'arbre ci-contre est un `arbre_somme`.
En effet, $26 = (10+6+4)+(3+2+1)$ et $10 = 4+6$ et $3 = 1+2$.
On se propose dans cette partie de déterminer si un arbre binaire est un `arbre_somme`



1. Écrire une fonction `somme(arbre)` qui renvoie la somme des valeurs des nœuds de l'arbre.

2. Compléter la fonction `est_arbre_somme(arbre)` qui renvoie un booléen.

```
def est_arbre_somme(arbre):
    """
    Renvoie True sur arbre est un arbre_somme, False sinon.
    """
    if arbre is None:
```

```
        return True
    elif .....................: # Tester si le noeud est une feuille
        return True
    else:
        return................ and ................. and  ..................
```

exercice052

Exercice

Cet exercice porte sur les bases de données, les fichiers csv et tableaux de dictionnaires.

Martin veut faire des économies dans sa vie de tous les jours en réduisant notamment ses dépenses en énergie. En particulier, il cherche à réduire ses factures d'électricité. Pour cela, il va utiliser ses connaissances en informatique pour obtenir des informations lui permettant de faire les économies souhaitées. Il est également par ailleurs très impliqué dans la lutte contre le réchauffement climatique et a décidé de créer une base de données pour étudier plus profondément ce phénomène. On étudie dans cet exercice séparément dans deux parties distinctes les deux intérêts de Martin. Les parties A et B sont donc indépendantes.

Partie A

Martin a récupéré sur internet des données d'un centre météorologique et a créé deux relations (tables). La relation Centres contient l'identifiant des centres météorologiques, la ville, la latitude, la longitude et l'altitude du centre. La relation Mesures contient l'identifiant de la mesure, l'identifiant du centre, la date de la mesure, la température, la pression et la pluviométrie mesurées.

Le schéma relationnel de la relation `centres` est le suivant :

`Centres(id_centre: INT, nom_ville: VARCHAR, latitude: FLOAT, longitude: FLOAT, altitude: INT)`

Le schéma relationnel de la relation `mesures` est le suivant :

`Mesures(id_mesure: INT, id_centre: INT, date_mesure: DATE, temperature: FLOAT, pression: INT, pluviometrie: INT)`

On fournit ci-dessous le contenu des deux relations.

Relation centres

id_centre	nom_ville	latitude	longitude	altitude
213	'Amiens'	49.894	2.293	60
138	'Grenoble'	45.185	5.723	550
263	'Brest'	48.388	-4.49	52

id_centre	nom_ville	latitude	longitude	altitude
185	'Tignes'	45.469	6.909	2594
459	'Nice'	43.706	7.262	260
126	'Le Puy-en-Velay'	45.042	3.888	744
317	'Gérardmer'	48.073	6.879	855

Relation mesures

id_mesure	id_centre	date_mesure	temperature	pression	pluviometrie
1566	138	'2021-10-29'	8.0	1015	3
1568	213	'2021-10-29'	15.1	1011	0
2174	126	'2021-10-30'	18.2	1023	0
2200	185	'2021-10-30'	5.6	989	20
2232	459	'2021-10-31'	25.0	1035	0
2514	213	'2021-10-31'	17.4	1020	0
2563	126	'2021-11-01'	10.1	1005	15
2592	459	'2021-11-01'	23.3	1028	2
3425	317	'2021-11-02'	9.0	1012	13
3430	138	'2021-11-02'	7.5	996	16
3611	263	'2021-11-03'	13.9	1005	8
3625	126	'2021-11-03'	10.8	1008	8

1. a) Proposer une clé primaire pour la relation `Mesures`. Justifier votre choix.

   b) Avec quel attribut peut-on faire une jointure entre la relation `Centres` et la relation `Mesures`?

2. a) Qu'affiche la requête suivante `SELECT nom_ville FROM Centres WHERE altitude > 500;`

   b) On souhaite récupérer le nom de la ville des centres météorologiques situés à une altitude comprise entre 700 m et 1200 m, inclus. Écrire la requête SQL correspondante.

   c) On souhaite récupérer la liste des longitudes et des noms des villes des centres météorologiques dont la longitude est supérieure à 5.0 La liste devra être triée par ordre alphabétique des noms de ville. Écrire la requête SQL correspondante.

3. a) Qu'affiche la requête suivante?

   ```
   SELECT * FROM Mesures WHERE date_mesure = '2021-10-30';
   ```

   b) Écrire une requête SQL permettant d'ajouter une mesure prise le 8 novembre 2021 dans le centre numéro 138, où la température était de 11°C, la pression de 1013 hPa et la pluviométrie de 0 mm. La donnée dont l'attribut est `id_mesure` aura pour valeur 3650.

c) Écrire une requête SQL donnant la température moyenne de toutes les mesures effectuées dans la relation `Mesures`.

4. Expliquer ce que renvoie la requête SQL suivante :

```
SELECT * FROM Centres WHERE latitude = (SELECT MIN(latitude) FROM Centres);
```

5. Écrire une requête SQL donnant la liste des villes (leur nom) dans lesquelles on a enregistré une pluviométrie supérieure ou égal à 10mm. On utilisera le mot clé DISTINCT afin d'éviter d'avoir des doublons. On rappelle que l'on peut utiliser les opérateurs de comparaison avec les dates.

Partie B

Dans cette partie, on s'intéresse à la consommation en électricité de Martin. La consommation électrique d'un appareil se mesure en kwH (kilowattheure). Par exemple, le réfrigérateur américain de Martin consomme en moyenne 1,8 kwH chaque jour. En l'espace d'un an, son réfrigérateur aura donc consommé $1,8 \times 365$ soit 657 kwH. Actuellement, au 1er janvier 2023, le prix du kwH pour un logement comme celui de Martin est de 0,1740 € le kwH (TRV : Tarif Réglementé de l'électricité en Vigueur). Si ce prix se maintient pendant 1 an, le réfrigérateur américain de Martin lui coûtera donc $657 \times 0,1740$ soit environ 114€.

Grâce à son compteur électrique communiquant, Martin dispose d'un fichier `consommations_martin.csv` donnant sa consommation en kwH chaque jour de l'année du mercredi 01/01/2020 au mardi 03/01/2023 (voir image ci-dessous). Par exemple, on voit que le mardi 03/01/2023, il a consommé au total 6,8 kwH avec tous ses appareils électriques.

Extrait du fichier `consommations_martin.csv`

	A	B	C	D
1	Jour	Date	Année	Consommation
2	Mardi	03/01	23	6,8
3	Lundi	02/01	23	8,5
4	Dimanche	01/01	23	7,5
5	Samedi	31/12	22	10
6	Vendredi	30/12	22	9,6
7	Jeudi	29/12	22	8
8	Mercredi	28/12	22	8,1
9	Mardi	27/12	22	11,1

Le but de cette partie est d'étudier les données de ce fichier afin de trouver une offre intéressante pour Martin. On décide d'importer les données en Python dans une variable consommations_martin de type liste de dictionnaires représentant toutes les consommations quotidiennes de Martin du 03/01/2023 au 01/01/2020.

```
>>> print(consommations_martin)
[{'Année': 23, 'Consommation': 6.8, 'Date': '03/01', 'Jour': 'Mardi'},
 {'Année': 23, 'Consommation': 8.5, 'Date': '02/01', 'Jour': 'Lundi'},
 {'Année': 23, 'Consommation': 7.5, 'Date': '01/01', 'Jour': 'Dimanche'},
```

```
 {'Année': 22, 'Consommation': 10, 'Date': '31/12', 'Jour': 'Samedi'},
 {'Année': 22, 'Consommation': 9.6, 'Date': '30/12', 'Jour': 'Vendredi'},
 {'Année': 22, 'Consommation': 8, 'Date': '29/12', 'Jour': 'Jeudi'},
 {'Année': 22, 'Consommation': 8.1, 'Date': '28/12', 'Jour': 'Mercredi'},
 {'Année': 22, 'Consommation': 11.1, 'Date': '27/12', 'Jour': 'Mardi'},
...
```

1. Que vaut `consommations_martin[0]` ? Quel est son type ?

2. Écrire une fonction `week_end(donnees_jour)` qui prend en paramètre un dictionnaire représentant une consommation un jour donné et qui renvoie `True` si ce jour est un samedi ou un dimanche et `False`sinon.

   On doit avoir par exemple :

   ```
   >>> week_end({'Année': 23, 'Consommation': '6.8', 'Date': '03/01', 'Jour': 'Mardi'})
   False
   ```

3. On voudrait connaître la date où Martin a le plus consommé. Compléter la fonction suivante `conso_maximale(consommations)` qui prend en paramètre une liste de dictionnaires consommations représentant les consommations d'un particulier et donnant sous forme d'un tuple la date et l'année où la consommation a été la plus forte.

   On doit avoir par exemple, en considérant l'échantillon précédent de la liste `consommations_martin` :

   ```
   >>> conso_maximale(consommations_martin)
   ("19/12", 22)
   ```

   ```
   def conso_maximale(consommations):
       date = None
       année = None
       res_maxi = 0
       for x in ..... :
           if ..... :
               res_maxi = .....
               date = .....
               année = .....
       return (date, année)
   ```

4. Martin aimerait connaître sa consommation totale en 2020 puis en 2021 puis en 2022 pour voir si d'une année à l'autre, il diminue ou augmente sa consommation. Cela lui permettra également de prédire sa consommation en 2023. Écrire une fonction `somme_conso(consommations, année)` qui prend en paramètre un liste de dictionnaires consommations représentant les consommations d'un particulier et un entier année et qui renvoie la somme totale des consommations en kwH lors de cette année.

À titre d'informations, Martin a obtenu les résultats suivants :

```
>>> somme_conso(consommations_martin, 20)
2669.3
>>> somme_conso(consommations_martin, 21)
2920.7
>>> somme_conso(consommations_martin, 22)
3174.7
```

5. Grâce à la question précédente, Martin se rend compte que sa consommation augmente d'une année sur l'autre. Il se rappelle qu'il a acheté plus d'appareils électriques notamment un lave-linge. En regardant les offres des fournisseurs d'énergie, il voit une offre nommée «ZEN WEEK-END» :

   - 0,1366€ le kwH pendant le week-end ;
   - 0,1912€ le kwH les 5 autres jours de la semaine.
     Il se demande si cette offre vaut plus le coup que l'offre à 0,1740€ le kwH tout le temps sachant qu'il est en effet plus présent à la maison le week-end et qu'il pourrait faire tourner ses machines le week-end. Pour cela, il aimerait connaître pour chacun des 7 jours de la semaine sa consommation totale accumulée en 2022 afin de savoir déjà les jours où il consomme le plus.

   Écrire une fonction `consommations_jours(consommations)` qui renvoie un dictionnaire dont les clés sont les jours de la semaine et les valeurs les consommations en kwH accumulées le jour correspondant.

   Par exemple, Martin a obtenu les résultats suivants :

```
>>> consommations_jours(consommations_martin)
{'Dimanche': 524.7,
'Jeudi': 406.9,
'Lundi': 446.59999999999997,
'Mardi': 416.7999999999999,
'Mercredi': 433.2,
'Samedi': 475.5999999999999,
'Vendredi': 470.9000000000002}
```

   Cela signifie qu'il a consommé 524.7 kwH les dimanches de 2022, 406.9 kwH les jeudis de 2022 etc

   … Cela confirme le fait que Martin consomme naturellement plus le week-end. S'il se force à faire ses machines le week-end, il y a des chances que l'offre «ZEN WEEK-END» soit plus intéressante pour lui …

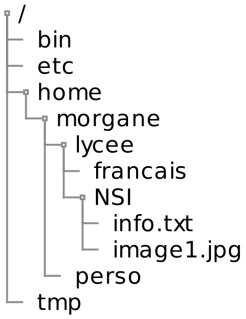
exercice053

Exercice

Partie A

Cet exercice pourra utiliser des commandes de systèmes d'exploitation de type UNIX telles que `cd`, `1s` , `mkdir`, `rm` ,`rmd`, `mv`, `cat`.

Dans un système d'exploitation de type UNIX, on considère l'arborescence des fichiers suivante :



On souhaite, grâce à l'utilisation du terminal de commande, explorer et modifier les répertoires et fichiers présents.

On suppose qu'on se trouve actuellement dans le dossier `/home/morgane`

1. a) Parmi les 4 propositions suivantes, donner celle correspondant à l'affichage obtenu lors de l'utilisation de la commande `ls` :

   - **proposition 1** : `lycee francais NSI info.txt image1.jpg perso`
   - **proposition 2** : `lycee perso`
   - **proposition 3** : `morgane`
   - **proposition 4** : `bin etc home tmp`

   b) Écrire la commande qui permet, à partir de cet emplacement, d'atteindre le répertoire `lycee`.

On suppose maintenant qu'on se trouve dans le répertoire `/home/morgane/lycee/NSI`.

3. a) Écrire la commande qui permet de créer, à cet emplacement, le dossier nommé `algorithmique`.

   b) Écrire la commande qui permet, à partir de cet emplacement, de supprimer le fichier `image1.jpg`.

Partie B

On rappelle qu'un processus est une instance d'application. Un processus peut être démarré par l'utilisateur, par un périphérique ou par un autre processus parent.

La commande UNIX `ps` présente un cliché instantané des processus en cours d'exécution.

```
UID          PID  PPID  C STIME TTY          TIME CMD
test         900   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfs-udisks2-vol
test         907   838  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-trash --sp
test         913   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-metadata
test         918   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test         919   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test         923     1  0 10:51 ?        00:00:02 xfce4-terminal
test         927   923  0 10:51 pts/0    00:00:00 bash
test        1036     1  0 11:18 ?        00:00:02 mousepad /home/test/Documents/
test        1058   923  0 11:22 pts/1    00:00:00 bash
root        1132     2  0 11:37 ?        00:00:00 [kworker/0:0-ata_sff]
root        1134     2  0 11:43 ?        00:00:00 [kworker/0:2-ata_sff]
test        1140   739  0 11:43 ?        00:00:00 /usr/lib/x86_64-linux-gnu/tumb
root        1149     2  0 11:43 ?        00:00:00 [kworker/u2:0-events_unbound]
test        1153   927  0 11:44 pts/0    00:00:00 vi
test        1154   927  0 11:44 pts/0    00:00:00 python3 prog.py
test        1155  1058  0 11:44 pts/1    00:00:00 ps -aef
```

On a exécuté la commande `ps` (avec quelques options qu'il n'est pas nécessaire de connaître pour la réussite de cet exercice). Un extrait du résultat de la commande est présenté ci-dessous :

On rappelle :

- l'`UID` est l'identifiant de l'utilisateur propriétaire du processus ;
- le `PID` est l'identifiant du processus ;
- le `PPID` est l'identifiant du processus parent ;
- `C` indique l'utilisation du processus ;
- `STIME` est l'heure de démarrage du processus ;
- `TTY` est le nom du terminal de commande auquel le processus est attaché ;
- `TIME` est la durée d'utilisation du processeur par le processus ;
- `CMD` le nom de commande utilisé pour démarrer le processus.

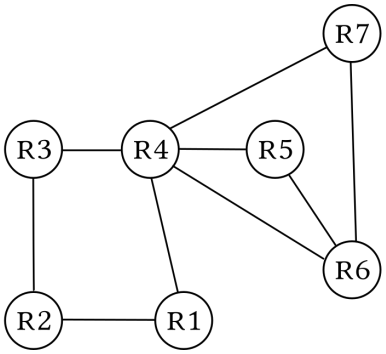
1. Donner le `PID` du parent du processus démarré par la commande `vi`.
2. Donner le `PID` d'un processus enfant du processus démarré par la commande `xfce4-terminal`.

3. Citer le `PID` de deux processus qui ont le même parent.

4. Parmi tous les processus affichés, citer le `PID` des deux qui ont consommé le plus de temps du processus.

Partie C

On considère le réseau suivant composé de sept routeurs.



On donne les tables de routage préalablement construites ci-dessous avec le protocole RIP (Routing information Protocole). Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur, la distance, en nombre de sauts, qui le sépare d'un autre routeur.

Table de routage R1

Destination	Lien	Distance
R2	R2	1
R3	R4	2
R4	R4	1
R5	R4	2
R6	R4	2
R7	R4	2

Table de routage R2

Destination	Lien	Distance
R1	R1	1
R3	R3	1
R4	R1	2
R5	R3	3
R6	R3	3
R7	R1	3

Table de routage R3

Destination	Lien	Distance
R1	R2	2
R2	R2	1
R4	R4	1
R5	R4	2
R6	R4	2
R7	R4	2

Table de routage R4

Destination	Lien	Distance
R1	R1	1
R2	R3	2
R3	R3	1
R5	R5	1
R6	R6	1
R7	R7	1

Table de routage R5

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R6	R6	1
R7	R6	2

Table de routage R6

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R5	R5	1
R7	R7	1

Table de routage R7

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R5	R4	2
R6	R6	1

1. Le routeur R2 doit envoyer un paquet de données au routeur R7 qui en accuse réception. Déterminer le chemin parcouru par le paquet de données ainsi que celui parcouru par l'accusé de réception.

2. a) Indiquer la faiblesse que présente ce réseau en cas de panne du routeur R4.

   b) Proposer une solution pour y remédier.

3. **Dans cette question uniquement**, on décide de rajouter un routeur R8 qui sera relié aux routeurs R2 et R6.

   a) Donner une table de routage pour R8 qui minimise le nombre de saut.

   b) Donner une nouvelle table de routage de R2,

4. **Pour la suite de l'exercice on considère le réseau sans le routeur R8.**

   Il a été décidé de modifier les règles de routage de ce réseau en appliquant dorénavant le protocole de routage OSPF qui prend en compte la bande passante.

   Ce protocole attribue un coût à chaque liaison afin de privilégier le choix de certaines routes plus rapides. Plus le coût est faible, plus le lien est intéressant.

   Le coût d'une liaison est calcul par la formule :

$$\text{coût} = \frac{10^8 \text{bit/s}}{\text{bande passante du lien en bit/s}}$$

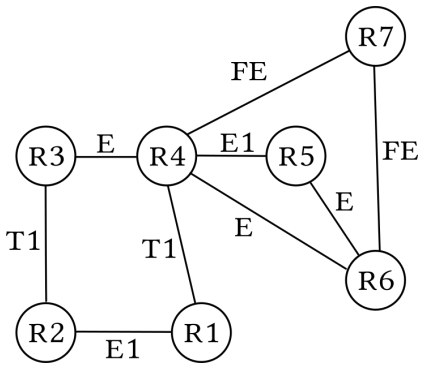
   Voici le tableau référençant les coûts de liaisons en fonction du type de liaison entre 2 routeurs :

Type de liaison	Bande passante	Coût
FastEthernet (FE)	?	1
Ethernet (E)	10 Mb/s	?
(E1)	2,048 Mb/s	49
(T1)	1,544 Mb/s	65

   On rappelle que 1 Mb/s = 1000 kb/s = 10^6 bit/s.

   a) Déterminer la bande passante du FastEthernet (FE) et justifier que le coût du réseau de type Ethernet (E) est de 10.

   On précise sur le graphe ci-dessous les types de liaison dans notre réseau :



b) Le coût d'un chemin est la somme des coûts des liaisons rencontrés. Donner en justifiant le chemin le moins coûteux pour relier R2 à R5. Préciser le coût.

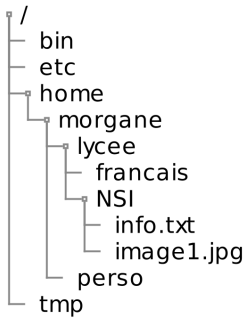
exercice054

Exercice

Partie A

Cet exercice pourra utiliser des commandes de systèmes d'exploitation de type UNIX telles que `cd`, `ls` , `mkdir`, `rm` , `rmd`, `mv`, `cat`.

Dans un système d'exploitation de type UNIX, on considère l'arborescence des fichiers suivante :



On souhaite, grâce à l'utilisation du terminal de commande, explorer et modifier les répertoires et fichiers présents.

On suppose qu'on se trouve actuellement dans le dossier `/home/morgane/lycee`

1. a) Parmi les 4 propositions suivantes, donner celle correspondant à l'affichage obtenu lors de l'utilisation de la commande `ls` :

   - **proposition 1** : francais NSI info.txt image1.jpg
   - **proposition 2** : francais NSI
   - **proposition 3** : lycee
   - **proposition 4** : bin etc home tmp

   b) Écrire la commande qui permet, à partir de cet emplacement, d'atteindre le répertoire `morgane`.

On suppose maintenant qu'on se trouve dans le répertoire `/home/morgane/lycee/NSI`.

2. a) Écrire la commande qui permet de créer, à cet emplacement, le dossier nommé `algorithmique`.

   b) Écrire la commande qui permet, à partir de cet emplacement, de supprimer le fichier `info.txt`.

Partie B

La commande UNIX `ps` présente un cliché instantané des processus en cours d'exécution.

Avec l'option `-eo pid,ppid,stat,command`, cette commande affiche dans l'ordre l'identifiant du processus `PID` (process identifier), le `PPID` (parent process identifier), l'état `STAT` et le nom de la commande à l'origine du processus.

Les valeurs du champ `STAT` indique l'état des processus :

- `R` : processus en cours d'exécution
- `S` : processus endormi (en attente d'une ressource)

Sur un ordinateur, on exécute la commande `ps -eo pid,ppid,stat,command` et on obtient un affichage dont on donne ci-dessous un extrait.

```
$ ps -eo pid,ppid,stat,command

PID   PPID  STAT  COMMAND
1     0     Ss    /sbin/init
....  ....  ..    ...
1912  1908  Ss    Bash
2014  1912  Ss    Bash
1920  1747  Sl    Gedit
2013  1912  Ss    Bash
2091  1593  Sl    /usr/lib/firefox/firefox
5437  1912  Sl    python programme1.py
5440  2013  R     python programme2.py
5450  1912  R+    ps -eo pid,ppid,stat,command
```

À l'aide de cet affichage, répondre aux questions ci-dessous.

1. Quel est le nom de la première commande exécutée par le système d'exploitation lors du démarrage ?

2. Quels sont les identifiants des processus actifs sur cet ordinateur au moment de l'appel de la commande `ps` ? Justifier la réponse.

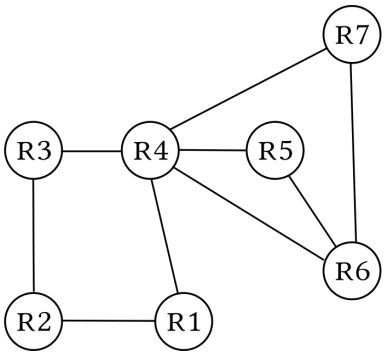
3. Depuis quelle application a-t-on exécuté la commande `ps` ? Donner les autres commandes qui ont été exécutées à partir de cette application.

4. Expliquer l'ordre dans lequel les deux commandes `python programme1.py` et `python programme2.py` ont été exécutées.

5. Peut-on prédire que l'une des deux commandes `python programme1.py` et `python programme2.py` finira avant l'autre ?

Partie C

On considère le réseau suivant composé de sept routeurs.



On donne les tables de routage préalablement construites ci-dessous avec le protocole RIP (Routing information Protocole). Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur, la distance, en nombre de sauts, qui le sépare d'un autre routeur.

Table de routage R1

Destination	Lien	Distance
R2	R2	1
R3	R4	2
R4	R4	1
R5	R4	2
R6	R4	2
R7	R4	2

Table de routage R2

Destination	Lien	Distance
R1	R1	1
R3	R3	1
R4	R1	2
R5	R3	3
R6	R3	3
R7	R1	3

Table de routage R3

Destination	Lien	Distance
R1	R2	2
R2	R2	1
R4	R4	1
R5	R4	2
R6	R4	2
R7	R4	2

Table de routage R4

Destination	Lien	Distance
R1	R1	1
R2	R3	2
R3	R3	1
R5	R5	1
R6	R6	1
R7	R7	1

Table de routage R5

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R6	R6	1
R7	R6	2

Table de routage R6

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R5	R5	1
R7	R7	1

Table de routage R7

Destination	Lien	Distance
R1	R4	2
R2	R4	3
R3	R4	2
R4	R4	1
R5	R4	2
R6	R6	1

1. Le routeur R2 doit envoyer un paquet de données au routeur R7 qui en accuse réception. Déterminer le chemin parcouru par le paquet de données ainsi que celui parcouru par l'accusé de réception.

2. a) Indiquer la faiblesse que présente ce réseau en cas de panne du routeur R4.

   b) Proposer une solution pour y remédier.

3. **Dans cette question uniquement**, on décide de rajouter un routeur R8 qui sera relié aux routeurs R2 et R6.

   a) Donner une table de routage pour R8 qui minimise le nombre de saut.

   b) Donner une nouvelle table de routage de R2,

4. **Pour la suite de l'exercice on considère le réseau sans le routeur R8.**

   Il a été décidé de modifier les règles de routage de ce réseau en appliquant dorénavant le protocole de routage OSPF qui prend en compte la bande passante.

   Ce protocole attribue un coût à chaque liaison afin de privilégier le choix de certaines routes plus rapides. Plus le coût est faible, plus le lien est intéressant.

   Le coût d'une liaison est calcul par la formule :

$$\text{coût} = \frac{10^8 \text{bit/s}}{\text{bande passante du lien en bit/s}}$$

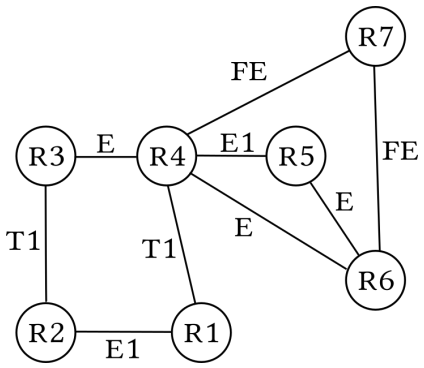
   Voici le tableau référençant les coûts de liaisons en fonction du type de liaison entre 2 routeurs :

Type de liaison	Bande passante	Coût
FastEthernet (FE)	?	1
Ethernet (E)	10 Mb/s	?
(E1)	2,048 Mb/s	49
(T1)	1,544 Mb/s	65

   On rappelle que 1 Mb/s = 1000 kb/s = 10^6 bit/s.

   a) Déterminer la bande passante du FastEthernet (FE) et justifier que le coût du réseau de type Ethernet (E) est de 10.

   On précise sur le graphe ci-dessous les types de liaison dans notre réseau :



b) Le coût d'un chemin est la somme des coûts des liaisons rencontrés. Donner en justifiant le chemin le moins coûteux pour relier R2 à R5. Préciser le coût.

exercice055

Exercice

On souhaite gérer un club de tennis en ligne avec la possibilité de réserver un terrain à un créneau horaire. Le site ne gère que des réservations pour des matchs en simple. Voici la structure de la base de données :

Relation contenant l'ensemble des joueurs du club avec leurs identifiants.
Table `joueurs`

id_joueur	nom_joueur	prenom_joueur	login	mdp
1	Dupont	Alice	alice	1234
2	Durand	Belina	belina	5694
3	Caron	Camilia	camilia	9478
4	Dupont	Dorine	dorine	1347

Relation précisant les matchs joués.
Table `matchs`

id_match	date	id_creneau	id_terrain	id_joueur1	id_joueur2
1	2020-08-01	2	1	1	4
2	2020-08-01	3	1	2	3
3	2020-08-02	6	2	1	3
4	2020-08-02	7	2	2	4
5	2020-08-08	3	3	1	2
6	2020-08-08	5	2	3	4

Relation précisant les différents terrains.
Table `terrains`

id_terrain	nom_terrain	surface
1	stade	terre battue
2	gymnase	synthétique
3	hangar	terre battue

Relation précisant les créneaux réservables.
Table `creneaux`

plage_horaire	id_creneau
1	8h-9h
2	9h-10h
3	10h-11h
4	11h-12h
5	12h-13h
6	13h-14h
7	14h-15h
8	15h-16h
9	16h-17h
10	17h-18h
11	18h-19h
12	19h-20h

1. a) Donner la clé primaire de la relation matchs.
   b) La relation matchs a-t-elle une ou des clés étrangères ? Si oui, quelles sont-elles ?
2. Par lecture et analyse des relations de la base de donnée.
   a) Déterminer le jour et la plage horaire du match entre Durand Belina et Caron Camilia.
   b) Déterminer le nom des deux joueurs qui sont les seuls à avoir joué dans le hangar.
3. Requêtes en langage SQL.
   a) Écrire une requête qui renvoie les prénoms des joueurs dont le nom est ‘Dupont’.
   b) Écrire une requête qui modifie le mot de passe de Dorine Dupont, son nouveau mot de passe étant 1976.
4. Écrire une requête permettant d’ajouter le nouveau membre « Zora MAGID » dont le login est « zora » et le mot de passe 2021.
5. Écrire une requête qui renvoie les jours où Alice joue.

exercice056

Exercice

On dispose de la liste jours suivante et du dictionnaire mois suivant :

```
jours=["dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi"]

mois={1 :("janvier",31),
2 :("février",28),
3 :("mars",31),
4 :("avril",30) ,
5 :("mai",31),
6 :("juin",30),
7 :("juillet",31),
8 :("aout",31),
9 :("septembre",30),
10 :("octobre",31),
11 :("novembre",30),
12 :("décembre",31)}
```

1. a) À partir de la liste jours, comment obtenir l'élément 'lundi' ?

   b) Que renvoie l'instruction `jours[18%7]` ?

2. Écrire le code de la fonction `n_jours_plus_tard(liste_jours, x, n)` qui renvoie le jour situé `n` jours après le jour `x`.

   Par exemple :

   ```
   >>> n_jours_plus_tard(jours, 1, 3)
   'jeudi'
   >>> n_jours_plus_tard(jours, 3, 9)
   'vendredi'
   ```

3. a) À partir du dictionnaire `mois`, comment obtenir le nombre de jours du mois de mars ?

b) Écrire le code de la fonction `n_mois_plus_tard( dico_mois, numero_mois, n)` permettant d'obtenir le nom du mois qu'il sera `n` mois après le mois `numero_mois`.

```
>>> n_mois_plus_tard( mois, 4, 5)
'septembre'
>>> n_mois_plus_tard( mois, 10, 3)
'janvier'
```

4. On définit une date comme un tuple : `(nom_jour,numero_jour, numero_mois,annee)`.

   a) Sachant que `date = ("samedi", 21, 10, 1995)`, que renvoie `mois[date[2]][1]`?

   b) Écrire une fonction `jour_suivant(date)` qui prend en paramètre une date sous forme de tuple et qui renvoie un tuple désignant la date du lendemain.

   On ne tient pas compte des années bissextiles et on considère que le mois de février comporte toujours 28 jours.

   Par exemple :

```
>>>jour_suivant( ("samedi", 21, 10, 1995) )
("dimanche", 22, 10, 1995)
>>> jour_suivant( ("mardi", 31, 10, 1995) )
("mercredi", 1, 11, 1995)
```

exercice057

Exercice

Voici le texte que nous voulons coder.

```
>>> texte_a_coder = "À dix-huit ans (1641), Pascal commence le développement
de la pascaline, machine à calculer capable d'effectuer des additions et des
soustractions20, afin d'aider son père dans son travail. Il en écrit le mode
d'emploi : Avis nécessaire à ceux qui auront la curiosité de voir ladite machine
et s'en servir. Plusieurs exemplaires sont conservés, en France, au Musée des arts
et métiers à Paris et au musée de Clermont-Ferrand."
```

Préparer le texte à coder

Écrire une fonction `preparer(texte,cle)` qui découpe le `texte` en ligne de `cle` caractères. Le résultat sera renvoyé sous forme de **liste** de chaînes de caractères. Veillez bien à ce que la dernière ligne contienne le bon nombre de caractères.

```
>>> solution = ['À dix-huit ans (1',
...             '641), Pascal comm',
...             'ence le développe',
...             'ment de la pascal',
...             'ine, machine à ca',
...             'lculer capable d''',
...             'effectuer des add',
...             'itions et des sou',
...             'stractions20, afi',
...             'n d'aider son père',
...             'e dans son travai',
...             'l. Il en écrit le',
...             ' mode d'emploi : ',
...             'Avis nécessaire à',
...             ' ceux qui auront ',
```

```
...             'la curiosité de v',
...             'oir ladite machin',
...             'e et s'en servir.',
...             ' Plusieurs exempl',
...             'aires sont conser',
...             'vés, en France, a',
...             'u Musée des arts ',
...             'et métiers à Pari',
...             's et au musée de ',
...             'Clermont-Ferrand.']

>>> assert preparer(texte_a_coder,17) == solution
```

Codage du texte

Écrire une fonction `coder(texte,cle)` qui renvoie le texte crypté. Cette fonction fera appel à la fonction `preparer(texte,cle)`.

```
>>> texte_code_1 = "À6emileisnel A loe avuesC 4nencftt  .mvcai Pié t ld1cneufirdd oie
relrsM eei)et,leoa'aIdsuc tue,umtrx,   ecncanle xul ss sé m- ldmrtstis n rasi
eétaohPeea u id edéqid'esneiunua  cceeoesn'cuoieuo  e tisdlhartnro eeistnrnFdrm
-tcéaip  s néms ie stresuF av nadd2s cpsat s  as sealepebee0otrlauémeecn àérn
la lss,nrioir arxoca erscosàe    atirodcvenerP a opc  asapv  enehims,tadn
(mpacddofèal: t irpe sred1mela'duirie à vn.lra i ."
>>> assert coder(texte_a_coder,17) == texte_code_1
```

La clé de décodage

Alice devra également faire parvenir à Bob la clé de décodage. On rappelle qu'il s'agit du nombre de ligne du texte préparé.

Écrire la fonction `cle_de_decryptage(texte,cle)` qui renvoie la clé de décryptage nécessaire pour décoder le texte.

```
>>> assert cle_de_decryptage(texte_a_coder,17) == 25
>>> tc = "À6emileisnel A loe avuesC 4nencftt  .mvcai Pié t ld1cneufirdd oie relrsM
eei)et,leoa'aIdsuc tue,umtrx,   ecncanle xul ss sé m- ldmrtstis  n rasi
eétaohPeea u id edéqid'esneiunua  cceeoesn'cuoieuo  e tisdlhartnro eeistnrnFdrm
-tcéaip  s néms ie stresuF av nadd2s cpsat s  as sealepebee0otrlauémeecn àérn
la lss,nrioir arxoca erscosàe    atirodcvenerP a opc  asapv  enehims,tadn
(mpacddofèal: t irpe sred1mela'duirie à vn.lra i ."
>>> texte_decode = coder(tc,25)
>>> assert texte_decode == texte_a_coder
```

Votre mission

Le fichier `texte_code.txt` contient un texte crypté. Votre mission est de décrypter le paragraphe dont l'indice est précisé ci-dessous. Le premier paragraphe est celui d'indice 0. Le deuxième paragraphe a pour indice 1.

```
>>> def paragraphe_a_dechiffrer(nom):
...     n = 0
...     for i in nom:
...         n += ord(i)
...     return n % 65
>>> PRENOM = "" # rentrez votre prenom
>>> np = paragraphe_a_dechiffrer(PRENOM)
```

La fonction `liste_paragraphe(fichier)` renvoie la liste de tous les paragraphes de `fichier`. Si vous parcourez le fichier `texte_code.txt` vous constaterez qu'un paragraphe se termine par un `\n` et est séparé d'un autre paragraphe par une ligne vide, qui en réalité n'est pas vide mais contient uniquement un `\n`.

```
>>> def liste_paragraphes(fichier):
...         lp = []
...         f = open(fichier,"r")
...         for ligne in f:
...             if len(ligne) != 1:
...                 lp.append(ligne[:-1])
...         f.close()
...         return lp
```

Ce codage n'est pas fiable surtout quand on a un casseur de vous en face. Avec un ordinateur il suffit de lui faire afficher le texte décodé en parcourant un ensemble de clé possible. En faisant l'hypothèse que la clé de décodage est comprise entre 2 et 250, déchiffrer le paragraphe que vous devez déchiffrer.

```
>>> lp = liste_paragraphes("texte_code.txt")
>>> paragraphe_a_dechiffrer = lp[np]
```

Préciser, dans votre fichier, la clé de décryptage que vous avez utilisée pour déchiffrer le paragraphe.

exercice058

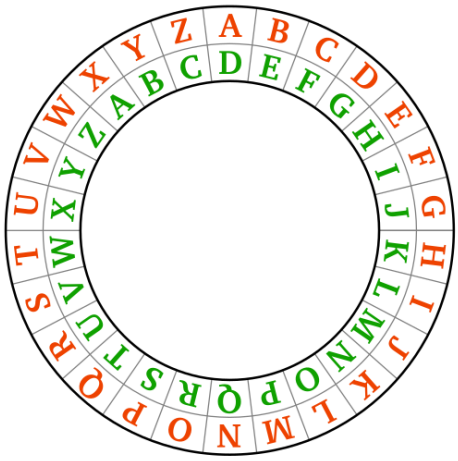
Exercice

Le *chiffre de César* est une façon simple de coder un message afin de conserver le secret du contenu jusqu'à son destinataire. Il s'agit tout simplement de décaler chaque lettre du message. Voyons l'exemple d'un décalage de trois lettres : A devient D ; B devient E ; etc.

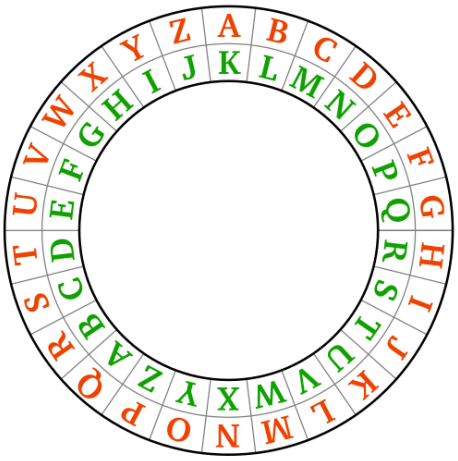
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	lettres du message en clair
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C	lettres du message codé

Par exemple le message : *CAPTUREZ IDEFIX* se chiffre en : *FDSWXUHC LGHILA*

Une autre façon de présenter le décalage est de placer les alphabets en clair (anneau extérieur) et codé (anneau intérieur) sur deux roues concentriques. À gauche un décalage avec $k = 3$ et à droite un décalage avec $k = 10$.



Décalage $k = 3$



Décalage $k = 10$

Pour chiffrer des messages tu passes des lettres à l'extérieur aux lettres à l'intérieur. Pour déchiffrer des messages il suffit de faire l'opération inverse

Déchiffre à la main les messages suivants :

- *EORTXHC DVWHULA* chiffré avec un décalage $k = 3$.
- *YE OCD ZKXYBKWSH* chiffré avec un décalage $k = 10$.

Passer d'un caractère à un nombre et inversement

On préfère travailler avec des nombres qu'avec des lettres! On supporera que les messages sont écrits en majuscules sans ponctuation. Pour la suite du TP, on rappelle que :

- `ord(x)` retourne le code UTF-8 (un entier) du caractère `x`;
- `chr(x)` retourne le caractère dont le code UTF-8 est l'entier `x`.

De la lettre au nombre et réciproquement

À chaque lettre de l'alphabet, on souhaite associer sa position dans l'alphabet en commençant à 0.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

1. Écrire une fonction `c2m(x)` qui retourne l'entier entre 0 et 25 correspondant au caractère `x`.

   ```
   >>> c2m("A")
   0
   >>> c2m("Z")
   25
   ```

2. Écrire une fonction `m2c(x)` qui retourne le caractère correspondant à l'entier x, $0 \leqslant x \leqslant 25$. Par exemple, `m2c(10)` doit renvoyer "K".

   ```
   >>> m2c(10)
   'K'
   >>> m2c(16)
   'Q'
   ```

3. Programmer une fonction `code_caractere(x,k)` qui renvoie le caractère `x` décalé de k rang (modulo 26).

   ```
   >>> code_caractere("A", -1)
   'Z'
   >>> code_caractere("A", 10)
   'K'
   ```

4. Programmer une fonction `code_phrase(phrase,k)` qui renvoie la phrase codée par un décalage de César k.

   ```
   >>> code_phrase("J AIME LA POTION MAGIQUE",-5)
   'E VDHZ GV KJODJI HVBDLPZ'
   ```

5. Programmer une fonction `decode_phrase(phrase,k)` qui décode la phrase codée par un décalage de César k.

```
>>> decode_phrase("EORTXHC DVWHULA", 3)
'BLOQUEZ ASTERIX'
```

6. Vous vous placez dans la peau d'un espion qui a intercepté un message codé par un chiffre de César, mais qui ne connaît pas la clé k. Programmer une fonction `attaque_brute(texte)` qui affiche les déchiffrements, en testant toutes les clés k possibles.
   Vous avez intercepté le message envoyé par le camp Babaorum à Jules César :

   HSFGJSEAP F S HDMK VW HGLAGF

   Que va maintenant faire César ?

```
>>> attaque_brute("HSFGJSEAP F S HDMK VW HGLAGF")
0 HSFGJSEAP F S HDMK VW HGLAGF
1 GREFIRDZO E R GCLJ UV GFKZFE
2 FQDEHQCYN D Q FBKI TU FEJYED
3 EPCDGPBXM C P EAJH ST EDIXDC
4 DOBCFOAWL B O DZIG RS DCHWCB
5 CNABENZVK A N CYHF QR CBGVBA
6 BMZADMYUJ Z M BXGE PQ BAFUAZ
7 ALYZCLXTI Y L AWFD OP AZETZY
8 ZKXYBKWSH X K ZVEC NO ZYDSYX
9 YJWXAJVRG W J YUDB MN YXCRXW
10 XIVWZIUQF V I XTCA LM XWBQWV
11 WHUVYHTPE U H WSBZ KL WVAPVU
12 VGTUXGSOD T G VRAY JK VUZOUT
13 UFSTWFRNC S F UQZX IJ UTYNTS
14 TERSVEQMB R E TPYW HI TSXMSR
15 SDQRUDPLA Q D SOXV GH SRWLRQ
16 RCPQTCOKZ P C RNWU FG RQVKQP
17 QBOPSBNJY O B QMVT EF QPUJPO
18 PANORAMIX N A PLUS DE POTION
19 OZMNQZLHW M Z OKTR CD ONSHNM
20 NYLMPYKGV L Y NJSQ BC NMRGML
21 MXKLOXJFU K X MIRP AB MLQFLK
22 LWJKNWIET J W LHQO ZA LKPEKJ
23 KVIJMVHDS I V KGPN YZ KJODJI
24 JUHILUGCR H U JFOM XY JINCIH
25 ITGHKTFBQ G T IENL WX IHMBHG
```

exercice059

Exercice

L'objectif de cet exercice est d'écrire un petit programme qui aide à décrypter un message codé suivant le codage de César évolué. C'est-à-dire que les lettres ne sont pas justes décalées mais mélangées aléatoirement. Pour simplifier le problème, les messages sont écrits en majuscules sans ponctuation.

A. Quelques fonctions utiles

1. Écrire une fonction `alphabet()` qui renvoie une liste avec les lettres de l'alphabet en majuscules.

```
>>> a = alphabet()
>>> type(a)
<class 'list'>
>>> len(a)
26
```

2. Écrire une fonction `nouveau_dico_alphabet()` qui renvoie un dictionnaire associant chaque lettre de l'alphabet à un point (`'.'`). Le dictionnaire ainsi crée pourra nous être utile pour conserver l'association entre deux lettres. Par exemple, `d['A'] = 'E'` signifiera que le `'E'` est codé par un `'A'`. En revanche, `d['D'] = '.'` signifie qu'on ne sait pas à quelle lettre correspond le `'D'`.

```
>>> d = nouveau_dico_alphabet()
>>> type(d)
<class 'dict'>
>>> len(d)
26
>>> d['A']
'.'
```

3. Écrire une fonction `index_max(l)` qui renvoie l'index du maximum de la liste `l`, le premier s'il y le maximum apparaît plusieurs fois.

```
>>> index_max([12, 5, 6, 15, 8, 9])
3
>>> index_max([12, 5, 6, 5, 8, 9])
0
```

B. Quelques statistiques

Cette méthode de codage peut être cassée grâce aux statistiques. En effet, la fréquence d'apparition des lettres dépend de la langue dans laquelle a été écrite le message. Voila les statistiques du français à partir du roman *Germinal* d'Émile Zola.

FRÉQUENCE DES LETTRES DANS LA LANGUE FRANÇAISE

source: *Germinal*, Émile Zola. (786 770 lettres)

E	17,18%		P	2,37 %
A	9,22%		V	1,64 %
S	7,82%		F	1,10 %
I	7,50%		H	1,07 %
T	7,37%		G	1,06 %
N	7,03%		B	1,04 %
R	6,61%		Q	1,02 %
U	6,36%		X	0,41 %
L	6,13%		J	0,38 %
O	4,91%		Y	0,23 %
D	3,75%		Z	0,12 %
C	3,02%		K	0,01 %
M	2,65%		W	0,20 %

Pour décoder un texte, nous allons donc étudier la fréquence d'apparition de chaque lettre. Le tableau précédent nous permettra ensuite d'orienter nos recherches.

1. Écrire une fonction `effectif(texte)` qui renvoie une liste avec le nombre d'apparition d'une lettre dans `texte`. Par exemple, si `e = effectif(texte)`, `e[0]` correspond au nombre de `'A'` dans le `texte`, `e[1]` correspond au nombre de `'B'` dans le `texte`, etc…

```
>>> t = "ABRACADABRA"
>>> e = effectifs(t)
>>> e[0]
5
>>> e[1]
2
>>> e[2]
1
>>> e[3]
1
```

```
>>> e[4]
0
```

2. Écrire une fonction `lettre_les_plus_frequentes(liste)` qui, à partir d'une liste correspondant aux effectifs des lettres d'une texte, renvoie les 10 lettres les plus fréquentes dans un texte, dans l'ordre décroissant de fréquence d'apparition.

```
>>> t = "MCXZ JC AJCQXT ICZT  ZSBZ JC XBQL ZCXZ TLSQJTZ  M BXT SGZOBIQLT TL
M BXT TACQZZTBI M TXOIT  BX PSHHT ZBQECQL ZTBJ JC VICXMT ISBLT MT HCIOPQTXXTZ
C HSXLZSB  MQF YQJSHTLITZ MT ACET OSBACXL LSBL MISQL  C LICETIZ JTZ OPCHAZ MT
GTLLTICETZ  MTECXL JBQ  QJ XT ESRCQL HTHT ACZ JT ZSJ XSQI  TL QJ X CECQL JC
ZTXZCLQSX MT J QHHTXZT PSIQKSX AJCL NBT ACI JTZ ZSBUUJTZ MB ETXL MT HCIZ  MTZ
ICUCJTZ JCIVTZ OSHHT ZBI BXT HTI  VJCOTTZ M CESQI GCJCRT MTZ JQTBTZ MT HCICQZ
TL MT LTIITZ XBTZ  CBOBXT SHGIT M CIGIT XT LCOPCQL JT OQTJ  JT ACET ZT MTISBJCQL
CETO JC ITOLQLBMT M BXT WTLTT  CB HQJQTB MT J THGIBX CETBVJCXL MTZ LTXTGITZ"
>>> e = effectifs(t)
>>> lettre_les_plus_frequentes(e)
['T', 'C', 'Z', 'L', 'I', 'J', 'B', 'X', 'Q', 'M']
```

C. Vers un assistant de décodage

On veut fabriquer un outil qui nous aide à décoder un message. Pour cela, on va utiliser un dictionnaire qui va garder en mémoire toutes les lettres déjà trouvées et qui va nous afficher le texte avec ce niveau de connaissance.

Par exemple, si le texte codé est `"ZNTZVZFZNTZ"`. Si on pense que le `'Z'` correspond à un `'A'`, et que c'est la seule lettre que l'on a découvert, alors on veut afficher `'A..A.A.A..A'`.

1. Écrire une fonction `traduire(texte,dico)` qui renvoie seulement les lettres connues du `texte`.

```
>>> t = "ZNTZVZFZNTZ"
>>> d = nouveau_dico_alphabet()
>>> d['Z'] = 'A'
>>> traduire(t,d)
'A..A.A.A..A'
```

Je pense que maintenant vous avez assez d'outils pour programmer un assistant de décodage qui vous aidera à décoder le message de l'exercice Moodle.

Bon courage! Soyez persévérant!

exercice060

Exercice

Dans cet exercice, les nombres sont des entiers ou des flottants.

Écrire une fonction `moyenne` renvoyant la moyenne pondérée d'une liste non vide, passée en paramètre, de tuples à deux éléments de la forme `(valeur, coefficient)` où `valeur` et `coefficient` sont des nombres positifs ou nuls. Si la somme des coefficients est nulle, la fonction renvoie `None`, si la somme des coefficients est non nulle, la fonction renvoie, sous forme de flottant, la moyenne des valeurs affectées de leur coefficient.

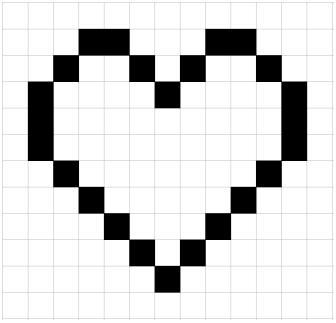
Exemple :

```
>>> moyenne([(8, 2), (12, 0), (13.5, 1), (5, 0.5)])
9.142857142857142
>>> moyenne([(3, 0), (5, 0)])
```

Dans le premier exemple la moyenne est calculée par la formule :
$$\frac{8\times 2+12\times 0+13,5\times 1+5\times 0,5}{2+0+1+0,5}$$

exercice061

Exercice



On travaille sur des dessins en noir et blanc obtenu à partir de pixels noirs et blancs : La figure « cœur » ci-contre va servir d'exemple. On la représente par une grille de nombres, c'est-à-dire par une liste composée de sous-listes de même longueurs. Chaque sous-liste représentera donc une ligne du dessin.
Dans le code ci-dessous, la fonction `affiche` permet d'afficher le dessin. Les pixels noirs (1 dans la grille) seront représentés par le caractère "*" et les blancs (0 dans la grille) par deux espaces.

La fonction `zoomListe` prend en argument une liste `liste_depart` et un entier `k`. Elle renvoie une liste où chaque élément de `liste_depart` est dupliqué `k` fois.
La fonction `zoomDessin` prend en argument la grille `dessin` et renvoie une grille où toutes les lignes de `dessin` sont zoomées `k` fois et répétées `k` fois.

Soit le code ci-dessous :

```
coeur = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0],
         [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0],
         [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
         [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
         [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
         [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

def affiche(dessin):
    """
```

```
    affichage d'une grille : les 1 sont représentés par
        des ' *' , les 0 par deux espaces '  '
    """
    for ligne in dessin:
        for col in ligne:
            if col == 1:
                print(" *", end="")
            else:
                print("  ", end="")
        print()


def zoomListe(liste_depart,k):
    """
    renvoie une liste contenant k fois chaque
    élément de liste_depart
    """
    liste_zoom = ...
    for elt in ... :
        for i in range(k):
            ...
    return liste_zoom

def zoomDessin(grille,k):
    """
    renvoie une grille où les lignes sont zoomées k fois
    ET répétées k fois
    """
    grille_zoom=[]
    for elt in grille:
        liste_zoom = ...
        for i in range(k):
            ... .append(...)
    return grille_zoom
```

Résultats à obtenir :

```
>>> affiche(coeur)
<BLANKLINE>
       * *       * *
     *     *   *     *
```

```
   *         *         *
   *                   *
   *                   *
     *               *
       *           *
         *       *
           *   *
             *
<BLANKLINE>

>>> affiche(zoomDessin(coeur,3))
<BLANKLINE>
<BLANKLINE>
<BLANKLINE>
                  * * * * * *                   * * * * * *
                  * * * * * *                   * * * * * *
                  * * * * * *                   * * * * * *
             * * *             * * *       * * *             * * *
             * * *             * * *       * * *             * * *
             * * *             * * *       * * *             * * *
       * * *                         * * *                         * * *
       * * *                         * * *                         * * *
       * * *                         * * *                         * * *
       * * *                                                       * * *
       * * *                                                       * * *
       * * *                                                       * * *
       * * *                                                       * * *
       * * *                                                       * * *
       * * *                                                       * * *
             * * *                                           * * *
             * * *                                           * * *
             * * *                                           * * *
                   * * *                               * * *
                   * * *                               * * *
                   * * *                               * * *
                         * * *                   * * *
                         * * *                   * * *
                         * * *                   * * *
                               * * *       * * *
                               * * *       * * *
                               * * *       * * *
```

* * *

* * *

* * *

<BLANKLINE>

<BLANKLINE>

<BLANKLINE>

exercice062

Exercice

Écrire une fonction `a_doublon` qui prend en paramètre une liste **triée** de nombres et renvoie `True` si la liste contient au moins deux nombres identiques, `False` sinon.

Par exemple :

```
>>> a_doublon([])
False
>>> a_doublon([1])
False
>>> a_doublon([1, 2, 4, 6, 6])
True
>>> a_doublon([2, 5, 7, 7, 7, 9])
True
>>> a_doublon([0, 2, 3])
False
```

exercice063

Exercice

On souhaite générer des grilles du jeu de démineur à partir de la position des bombes à placer.

On se limite à la génération de grilles carrées de taille $n \times n$ où n est le nombre de bombes du jeu.

Dans le jeu du démineur, chaque case de la grille contient soit une bombe, soit une valeur qui correspond aux nombres de bombes situées dans le voisinage direct de la case (au-dessus, en dessous, à droite, à gauche ou en diagonale : chaque case a donc 8 voisins si elle n'est pas située au bord de la grille).

Voici un exemple de grille 5×5 de démineur dans laquelle la bombe est représentée par une étoile :

1	1	1	0	0
1	*	1	1	1
2	2	3	2	*
1	*	2	*	3
1	1	2	2	*

On utilise une liste de listes pour représenter la grille et on choisit de coder une bombe par la valeur -1.

L'exemple ci-contre sera donc codé par la liste :

```
[[1, 1, 1, 0, 0],
 [1, -1, 1, 1, 1],
 [2, 2, 3, 2, -1],
 [1, -1, 2, -1, 3],
 [1, 1, 2, 2, -1]]
```

Compléter le code suivant afin de générer des grilles de démineur, on pourra vérifier que l'instruction `genere_grille([(1, 1), (2, 4), (3, 1), (3, 3), (4, 4)])` produit bien la liste donnée en exemple.

```python
def voisinage(n, ligne, colonne):
    """
    Renvoie la liste des coordonnées des voisins de la case
    (ligne, colonne) en gérant les cases sur les bords.
    """
    voisins = []
    for l in range(max(0,ligne-1), min(n, ligne+2)):
        for c in range(max(0, colonne-1), min(n, colonne+2)):
            if (l, c) != (ligne, colonne):
                voisins.append((l,c))
    return voisins


def incremente_voisins(grille, ligne, colonne):
    """
    Incrémente de 1 toutes les cases voisines d'une bombe.
    """
    voisins = ...
    for l, c in voisins:
        if grille[l][c] != ...: # si ce n'est pas une bombe
            ...  # on ajoute 1 à sa valeur


def genere_grille(bombes):
    """
    Renvoie une grille de démineur de taille nxn où n est
    le nombre de bombes, en plaçant les bombes à l'aide de
    la liste bombes de coordonnées (tuples) passée en
    paramètre.
    """
    n = len(bombes)
    # Initialisation d'une grille nxn remplie de 0
    grille = [[0 for colonne in range(n)] for ligne in range(n)]
    # Place les bombes et calcule les valeurs des autres cases
    for ligne, colonne in bombes:
        grille[ligne][colonne] = ... # place la bombe
        ... # incrémente ses voisins

    return grille
```

exercice064

Exercice

Écrire en python deux fonctions :

- `lancer` de paramètre `n`, un entier positif, qui renvoie un tableau de type `list` de `n` entiers obtenus aléatoirement entre 1 et 6 (1 et 6 inclus) ;
- `paire_6` de paramètre `tab`, un tableau de type `list` de `n` entiers entre 1 et 6 obtenus aléatoirement, qui renvoie un booléen égal à `True` si le nombre de 6 est supérieur ou égal à 2, `False` sinon.

On pourra utiliser la fonction `randint(a,b)` du module `random` pour laquelle la documentation officielle est la suivante : `Renvoie un entier aléatoire N tel que a <=N <= b.`

Démarche à suivre impérativement : importer le module `random` puis appeler la fonction `random.randint`.

```
import random # pour importer le module random
N = random.randint(1, 6) # pour lancer un dé
```

Exemples :

```
>>> lancer1 = lancer(5)
>>> lancer1
[5, 6, 6, 2, 2]
>>> paire_6(lancer1)
True
>>> lancer2 = lancer(5)
>>> lancer2
[6, 5, 1, 6, 6]
>>> paire_6(lancer2)
True
>>> lancer3 = lancer(3)
>>> lancer3
```

```
[2, 2, 6]
>>> paire_6(lancer3)
False
>>> lancer4 = lancer(0)
>>> lancer4
[]
>>> paire_6(lancer4)
False
```

exercice065

Exercice

On considère une image en 256 niveaux de gris que l'on représente par une grille de nombres, c'est-à-dire une liste composée de sous-listes toutes de longueurs identiques.

La largeur de l'image est donc la longueur d'une sous-liste et la hauteur de l'image est le nombre de sous-listes.

Chaque sous-liste représente une ligne de l'image et chaque élément des sous-listes est un entier compris entre 0 et 255, représentant l'intensité lumineuse du pixel.

Le négatif d'une image est l'image constituée des pixels `x_n` tels que `x_n + x_i = 255` où `x_i` est le pixel correspondant de l'image initiale.

Compléter le programme suivant :

```
def nbLig(image):
    """renvoie le nombre de lignes de l'image"""
    return ...

def nbCol(image):
    """renvoie la largeur de l'image"""
    return ...

def negatif(image):
    """renvoie le negatif de l'image sous la forme
       d'une liste de listes"""

    # on cree une image de 0 aux memes dimensions que le parametre image
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]

    for i in range(nbLig(image)):
        for j in range(...):
            L[i][j] = ...
    return L
```

```
def binaire(image, seuil):
    """renvoie une image binarisee de l'image sous la forme
       d'une liste de listes contenant des 0 si la valeur
       du pixel est strictement inferieure au seuil
       et 1 sinon"""

    # on cree une image de 0 aux memes dimensions que le parametre image
    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]

    for i in range(nbLig(image)):
        for j in range(...):
            if image[i][j] < ... :
                L[i][j] = ...
            else:
                L[i][j] = ...
    return L
```

Exemples :

```
>>> img=[[20, 34, 254, 145, 6], [23, 124, 237, 225, 69], [197, 174,
207, 25, 87], [255, 0, 24, 197, 189]]
>>> nbLig(img)
4
>>> nbCol(img)
5
>>> negatif(img)
[[235, 221, 1, 110, 249], [232, 131, 18, 30, 186], [58, 81, 48, 230,
168], [0, 255, 231, 58, 66]]
>>> binaire(img,120)
[[0, 0, 1, 1, 0], [0, 1, 1, 1, 0], [1, 1, 1, 0, 0], [1, 0, 0, 1, 1]]
```

exercice066

Exercice

Programmer la fonction `fusion` prenant en paramètres deux tableaux non vides `tab1` et `tab2` (type `list`) d'entiers, chacun dans l'ordre croissant, et renvoyant un tableau trié dans l'ordre croissant et contenant l'ensemble des valeurs de `tab1` et `tab2`.

Exemples :

```
>>> fusion([3, 5], [2, 5])
[2, 3, 5, 5]
>>> fusion([-2, 4], [-3, 5, 10])
[-3, -2, 4, 5, 10]
>>> fusion([4], [2, 6])
[2, 4, 6]
```

exercice067

Exercice

Le but de cet exercice est d'écrire une fonction récursive `traduire_romain` qui prend en paramètre une chaîne de caractères, non vide, représentant un nombre écrit en chiffres romains et qui renvoie son écriture décimale.

Les chiffres romains considérés sont : I, V, X, L, C, D et M. Ils représentent respectivement les nombres 1, 5, 10, 50, 100, 500, et 1000 en base dix.

On dispose d'un dictionnaire `romains` dont les clés sont les caractères apparaissant dans l'écriture en chiffres romains et les valeurs sont les nombres entiers associés en écriture décimale :

`romains = {"I":1, "V":5, "X":10, "L":50, "C":100, "D":500, "M":1000}`

Le code de la fonction `traduire_romain` fournie repose sur le principe suivant :

- la valeur d'un caractère est ajoutée à la valeur du reste de la chaîne si ce caractère a une valeur supérieure (ou égale) à celle du caractère qui le suit ;
- la valeur d'un caractère est retranchée à la valeur du reste de la chaîne si ce caractère a une valeur strictement inférieure à celle du caractère qui le suit.

Ainsi, XIV correspond au nombre 10 + 5 - 1 puisque :

- la valeur de X (10) est supérieure à celle de I (1), on ajoute donc 10 à la valeur du reste de la chaîne, c'est-à-dire IV ;
- la valeur de I (1) est strictement inférieure à celle de V (5), on soustrait donc 1 à la valeur du reste de la chaîne, c'est-à-dire V.

On rappelle que pour priver une chaîne de caractères de son premier caractère, on utilisera l'instruction : `nom_de_variable[1:]`

Par exemple, si la variable `mot` contient la chaîne `"CDI"`, `mot[1:]` renvoie `"DI"`.

```
romains = {"I":1, "V":5, "X":10, "L":50, "C":100, "D":500, "M":1000}

def traduire_romain(nombre):
```

```
    """ Renvoie l'écriture décimale du nombre donné en chiffres
    romains """
    if len(nombre) == 1:
        return ...
    elif romains[nombre[0]] >= ...
        return romains[nombre[0]] + ...
    else:
        return ...
```

Compléter le code de la fonction `traduire_romain` et le tester.

```
>>> traduire_romain("XIV")
14
>>> traduire_romain("CXLII")
142
>>> traduire_romain("MMXXIII")
2023
```

exercice068

Exercice

Sur le réseau social TipTop, on s'intéresse au nombre de «like» des abonnés. Les données sont stockées dans des dictionnaires où les clés sont les pseudos et les valeurs correspondantes sont les nombres de «like» comme ci-dessous :

```
{'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}
```

Écrire une fonction `max_dico` qui :

- Prend en paramètre un dictionnaire `dico` non vide dont les clés sont des chaînes de caractères et les valeurs associées sont des entiers ;
- Renvoie un tuple dont :
  - La première valeur est la clé du dictionnaire associée à la valeur maximale ;
  - La seconde valeur est la première valeur maximale présente dans le dictionnaire.

Exemples :

```
>>> max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50})
('Ada', 201)
>>> max_dico({'Alan': 222, 'Ada': 201, 'Eve': 220, 'Tim': 50})
('Alan', 222)
```

exercice069

Exercice

Nous avons l'habitude de noter les expressions arithmétiques avec des parenthèses comme par exemple : $(2 + 3) \times 5$.

Il existe une autre notation utilisée par certaines calculatrices, appelée notation postfixe, qui n'utilise pas de parenthèses. L'expression arithmétique précédente est alors obtenue en saisissant successivement 2, puis 3, puis l'opérateur +, puis 5, et enfin l'opérateur ×. On modélise cette saisie par le tableau `[2, 3, '+', 5, '*']`.

Autre exemple, la notation postfixe de $3 \times 2 + 5$ est modélisée par le tableau :
`[3, 2, '*', 5, '+']`.

D'une manière plus générale, la valeur associée à une expression arithmétique en notation postfixe est déterminée à l'aide d'une pile en parcourant l'expression arithmétique de gauche à droite de la façon suivante :

- Si l'élément parcouru est un nombre, on le place au sommet de la pile ;
- Si l'élément parcouru est un opérateur, on récupère les deux éléments situés au sommet de la pile et on leur applique l'opérateur. On place alors le résultat au sommet de la pile.
- À la fin du parcours, il reste alors un seul élément dans la pile qui est le résultat de l'expression arithmétique.

Dans le cadre de cet exercice, on se limitera aux opérations × et +.

Pour cet exercice, on dispose d'une classe `Pile` qui implémente les méthodes de base sur la structure de pile.

Compléter le script de la fonction `eval_expression` qui reçoit en paramètre une liste python représentant la notation postfixe d'une expression arithmétique et qui renvoie sa valeur associée.

```
class Pile:
    """Classe définissant une structure de pile."""
    def __init__(self):
        self.contenu = []

    def est_vide(self):
```

```
        """Renvoie le booléen True si la pile est vide, False sinon."""
        return self.contenu == []

    def empiler(self, v):
        """Place l'élément v au sommet de la pile"""
        self.contenu.append(v)

    def depiler(self):
        """
        Retire et renvoie l'élément placé au sommet de la pile,
        si la pile n'est pas vide.
        """
        if not self.est_vide():
            return self.contenu.pop()


def eval_expression(tab):
    p = Pile()
    for ... in tab:
        if element != '+' ... element != '*':
            p.empiler(...)
        else:
            if element == ...:
                resultat = p.depiler() + ...
            else:
                resultat = ...
            p.empiler(...)
    return ...
```

Exemple :

```
>>> eval_expression([2, 3, '+', 5, '*'])
25
```

exercice070

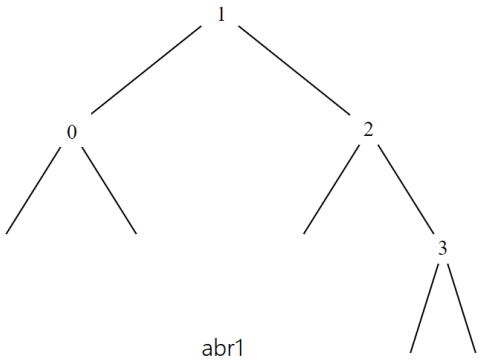
Exercice

On considère la classe ABR, dont le constructeur est le suivant :

```
class ABR:
    def __init__(self, g0, v0, d0):
        self.gauche = g0
        self.cle = v0
        self.droit = d0

    def __repr__(self):
        if self is None:
            return ''
        else:
            return '(' + (self.gauche).__repr__() + ', ' + str(self.cle) + ', '
                                                  +(self.droit).__repr__() + ')'
```

Ainsi, l'arbre binaire de recherche abr1 ci- contre est créé par le code python ci- dessous



abr1

```
>>> n0 = ABR(None, 0, None)
>>> n3 = ABR(None, 3, None)
>>> n2 = ABR(None, 2, n3)
>>> abr1 = ABR(n0, 1, n2)
```

Dans tout le code, None correspondra à un arbre vide.

La classe ABR dispose aussi d'une méthode de représentation (__repr__), qui affiche entre parenthèses le contenu du sous arbre gauche, puis la clé de l'arbre, et enfin le contenu du sous arbre droit. Elle s'utilise en console de la manière suivante :

```
>>> abr1
((None, 0, None), 1, (None, 2, (None, 3, None)))
```

Écrire **une fonction récursive** `ajoute(cle, a)` qui prend en paramètres une clé `cle` et un arbre binaire de recherche `a` , et qui renvoie un arbre binaire de recherche dans lequel `cle` a été insérée.

Dans le cas où `cle` est déjà présente dans `a`, la fonction renvoie l'arbre a inchangé.

Résultats à obtenir :

```
>>> a = ajoute(4, abr1)
>>> a
((None, 0, None), 1, (None, 2, (None, 3, (None, 4, None))))
>>> ajoute(-5, abr1)
(((None, -5, None), 0, None), 1, (None, 2, (None, 3, (None, 4, None))))
>>> ajoute(2, abr1)
(((None, -5, None), 0, None), 1, (None, 2, (None, 3, (None, 4, None))))
```

exercice071

Exercice

On dispose d'un ensemble d'objets dont on connaît, pour chacun, la masse. On souhaite ranger l'ensemble de ces objets dans des boites identiques de telle manière que la somme des masses des objets contenus dans une boîte ne dépasse pas la capacité c de la boîte. On souhaite utiliser le moins de boîtes possibles pour ranger cet ensemble d'objets.

Pour résoudre ce problème, on utilisera un algorithme glouton consistant à placer chacun des objets dans la première boîte où cela est possible.

Par exemple, pour ranger dans des boîtes de capacité `c = 5` un ensemble de trois objets dont les masses sont représentées en Python par la liste `[1, 5, 2]`, on procède de la façon suivante :

- Le premier objet, de masse 1, va dans une première boite.
- Le deuxième objet, de masse 5, ne peut pas aller dans la même boite que le premier objet car cela dépasserait la capacité de la boite. On place donc cet objet dans une deuxième boîte.
- Le troisième objet, de masse 2, va dans la première boîte.

On a donc utilisé deux boîtes de capacité `c = 5` pour ranger les 3 objets.

Compléter la fonction Python `empaqueter(liste_masses, c)` suivante pour qu'elle renvoie le nombre de boîtes de capacité c nécessaires pour empaqueter un ensemble d'objets dont les masses sont contenues dans la liste `liste_masses`.

```
def empaqueter(liste_masses, c):
    n = len(liste_masses)
    nb_boites = 0
    boites = [0]*n
    for masse in ... :
        i = 0
        while i <= nb_boites and boites[i] + ... > C:
            i = i + 1
        if i == nb_boites + 1:
            ...
        boites[i] = ...
    return ...
```

exercice072

Exercice

Écrire une fonction `recherche_indices_classement` qui prend en paramètres un entier `elt` et une liste d'entiers `tab`, et qui renvoie trois listes :

- la première liste contient les indices des valeurs de la liste `tab` strictement inférieures à `elt` ;
- la deuxième liste contient les indices des valeurs de la liste `tab` égales à `elt` ;
- la troisième liste contient les indices des valeurs de la liste `tab` strictement supérieures à `elt`.

Exemples :

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

exercice073

Exercice

Un professeur de NSI décide de gérer les résultats de sa classe sous la forme d'un dictionnaire :

- les clefs sont les noms des élèves ;
- les valeurs sont des dictionnaires dont les clefs sont les types d'épreuves sous forme de chaîne de caractères et les valeurs sont les notes obtenues associées à leurs coefficients dans une liste.

Avec :

```
resultats = {'Dupont': {
                        'DS1': [15.5, 4],
                        'DM1': [14.5, 1],
                        'DS2': [13, 4],
                        'PROJET1': [16, 3],
                        'DS3': [14, 4]
                    },
             'Durand': {
                        'DS1': [6 , 4],
                        'DM1': [14.5, 1],
                        'DS2': [8, 4],
                        'PROJET1': [9, 3],
                        'IE1': [7, 2],
                        'DS3': [8, 4],
                        'DS4':[15, 4]
                    }
             }
```

L'élève dont le nom est Durand a ainsi obtenu au DS2 la note de 8 avec un coefficient 4.

Le professeur crée une fonction moyenne qui prend en paramètre le nom d'un de ses élèves et renvoie sa moyenne arrondie au dixième.

Compléter le code du professeur ci-dessous :

```
def moyenne(nom, dico_result):
    if nom in ...:
        notes = dico_result[nom]
        total_points = ...
        total_coefficients = ...
        for ...  in notes.values():
            note, coefficient = valeurs
            total_points = total_points + ... * coefficient
            total_coefficients = ... + coefficient
        return round( ... / total_coefficients, 1 )
    else:
        return -1
```

Exemple :

```
>>> moyenne('Dupont', resultats)
14.5
```

exercice074

Exercice

Écrire une fonction `max_et_indice` qui prend en paramètre une liste non vide `tab` de nombres entiers et qui renvoie la valeur du plus grand élément de cette liste ainsi que l'indice de sa première apparition dans cette liste.

L'utilisation de la fonction native `max` n'est pas autorisée.

Ne pas oublier d'ajouter au corps de la fonction une documentation et une ou plusieurs assertions pour vérifier les pré-conditions.

Exemples :

```
>>> max_et_indice([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
(9, 3)
>>> max_et_indice([-2])
(-2, 0)
>>> max_et_indice([-1, -1, 3, 3, 3])
(3, 2)
>>> max_et_indice([1, 1, 1, 1])
(1, 0)
```

exercice075

Exercice

L'ordre des gènes sur un chromosome est représenté par un tableau `ordre` de `n` cases d'entiers distincts deux à deux et compris entre 1 et `n`.

Par exemple, `ordre = [5, 4, 3, 6, 7, 2, 1, 8, 9]` dans le cas `n = 9`.

On dit qu'il y a un point de rupture dans `ordre` dans chacune des situations suivantes :

- la première valeur de `ordre` n'est pas 1 ;
- l'écart entre deux gènes consécutifs n'est pas égal à 1 ;
- la dernière valeur de `ordre` n'est pas n.

Par exemple, si `ordre = [5, 4, 3, 6, 7, 2, 1, 8, 9]` avec `n = 9`, on a

- un point de rupture au début car 5 est différent de 1
- un point de rupture entre 3 et 6 (l'écart est de 3)
- un point de rupture entre 7 et 2 (l'écart est de 5)
- un point de rupture entre 1 et 8 (l'écart est de 7)

Il y a donc 4 points de rupture.

Compléter les fonctions Python `est_un_ordre` et `nombre_points_rupture` proposées à la page suivante pour que :

- la fonction `est_un_ordre` renvoie `True` si le tableau passé en paramètre représente bien un ordre de gènes de chromosome et `False` sinon ;
- la fonction `nombre_points_rupture` renvoie le nombre de points de rupture d'un tableau passé en paramètre représentant l'ordre de gènes d'un chromosome.

```
def est_un_ordre(tab):
    """
    Renvoie True si tab est de longueur n et contient tous les entiers
    de 1 à n, False sinon
    """
```

```
    for i in range(1,...):
        if ...:
            return False
    return True


def nombre_points_rupture(ordre):
    """
    Renvoie le nombre de point de rupture de ordre qui représente un ordre
    de gènes de chromosome
    """
    assert ... # ordre n'est pas un ordre de gènes
    n = len(ordre)
    nb = 0
    if ordre[...] != 1: # le premier n'est pas 1
        nb = nb + 1
    i = 0
    while i < ...:
        if ... not in [-1, 1]: # l'écart n'est pas 1
            nb = nb + 1
        i = i + 1
    if ordre[...] != n: # le dernier n'est pas n
        nb = nb + 1
    return nb
```

Exemples :

```
>>> est_un_ordre([1, 6, 2, 8, 3, 7])
False
>>> est_un_ordre([5, 4, 3, 6, 7, 2, 1, 8, 9])
True

>>> nombre_points_rupture([5, 4, 3, 6, 7, 2, 1, 8, 9])
4
>>> nombre_points_rupture([1, 2, 3, 4, 5])
0
```

exercice076

Exercice

Écrire une fonction `ajoute_dictionnaires` qui prend en paramètres deux dictionnaires `d1` et `d2` dont les clés sont des nombres et renvoie le dictionnaire `d` défini de la façon suivante :

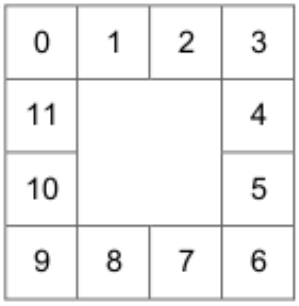
- Les clés de `d` sont celles de `d1` et celles de `d2` réunies.
- Si une clé est présente dans les deux dictionnaires `d1` et `d2`, sa valeur associée dans le dictionnaire `d` est la somme de ses valeurs dans les dictionnaires `d1` et `d2`.
- Si une clé n'est présente que dans un des deux dictionnaires, sa valeur associée dans le dictionnaire `d` est la même que sa valeur dans le dictionnaire où elle est présente.

Exemples :

```
>>> ajoute_dictionnaires({1: 5, 2: 7}, {2: 9, 3: 11})
{1: 5, 2: 16, 3: 11}
>>> ajoute_dictionnaires({}, {2: 9, 3: 11})
{2: 9, 3: 11}
>>> ajoute_dictionnaires({1: 5, 2: 7}, {})
{1: 5, 2: 7}
```

exercice077

Exercice



On considère une piste carrée qui contient 4 cases par côté. Les cases sont numérotées de 0 inclus à 12 exclu comme ci-contre.

L'objectif de l'exercice est d'implémenter le jeu suivant :

Au départ, le joueur place son pion sur la case 0. A chaque coup, il lance un dé équilibré à six faces et avance son pion d'autant de cases que le nombre indiqué par le dé (entre 1 et 6 inclus) dans le sens des aiguilles d'une montre.

Par exemple, s'il obtient 2 au premier lancer, il pose son pion sur la case 2 puis s'il obtient 6 au deuxième lancer, il le pose sur la case 8, puis s'il obtient à nouveau 6, il pose le pion sur la case 2.

Le jeu se termine lorsque le joueur a posé son pion sur **toutes les cases** de la piste.

Compléter la fonction `nbre_coups` ci-dessous de sorte qu'elle renvoie le nombre de lancers aléatoires nécessaires pour terminer le jeu.

Proposer ensuite quelques tests pour en vérifier le fonctionnement.

```
from random import randint

def nbre_coups():
    n = ...
    cases_vues = [0]
    case_en_cours = 0
    nbre_cases = 12
    while ... < ...:
        x = randint(1, 6)
        case_en_cours = (case_en_cours + ...) % ...
        if ...:
            cases_vues.append(case_en_cours)
        n = ...
    return n
```

exercice078

Exercice

Le codage par différence (*delta encoding* en anglais) permet de compresser un tableau de données en indiquant pour chaque donnée, sa différence avec la précédente (plutôt que la donnée elle-même). On se retrouve alors avec un tableau de données plus petit, nécessitant moins de place en mémoire. Cette méthode se révèle efficace lorsque les valeurs consécutives sont proches.

Programmer la fonction `delta(liste)` qui prend en paramètre un tableau non vide de nombres entiers et qui renvoie un tableau contenant les valeurs entières compressées à l'aide cette technique.

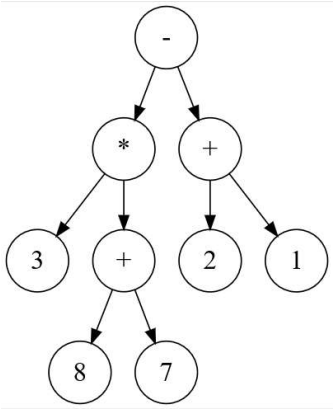
Exemples :

```
>>> delta([1000, 800, 802, 1000, 1003])
[1000, -200, 2, 198, 3]
>>> delta([42])
[42]
```

exercice079

Exercice

Une expression arithmétique ne comportant que les quatre opérations +, −, ×, ÷ peut être représentée sous forme d'arbre binaire. Les nœuds internes sont des opérateurs et les feuilles sont des nombres. Dans un tel arbre, la disposition des nœuds joue le rôle des parenthèses que nous connaissons bien.



En parcourant en profondeur infixe l'arbre binaire ci-dessus, on retrouve l'expression notée habituellement :

$$(3 \times (8 + 7)) - (2 + 1)$$

La classe `Noeud` ci-après permet d'implémenter une structure d'arbre binaire.

Compléter la fonction récursive `expression_infixe` qui prend en paramètre un objet de la classe `Noeud` et qui renvoie l'expression arithmétique représentée par l'arbre binaire passé en paramètre, sous forme d'une chaîne de caractères contenant des parenthèses.

Résultat attendu avec l'arbre ci-dessus :

```
>>> e = Noeud(Noeud(Noeud(None, 3, None), '*', Noeud(Noeud(None, 8, None),
          '+', Noeud(None, 7, None))), '-', Noeud(Noeud(None, 2, None),
          '+', Noeud(None, 1, None)))
>>> expression_infixe(e)
'((3*(8+7))-(2+1))'
```

Complétez le code suivant :

```
class Noeud:
    """
    classe implémentant un noeud d'arbre binaire
    """

    def __init__(self, g, v, d):
        """
        un objet Noeud possède 3 attributs :
        - gauche : le sous-arbre gauche,
        - valeur : la valeur de l'étiquette,
        - droit : le sous-arbre droit.
        """
        self.gauche = g
        self.valeur = v
        self.droit = d

    def __str__(self):
        """
        renvoie la représentation du noeud en chaine de caractères
        """
        return str(self.valeur)

    def est_une_feuille(self):
        """
        renvoie True si et seulement si le noeud est une feuille
        """
        return self.gauche is None and self.droit is None


def expression_infixe(e):
    s = ...
    if e.gauche is not None:
        s = '(' + s + expression_infixe(...)
    s = s + ...
    if ... is not None:
        s = s + ... + ...
    return s
```

exercice080

Exercice

Écrire une fonction `enumere` qui prend en paramètre une liste `L` et renvoie un dictionnaire `d` dont les clés sont les éléments de `L` avec pour valeur associée la liste des indices de l'élément dans la liste `L`.

Exemple :

```
>>> enumere([1, 1, 2, 3, 2, 1])
{1: [0, 1, 5], 2: [2, 4], 3: [3]}
```

exercice081

Exercice

Un arbre binaire est implémenté par la classe `Arbre` donnée ci-dessous.
Les attributs `fg` et `fd` prennent pour valeurs des instances de la classe `Arbre` ou `None`.

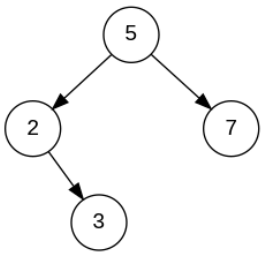
```
class Arbre:
    def __init__(self, etiquette):
        self.v = etiquette
        self.fg = None
        self.fd = None

def parcours(arbre, liste):
    if arbre != None:
        parcours(arbre.fg, liste)
        liste.append(arbre.v)
        parcours(arbre.fd, liste)
    return liste
```

La fonction récursive `parcours` renvoie la liste des étiquettes des nœuds de l'arbre implémenté par l'instance arbre dans l'ordre du parcours en profondeur infixe à partir d'une liste vide passée en argument.

Compléter le code de la fonction `insere` qui insère un nœud d'étiquette `cle` en feuille de l'arbre implémenté par l'instance `arbre` selon la spécification indiquée et de façon que l'arbre ainsi complété soit encore un arbre binaire de recherche.

Tester ensuite ce code en utilisant la fonction parcours et en insérant successivement des nœuds d'étiquette 1, 4, 6 et 8 dans l'arbre binaire de recherche représenté ci- dessous :



```
def insere(arbre, cle):
    """ arbre est une instance de la classe Arbre qui implémente
        un arbre binaire de recherche.
    """
    if ...:
        if ...:
            insere(arbre.fg, cle)
        else:
            arbre.fg = Arbre(cle)
    else:
        if ...:
            insere(arbre.fd, cle)
        else:
            arbre.fd = Arbre(cle)
```

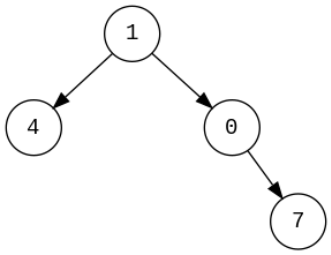
exercice082

Exercice

Un arbre binaire est implémenté par la classe `Arbre` donnée ci-dessous. Les attributs `fg` et `fd` prennent pour valeurs des instances de la classe `Arbre` ou `None`.

```
class Arbre:
    def __init__(self, etiquette):
        self.v = etiquette
        self.fg = None
        self.fd = None
```

L'arbre ci-contre sera donc implémenté de la manière suivante :



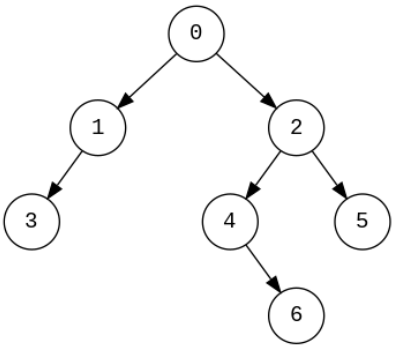

```
a = Arbre(1)
a.fg = Arbre(4)
a.fd = Arbre(0)
a.fd.fd = Arbre(7)
```

Écrire une fonction récursive `taille` prenant en paramètre une instance `a` de la classe `Arbre` et qui renvoie la taille de l'arbre que cette instance implémente.

Écrire de même une fonction récursive `hauteur` prenant en paramètre une instance `a` de la classe `Arbre` et qui renvoie la hauteur de l'arbre que cette instance implémente.

Si un arbre a un seul nœud, sa taille et sa hauteur sont égales à 1. S'il est vide, sa taille et sa hauteur sont égales à 0.

Tester les deux fonctions sur l'arbre représenté ci-dessous :



exercice083

Exercice

La méthode `insert` de la classe `list` permet d'insérer un élément dans une liste à un `indice` donné.

Le but de cet exercice est, *sans utiliser cette méthode*, d'écrire une fonction `ajoute` réalisant cette insertion en produisant une nouvelle liste.

Cette fonction `ajoute` prend en paramètres trois variables `indice`, `element` et `liste` et renvoie une liste `L` dans laquelle les éléments sont ceux de la liste `liste` avec, en plus, l'élément `element` à l'indice `indice`.

On considère que les variables `indice` et `element` sont des entiers positifs et que les éléments de `liste` sont également des entiers positifs.

Les éléments de la liste `liste`, dont les indices sont supérieurs ou égaux à `indice` apparaissent décalés vers la droite dans la liste `L`.

Si `indice` est supérieur ou égal au nombre d'éléments de la liste `liste`, l'élément element est ajouté dans `L` après tous les éléments de la liste `liste`.

Exemple :

```
>>> ajoute(1, 4, [7, 8, 9])
[7, 4, 8, 9]
>>> ajoute(3, 4, [7, 8, 9])
[7, 8, 9, 4]
>>> ajoute(4, 4, [7, 8, 9])
[7, 8, 9, 4]
```

Compléter et tester le code ci-dessous :

```
def ajoute(indice, element, liste):
    nbre_elts = len(liste)
    L = [0 for i in range(nbre_elts + 1)]
    if ...:
        for i in range(indice):
            L[i] = ...
```

```
        L[...] = ...
        for i in range(indice + 1, nbre_elts + 1):
            L[i] = ...
    else:
        for i in range(nbre_elts):
            L[i] = ...
        L[...] = ...
    return L
```

exercice084

Exercice

L'opérateur « ou exclusif » entre deux bits renvoie 0 si les deux bits sont égaux et 1 s'ils sont différents. Il est symbolisé par le caractère $\oplus$. Ainsi :

- $0 \oplus 0 = 0$
- $0 \oplus 1 = 1$
- $1 \oplus 0 = 1$
- $1 \oplus 1 = 0$

On représente ici une suite de bits par un tableau contenant des 0 et des 1.
Exemples :

```
a = [1, 0, 1, 0, 1, 1, 0, 1]
b = [0, 1, 1, 1, 0, 1, 0, 0]
c = [1, 1, 0, 1]
d = [0, 0, 1, 1]
```

Écrire la fonction `ou_exclusif` qui prend en paramètres deux tableaux de même longueur et qui renvoie un tableau où l'élément situé à position `i` est le résultat, par l'opérateur « ou exclusif », des éléments à la position `i` des tableaux passés en paramètres.
En considérant les quatre exemples ci-dessus, cette fonction donne :

```
>>> ou_exclusif(a, b)
[1, 1, 0, 1, 1, 0, 0, 1]
>>> ou_exclusif(c, d)
[1, 1, 1, 0]
```

exercice085

Exercice

Dans cet exercice, on appelle carré d'ordre n un tableau de n lignes et n colonnes dont chaque case contient un entier naturel.

Exemples :

1	7
7	1

c2

Un carré d'ordre 2

3	4	5
4	4	4
5	4	3

c3

Un carré d'ordre 3

2	9	4
7	0	3
6	1	8

c3bis

Un autre carré d'ordre 3

Un carré est dit semimagique lorsque les sommes des éléments situés sur chaque ligne, chaque colonne sont égales.

- Ainsi c2 et c3 sont semimagiques car la somme de chaque ligne, chaque colonne et chaque diagonale est égale à 8 pour c2 et 12 pour c3.
- Le carre c3bis n'est pas semimagique car la somme de la première ligne est égale à 15 alors que celle de la deuxième ligne est égale à 10.

La classe `Carre` ci-après contient des méthodes qui permettent de manipuler des carrés.

- La méthode constructeur crée un carré sous forme d'un tableau à deux dimensions à partir d'une liste d'entiers, et d'un ordre.
- La méthode `affiche` permet d'afficher le carré créé.

Exemple :

```
>>> liste = (3, 4, 5, 4, 4, 4, 5, 4, 3)
>>> c3 = Carre(liste, 3)
>>> c3.affiche()
[3, 4, 5]
[4, 4, 4]
[5, 4, 3]
```

Compléter la méthode `est_semimagique` qui renvoie `True` si le carré est semimagique, `False` sinon. Puis tester la fonction `est_semimagique` sur les carrés c2, c3 et c3bis.

```
class Carre:
    def __init__(self, liste, n):
        self.ordre = n
        self.tableau = [[liste[i + j * n] for i in range(n)] for j in range(n)]

    def affiche(self):
        '''Affiche un carré'''
        for i in range(self.ordre):
            print(self.tableau[i])

    def somme_ligne(self, i):
        '''Calcule la somme des valeurs de la ligne i'''
        somme = 0
        for j in range(self.ordre):
            somme = somme + self.tableau[i][j]
        return somme

    def somme_col(self, j):
        '''Calcule la somme des valeurs de la colonne j'''
        somme = 0
        for i in range(self.ordre):
            somme = somme + self.tableau[i][j]
        return somme

    def est_semimagique(self):
        s = self.somme_ligne(0)

        #test de la somme de chaque ligne
        for i in range(...):
            if ... != s:
                return ...
```

```
        #test de la somme de chaque colonne
        for j in range(...):
            if ... != s:
                return ...

        return ...
```

Listes permettant de générer les carrés c2, c3 et c3bis :

```
lst_c2 = [1, 7, 7, 1]
lst_c3 = [3, 4, 5, 4, 4, 4, 5, 4, 3]
lst_c3bis = [2, 9, 4, 7, 0, 3, 6, 1, 8]
```

exercice086

Exercice

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la dernière occurrence de `elt` dans `tab` si `elt` est dans `tab` et `-1` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(1, [1, 0, 42, 7])
0
>>> recherche(1, [1, 50, 1])
2
>>> recherche(1, [8, 1, 10, 1, 7, 1, 8])
5
```

exercice087

Exercice

On définit une classe gérant une adresse IPv4.

On rappelle qu'une adresse IPv4 est une adresse de longueur 4 octets, notée en décimale à point, en séparant chacun des octets par un point. On considère un réseau privé avec une plage d'adresses IP de `192.168.0.0` à `192.168.0.255`.

On considère que les adresses IP saisies sont valides.

Les adresses IP `192.168.0.0` et `192.168.0.255` sont des adresses réservées.

Le code ci-dessous implémente la classe `AdresseIP`.

```
class AdresseIP:
    def __init__(self, adresse):
        self.adresse = ...

    def liste_octet(self):
        """renvoie une liste de nombres entiers,
        la liste des octets de l'adresse IP"""
        return [int(i) for i in self.adresse.split(".")]

    def est_reservee(self):
        """renvoie True si l'adresse IP est une adresse
        réservée, False sinon"""
        return ... or ...

    def adresse_suivante(self):
        """renvoie un objet de AdresseIP avec l'adresse
        IP qui suit l'adresse self
        si elle existe et False sinon"""
        if ... < 254:
            octet_nouveau = ... + ...
            return AdresseIP('192.168.0.' + ...)
```

```
        else:
            return False
```

Compléter le code ci-dessus et instancier trois objets : `adresse1`, `adresse2`, `adresse3` avec respectivement les arguments suivants : `192.168.0.1`, `192.168.0.2`, `192.168.0.0`

Vérifier que :

```
>>> adresse1.est_reservee()
False
>>> adresse3.est_reservee()
True
>>> adresse2.adresse_suivante().adresse
'192.168.0.3'
```

exercice088

Exercice

La classe ABR ci-dessous permet d'implémenter une structure d'arbre binaire de recherche.

```
class Noeud:
    def __init__(self, valeur):
        """Méthode constructeur pour la classe Noeud.
        Paramètre d'entrée : valeur (str)"""
        self.valeur = valeur
        self.gauche = None
        self.droit = None

    def getValeur(self):
        """Méthode accesseur pour obtenir la valeur du noeud
        Aucun paramètre en entrée"""
        return self.valeur

    def droitExiste(self):
        """Méthode renvoyant True si l'enfant droit existe
        Aucun paramètre en entrée"""
        return (self.droit is not None)

    def gaucheExiste(self):
        """Méthode renvoyant True si l'enfant gauche existe
        Aucun paramètre en entrée"""
        return (self.gauche is not None)

    def inserer(self, cle):
        """Méthode d'insertion de clé dans un arbre binaire de recherche
        Paramètre d'entrée : cle (int)"""
        if cle < ...:
```

```
            # on insère à gauche
            if self.gaucheExiste():
                # on descend à gauche et on retente l'insertion de la clé
                ...
            else:
                # on crée un fils gauche
                self.gauche = ...
        elif cle > ... :
            # on insère à droite
            if ... :
                # on descend à droite et on retente l'insertion de la clé
                ...
            else:
                # on crée un fils droit
                ... = Noeud(cle)
```

Compléter la fonction récursive `inserer` afin qu'elle permette d'insérer un nœud dans l'arbre binaire de recherche proposé, à l'aide de sa clé.

Voici un exemple d'utilisation :

```
>>> arbre = Noeud(7)
>>> for cle in (3, 9, 1, 6):
...     arbre.inserer(cle)
>>> arbre.gauche.getValeur()
3
>>> arbre.droit.getValeur()
9
>>> arbre.gauche.gauche.getValeur()
1
>>> arbre.gauche.droit.getValeur()
6
```