BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Mardi 14 janvier 2025 (après-midi)

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice et du dictionnaire n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 8 pages numérotées de 1/8 à 8/8.

Les 3 exercices proposés sont indépendants.

Exercice 1 (6 points)

Cet exercice porte sur les arbres et la programmation orientée objet.

Une agence immobilière développe un programme pour gérer les biens immobiliers qu'elle propose à la vente.

Dans ce programme, pour modéliser les données de biens immobiliers, on définit une classe `Bim` avec les attributs suivants :

- `nt` de type `str` représente la nature du bien (appartement, maison, bureau, commerces, ...) ;
- `sf` de type `float` est la surface du bien ;
- `pm` de type `float` est le prix moyen par m² du bien qui dépend de son emplacement.

La classe `Bim` possède une méthode `estim_prix` qui renvoie une estimation du prix du bien. Le code (incomplet) de la classe `Bim` est donné ci-dessous :

```
class Bim:
    def __init__(self, nature, surface, prix_moy):
        ...
    def estim_prix(self):
        return self.sf * self.pm
```

1. Recopier et compléter le code du constructeur de la classe `Bim`.

2. On exécute l'instruction suivante :

   ```
   b1 = Bim('maison', 70.0, 2000.0)
   ```
   Que renvoie l'instruction `b1.estim_prix()` ? Préciser le type de la valeur renvoyée.

3. On souhaite affiner l'estimation du prix d'un bien en prenant en compte sa nature :
   - pour un bien dont l'attribut `nt` est `'maison'` la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 1,1 ;
   - pour un bien dont l'attribut `nt` est `'bureau'` la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 0,8 ;
   - pour les biens d'autres natures, l'estimation du prix ne change pas.

   Modifier le code de la méthode `estim_prix` afin de prendre en compte ce changement de calcul.

4. Écrire le code Python d'une fonction `nb_maison(lst)` qui prend en argument une liste Python de biens immobiliers de type `Bim` et qui renvoie le nombre d'objets de nature `'maison'` contenus dans la liste `lst`.

5. Pour une recherche efficace des biens immobiliers selon le critère de leur surface, on stocke les objets de type `Bim` dans un arbre binaire de recherche, nommé `abr`. Pour tout nœud de cet arbre :

- tous les objets de son sous-arbre gauche ont une surface inférieure ou égale à la surface de l'objet contenue dans ce nœud ;
- tous les objets de son sous-arbre droit ont une surface strictement supérieure à la surface de l'objet contenue dans ce nœud.

L'objet `abr` dispose des méthodes suivantes :

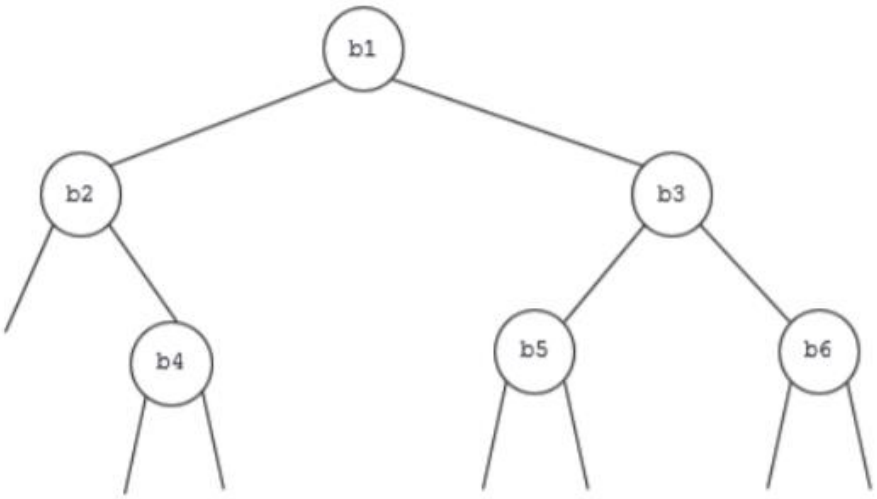
`abr.est_vide()` : renvoie `True` si `abr` est vide et `False` sinon.

`abr.get_v()` : renvoie l'élément (de type `Bim`) situé à la racine de `abr` si `abr` n'est pas vide et `None` sinon.

`abr.get_g()` : renvoie le sous-arbre gauche de `abr` si `abr` n'est pas vide et `None` sinon.

`abr.get_d()` : renvoie le sous-arbre droit de `abr` si `abr` n'est pas vide et `None` sinon.

a. Dans cette question, on suppose que l'arbre binaire `abr` a la forme ci-dessous :



Donner la liste les biens `b1`, `b2`, `b3`, `b4`, `b5`, `b6` triée dans l'ordre croissant de leur surface.

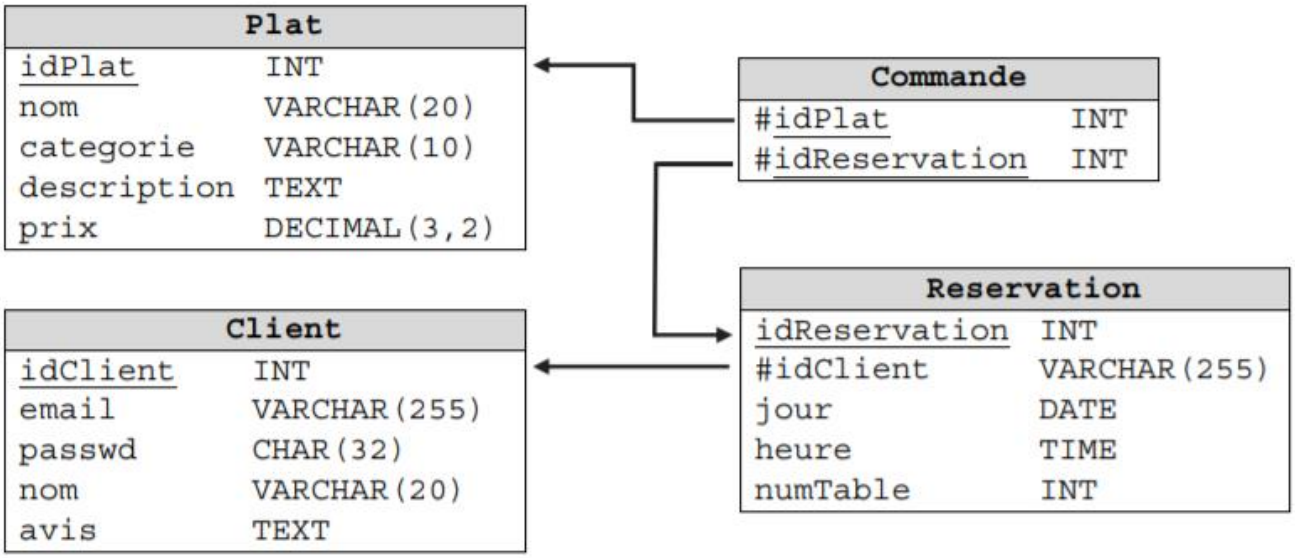
b. Recopier et compléter le code de la fonction récursive `contient` donnée ci-dessous, qui prend en arguments un nombre `surface` de type `float` et un arbre binaire de recherche `abr` contenant des éléments de type `Bim` ordonnés selon leur attribut de surface `sf`. La fonction `contient(surface, abr)` renvoie `True` s'il existe un bien dans `abr` d'une surface supérieure ou égale à `surface` et `False` sinon.

```
def contient(surface, abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= ..... :
        return True
    else:
        return contient( surface , ..... )
```

Exercice 2 (6 points)

Cet exercice porte sur les bases de données relationnelles.

Une restauratrice a mis en place un site Web pour gérer ses réservations en ligne. Chaque client peut s'inscrire en saisissant ses identifiants. Une fois connecté, il peut effectuer une réservation en renseignant le jour et l'heure. Il peut également commander son menu en ligne et écrire un avis sur le restaurant.
Le gestionnaire du site Web a créé une base de données associée au site nommée `restaurant`, contenant les quatre relations du schéma relationnel ci-dessous :



Dans le schéma relationnel précédent, un attribut souligné indique qu'il s'agit d'une clé primaire. Un attibut précédé du symbole # indique qu'il s'agit d'une clé étrangère et la flèche associée indique l'attribut référencé. Ainsi, par exemple, l'attribut `idPlat` de la relation `Commande` est une clé étrangère qui fait référence à l'attribut `idPlat` de la relation `Plat`.

Dans la suite, les mots clés suivants du langage SQL pourront être utilisés dans les requêtes :

`SELECT, FROM, WHERE, JOIN, ON, DELETE, UPDATE, SET, INSERT INTO, AND, OR.`

1. Parmi les trois requêtes suivantes, écrites dans le langage SQL, laquelle renvoie les valeurs de tous les attributs des plats de la catégorie `'entrée'` :

R1 :
```
SELECT nom, prix
FROM Plat
WHERE categorie  = 'entrée';
```

R2 :
```
SELECT *
FROM Plat
WHERE categorie  = 'entrée';
```

R3 :
```
UPDATE Plat
SET categorie  = 'entrée'
WHERE 1;
```

2. Écrire, dans le langage SQL, des requêtes d'interrogation sur la base de données `restaurant` permettant de réaliser les tâches suivantes :

   a. Afficher les noms et les avis des clients ayant effectué une réservation pour la date du `'2021-06-05'` à l'heure `'19:30:00'`.
   b. Afficher le nom des plats des catégories `'plat principal'` et `'dessert'`, correspondant aux commandes de la date `'2021-04-12'`.

3. Que réalise la requête SQL suivante ?

```
INSERT INTO Plat
VALUES(58,'Pêche Melba', 'dessert', 'Pêches et glace vanille', 6.5);
```

4. Écrire des requêtes SQL permettant de réaliser les tâches suivantes :

   a. Supprimer les commandes ayant comme `idReservation` la valeur `2047`.

   b. Augmenter de 5% tous les prix de la relation `plat` strictement inférieurs à `20.00`.

Exercice 3 (8 points)

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.

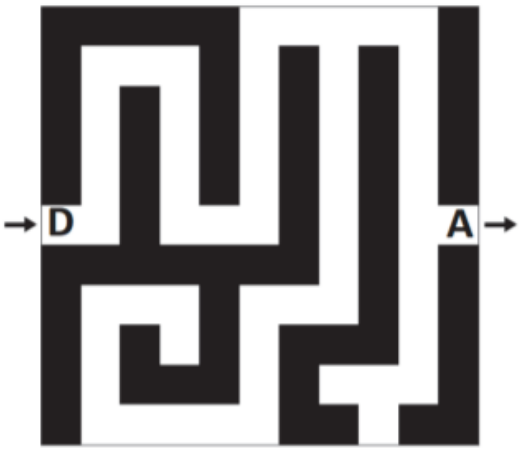
Partie A : Représentation d'un labyrinthe

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 :

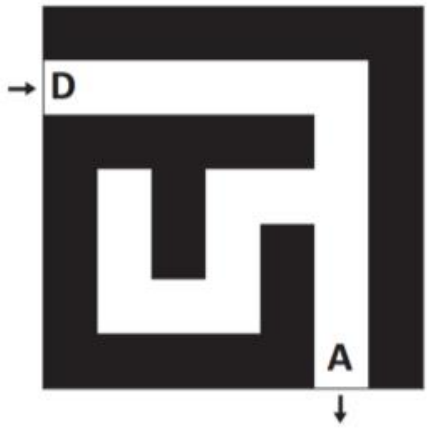
- 0 représente une case vide, hors case de départ et arrivée,
- 1 représente un mur,
- 2 représente le départ du labyrinthe,
- 3 représente l'arrivée du labyrinthe.

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],
        [1, 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. Compléter la fonction `depart(lab)` ci-dessous 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, l'appel `depart(lab1)` doit renvoyer le tuple `(5, 0)`.

```
def depart(lab) :
    n = len(lab)
    m = len(lab[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, l'appel `nb_cases_vides(lab2)` doit renvoyer la valeur `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.
On considère 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, l'appel `voisines(1, 1, [[1, 1, 1], [4, 0, 0], [1, 0, 1]])` renvoie la liste `[(2, 1), (1, 2)]`.

   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. Recopier et compléter la 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`.

```
def solution(lab):
    chemin = [depart(lab)]
    case = chemin[0]
    i = case[0]
    j = case[1]
    ...
```

Par exemple, l 'appel `solution(lab2)` doit renvoyer `[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5)]`.