BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

Décembre 2021 – Bac Blanc

N.S.I.
Numérique et Sciences Informatiques

Durée de l'épreuve : **2 heures 20**

L'usage de la calculatrice n'est pas autorisé

Le candidat traite au choix 2 exercices parmi les 3 exercices proposés.

Chaque exercice est noté sur 5 points

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.
Ce sujet comporte 9 pages numérotées de 1 à 9

Exercice 1 : *(5 points)*

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

L'énoncé de cet exercice utilise les mots du langage SQL suivant :

`SELECT`, `FROM`, `WHERE`, `JOIN`, `INSERT INTO`, `VALUES`, `ORDER BY`

On rappelle qu'en SQL la clause `ORDER BY`, suivie d'un attribut, permet de classer les résultats par ordre croissant de l'attribut.
Dans un lycée, le parc informatique est constitué d'ordinateurs, d'imprimantes, de vidéoprojecteurs et de TNI (Tableau Numérique Interactif).
Tous les ordinateurs et toutes les imprimantes sont connectés au réseau de l'établissement.
Chaque salle de cours est identifiée par un numéro unique et contient :

- un ou plusieurs ordinateurs reliés au réseau de l'établissement ;
- aucun ou un seul vidéoprojecteur ;
- s'il y a un vidéoprojecteur, aucun ou un seul TNI ;
- une ou plusieurs imprimantes réseau.

Un ordinateur peut être connecté via le réseau à une ou plusieurs imprimantes (situées éventuellement dans une ou plusieurs autres salles) et à un vidéoprojecteur avec TNI ou non.
Les ordinateurs et les imprimantes possèdent un nom unique sur le réseau de l'établissement.
Les vidéoprojecteurs ne sont pas connectés au réseau, ils ne possèdent pas de nom unique. Ils sont donc identifiés par le numéro de la salle où ils sont installés.
Tous ces matériels sont gérés à l'aide d'une base de données relationnelle qui comprend 3 relations (ou tables) nommées : `Ordinateur`, `Videoprojecteur`, `Imprimante`.

On donne ci dessous le schéma relationnel de la relation `Ordinateur`, suivi d'un extrait de la relation `Ordinateur`. La clé primaire est soulignée.

`Ordinateur`(`nom_ordi` : String, `salle` : String, `marque_ordi` : String, `modele_ordi` : String, `annee` : Int, `video` : Boolean)

On distingue deux types d'ordinateurs :

- les ordinateurs multimédias pour les logiciels généraux avec un nom commençant par le groupe de lettres Gen.
- les ordinateurs techniques pour les logiciels qui demandent plus de ressources avec un nom commençant par le groupe de lettres Tech.

Ce groupe de lettres est suivi d'un tiret et du numéro unique de l'ordinateur pour chaque type.

Les 5 premières lignes de la relation `Ordinateur`

nom_ordi	salle	marque_ordi	modele_ordi	annee	video
Gen-24	012	HP	compaq pro 6300	2012	true
Tech-62	114	Lenovo	p300	2015	true
Gen-132	223	Dell	Inspiron Compact	2019	true
Gen-133	223	Dell	Inspiron Compact	2019	false
Gen-134	223	Dell	Inspiron Compact	2019	false

1. (a) À l'aide d'un système de gestion de base de données, on envoie au serveur la requête SQL suivante :

SELECT `salle`, `marque_ordi` FROM `Ordinateur` ;

Quel résultat produit cette requête sur l'extrait de la relation `Ordinateur` donné ci-dessus ?

(b) Quel résultat produit la requête suivante sur l'extrait de la relation `Ordinateur` donné ci-dessus ?

SELECT `nom_ordi`, `salle` FROM `Ordinateur` WHERE `video` = true ;

2. Écrire une requête SQL donnant tous les attributs des ordinateurs correspondant aux années supérieures ou égales à 2017 ordonnées par dates croissantes.
3. (a) Pour quelle raison l'attribut `salle` ne peut-il pas être une clé primaire pour la relation `Ordinateur` ?

(b) On donne ci-dessous un extrait de la relation `Imprimante` :

Les 5 premières lignes de la relation `Imprimante`

nom_imprimante	marque_imp	modele_imp	salle	nom_ordi
imp_BTS_NB	HP	Laserjet pro M15w	114	Tech-62
imp_BTS_Couleur	Canon	Megatank Pixma G5050	114	Tech-62
imp_salle-info1	Brother	2360DN	223	Gen-132
imp_salle-info1	Brother	2360DN	223	Gen-133
imp_salle-info1	Brother	2360DN	223	Gen-134

On prend `(nom_imprimante, nom_ordi)` comme clé primaire. Écrire le schéma relationnel de la relation `Imprimante` en précisant les éventuelles clés étrangères pour les autres relations.

4. On donne ci-dessous un extrait de la relation `Videoprojecteur` :

Les 4 premières lignes de la relation `Videoprojecteur`

salle	marque_video	modele_video	tni
012	Epson	xb27	true
114	Sanyo	PLV-Z3	false
223	Optoma	HD143X	false
225	Optoma	HD143X	true

(a) Écrire une requête SQL pour ajouter à la relation `Videoprojecteur` le vidéoprojecteur nouvellement installé en salle 315 de marque NEC, modèle ME402X et non relié à un TNI.

(b) Écrire une requête SQL permettant de récupérer les attributs `salle, nom_ordi, marque_video` des ordinateurs connectés à un vidéoprojecteur équipé d'un TNI.

Exercice 2 : *(5 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.

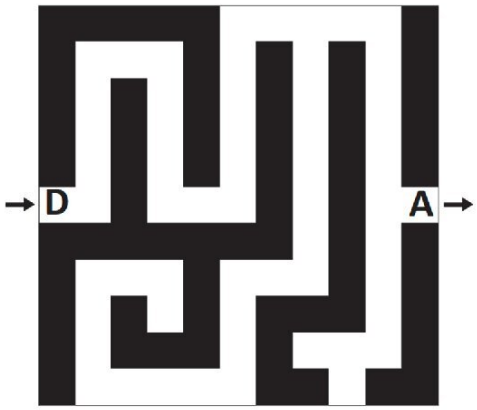
Partie A : Représentation d'un labyrinthe

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 :

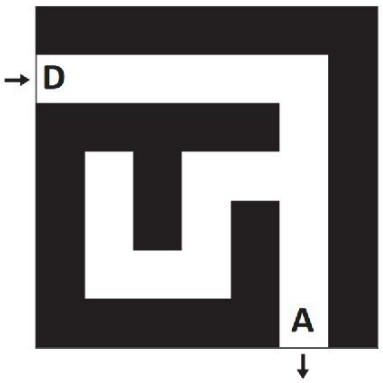
- 0 représente une case vide, hors case de départ et arrivée,
- 1 représente un mur,
- 2 représente le départ du labyrinthe,
- 3 représente l'arrivée du labyrinthe.

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)]`.

Exercice 3 : *(5 points)*

Notion abordée : structures de données (dictionnaires)

Une ville souhaite gérer son parc de vélos en location partagée. L'ensemble de la flotte de vélos est stocké dans une table de données représentée en langage Python par un dictionnaire contenant des associations de type `id_velo : dict_velo` où `id_velo` est un nombre entier compris entre 1 et 199 qui correspond à l'identifiant unique du vélo et `dict_velo` est un dictionnaire dont les clés sont : `"type"`, `"etat"`, `"station"`.

Les valeurs associées aux clés `"type"`, `"etat"`, `"station"` de `dict_velo` sont de type chaînes de caractères ou nombre entier :

- `"type"` : chaîne de caractères qui peut prendre la valeur `"electrique"` ou `"classique"`
- `"état"` : nombre entier qui peut prendre la valeur 1 si le vélo est disponible, 0 si le vélo est en déplacement, -1 si le vélo est en panne
- `"station"` : chaînes de caractères qui identifie la station où est garé le vélo.

Dans le cas où le vélo est en déplacement ou en panne, `"station"` correspond à celle où il a été dernièrement stationné.

Voici un extrait de la table de données :

```
flotte = {
     12 : {"type" : "electrique", "etat" : 1, "station" : "Prefecture"},
     80 : {"type" : "classique", "etat" : 0, "station" : "Saint-Leu"},
     45 : {"type" : "classique", "etat" : 1, "station" : "Baraban"},
     41 : {"type" : "classique", "etat" : -1, "station" : "Citadelle"},
     26 : {"type" : "classique", "etat" : 1, "station" : "Coliseum"},
     28 : {"type" : "electrique", "etat" : 0, "station" : "Coliseum"},
     74 : {"type" : "electrique", "etat" : 1, "station" : "Jacobins"},
     13 : {"type" : "classique", "etat" : 0, "station" : "Citadelle"},
     83 : {"type" : "classique", "etat" : -1, "station" : "Saint-Leu"},
     22 : {"type" : "electrique", "etat" : -1, "station" : "Joffre"}
  }
```

`flotte` étant une variable globale du programme.

Toutes les questions de cet exercice se réfèrent à l'extrait de la table `flotte` fourni ci-dessus. **L'annexe 1 présente** un rappel sur les dictionnaires en langage Python.

1.

1.a. Que renvoie l'instruction `flotte[26]` ?

1.b. Que renvoie l'instruction `flotte[80]["etat"]` ?

1.c. Que renvoie l'instruction `flotte[99]["etat"]` ?

2. Voici le script d'une fonction :

```
def proposition(choix):
  for v in flotte:
    if flotte[v]["type"] == choix and flotte[v]["etat"] == 1:
      return flotte[v]["station"]
```

2.a. **Quelles sont** les valeurs possibles de la variable `choix` ?

2.b. **Expliquer** ce que renvoie la fonction lorsque l'on choisit comme paramètre l'une des valeurs possibles de la variable `choix`.

3.

3.a. Écrire un script en langage Python qui affiche les identifiants (`id_velo`) de tous les vélos disponibles à la station "`Citadelle`".

3.b. Écrire un script en langage Python qui permet d'afficher l'identifiant (`id_velo`) et la station de tous les vélos électriques qui ne sont pas en panne.

4. On dispose d'une table de données des positions GPS de toutes les stations, dont un extrait est donné ci-dessous. Cette table est stockée sous forme d'un dictionnaire.

Chaque élément du dictionnaire est du type:

'nom de la station' : (latitude, longitude)

```
stations =  {
'Prefecture' : (49.8905, 2.2967) ,
'Saint-Leu' : (49.8982, 2.3017),
'Coliseum' :  (49.8942, 2.2874),
'Jacobins' : (49.8912, 2.3016)
}
```

On **admet** que l'on dispose d'une fonction `distance(p1, p2)` permettant de renvoyer la distance en mètres entre deux positions données par leurs coordonnées GPS (latitude et longitude).

Cette fonction prend en paramètre deux tuples représentant les coordonnées des deux positions GPS et renvoie un nombre entier représentant cette distance en mètres.

Par exemple, `distance((49.8905, 2.2967), (49.8912, 2.3016))` renvoie `9591`

Écrire une fonction qui prend en paramètre les coordonnées GPS de l'utilisateur sous forme d'un tuple et qui renvoie, pour chaque station située à moins de 800 mètres de l'utilisateur :

- le nom de la station ;
- la distance entre l'utilisateur et la station ;
- les identifiants des vélos disponibles dans cette station.

Une station où aucun vélo n'est disponible ne doit pas être affichée.

Annexe 1

Action	Instruction et syntaxe
Créer un dictionnaire vide	`dico={}`
Obtenir un élément d'un dictionnaire existant à partir de sa clé renvoie une erreur si `cle` n'existe pas dans le dictionnaire	`dico[cle]`
Modifier la valeur d'un élément d'un dictionnaire à partir de sa clé	`dico[cle]=nouvelle_valeur`
Ajouter un élément dans un dictionnaire existant	`dico[nouvelle_cle]=valeur`
Supprimer et obtenir un élément d'un dictionnaire à partir de sa clé	`dico.pop(cle)`
Tester l'appartenance d'un élément à un dictionnaire (renvoie un booléen)	`cle in dico`
Objet itérable contenant les clés. ce n'est pas un objet de type list	`dico.keys()`
Objet itérable contenant les valeurs. ce n'est pas un objet de type list	`dico.values()`
Objet itérable contenant les couples (clé,valeur)	`dico.items()`
Afficher les associations cle:valeur du dictionnaire `dico`	`for cle in dico:` `  print(cle, dico[cle])`