BACCALAURÉAT BLANC
SESSION DE DÉCEMBRE 2023

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES
Partie écrite

Classe terminale de la voie générale

Épreuve 1

DURÉE DE L'ÉPREUVE : 3 heures 30 minutes

Le sujet comporte 11 pages numérotées de 1 à 11, ainsi qu'une annexe, ajoutée à la fin du sujet. Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Il est rappelé que la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies.

L'usage de la calculatrice est interdit.

1 Nuage (6 points)

Cet exercice porte sur l'algorithmique, la programmation orientée objet et la récursivit

L'objectif de cet exercice est de trouver les deux points les plus proches dans un nuage de points pour lesquels on connaît les coordonnées dans un repère orthogonal. On rappelle que la distance entre deux points A et B de coordonnées $(xA;yA)$ et $(xB;yB)$ est donnée par la formule :

$$AB = \sqrt{(xB - xA)^2 + (yB - yA)^2}$$

Les coordonnées d'un point seront stockées dans un tuple de deux nombres réels. Le nuage de points sera représenté en Python par une liste de tuples de taille n, n étant le nombre total de points. On suppose qu'il n'y a pas de points confondus (mêmes abscisses et mêmes ordonnées) et qu'il y a au moins deux points dans le nuage.

Pour calculer la racine carrée, on utilisera la fonction `sqrt` du module `math`, pour rappel :

```
1 >>> from math import sqrt
2 >>> sqrt(16)
3 4.0
```

1. Cette partie comprend plusieurs questions générales :

a. Donner le rôle de l'instruction de la ligne 1 du code précédent.

b. Expliquer le résultat suivant :

```
1 >>> 0.1 + 0.2 == 0.3
2 False
```

c. Expliquer l'erreur suivante :

```
1     >>> point_A = (3, 4)
2     >>> point_A[0]
3     3
4     >>> point_A[0] = 2
5     Traceback (most recent call last):
6     File "<console>", line 1, in <module>
7     TypeError : 'tuple' object does not support item assignment
```

2. On définit la classe `Segment` ci-dessous :

```
1 from math import sqrt
2 class Segment:
3     def __init__(self, point1, point2):
4         self.p1 = point1
5         self.p2 = point2
6         self.longueur = .............. # à compléter
```

a. Recopier et compléter la ligne 6 du constructeur de la classe `Segment`.

La fonction `liste_segments` donnée ci-dessous prend en paramètre une liste de points et renvoie une liste contenant des objets `Segment` qu'il est possible de construire à partir de ces points. On considère les segments [AB] et [BA] comme étant confondus et ajoutera un seul objet dans la liste.

```
1 def liste_segments(liste_points):
2     n = len(liste_points)
3     segments = []
4     for i in range(..............):
5         for j in range(.............., n):
6             # On construit le segment à partir des points i et j.
7             seg = ..........................................
8             segments.append(seg) # On l'ajoute à la liste
9     return segments
```

b. Recopier la fonction sans les commentaires et compléter le code manquant.

c. Donner en fonction de `n` la longueur de la liste `segments`. Le résultat peut être laissé sous la forme d'une somme.

d. Donner, en fonction de `n`, la complexité en temps de la fonction `liste_segments`.

3. L'objectif de cette partie est d'écrire la fonction de recherche des deux points les plus proches.

On dispose de deux fonctions : `moitie_gauche` (respectivement `moitie_droite`) qui prennent en paramètre une liste et qui renvoient chacune une nouvelle liste contenant la moitié gauche (respectivement la moitié droite) de la liste de départ. Si le nombre d'éléments de celle-ci est impair, l'élément du center se trouve dans la partie gauche.

Exemples :

```
>>> liste = [1, 2, 3, 4]
>>> moitie_gauche(liste)
[1, 2]
>>> moitie_droite(liste)
[3, 4]

>>> liste = [1, 2, 3, 4, 5]
>>> moitie_gauche(liste)
[1, 2, 3]
>>> moitie_droite(liste)
[4, 5]
```

Écrire la fonction `plus_court_segment` qui prend en paramètre une liste d'objets `Segment` et renvoie l'objet `Segment` dont la longueur est la plus petite.

On procédera de la façon suivante :

- Tester si le cas de base est atteint, c'est-à-dire lorsque la liste contient un seul segment ;
- Découper la liste en deux listes de tailles égales (à une unité près) ;
- Appeler récursivement la fonction pour rechercher le minimum dans chacune des deux listes ;
- Comparer les deux valeurs récupérées et renvoyer la plus petite des deux

4. On considère les trois points $A(3;4)$, $B(2;3)$ et $C(-3;-1)$.

a. Donner l'instruction Python permettant de construire la variable `nuage_points` contenant les trois points A, B et C.

b. En utilisant les fonctions de l'exercice, écrire les instructions Python qui affichent les coordonnées des deux points les plus proches du nuage de points `nuage_points`.

2 Forrage (6 points)

`Cet exercice porte sur la notion de listes et la récursivité`

On souhaite créer un puits dans des zones de terrain instable en forant un conduit dans le sol tout en préservant l'intégrité du terrain. Pour illustrer cette situation, nous envisageons de commencer le forage à partir de la surface, en progressant vers le bas en choisissant de déplacer le forage vers la gauche ou la droite à chaque niveau, jusqu'à atteindre le niveau de la nappe phréatique.

Le sol peut être modélisé par une pyramide d'entiers, où chaque entier représente le niveau de confiance dans le forage de la zone correspondante. La pyramide est présentée dans la figure 1, avec des flèches indiquant les différents déplacements possibles lors du forage.

Le but est de créer un conduit partant du sommet de la pyramide et descendant jusqu'au niveau le plus bas, où se trouve l'eau, en effectuant des déplacements élémentaires, c'est-à-dire en choisissant de descendre vers la gauche ou la droite à chaque niveau. Le score de confiance d'un conduit est la somme des nombres rencontrés le long de ce conduit. Le conduit gris représenté à droite dans la figure 1 a un score de confiance de 4+2+5+1+3=15.

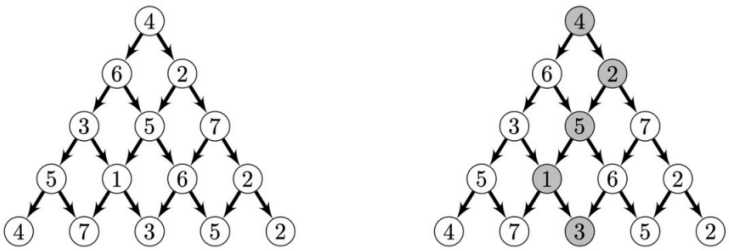

Figure 1.

On va utiliser un ordinateur pour chercher à résoudre ce problème. Pour cela, on représente chaque niveau par la liste des nombres de ce niveau et une pyramide par une liste de niveaux.

La pyramide ci-dessus est donc représentée par la liste de listes:

`ex1 = [[4],[6,2],[3,5,7],[5,1,6,2],[4,7,3,5,2]].`

1. Dessiner la pyramide représentée par la liste de listes `ex2 = [[4],[2,1],[4,5,9],[3,6,2,1]]`.

2. Déterminer un conduit de score de confiance maximal dans la pyramide `ex2` et donner son score.

L'objectif est de déterminer le score de confiance maximal possible pour une pyramide donnée. Une approche initiale consiste à énumérer tous les conduits et à calculer leur score pour identifier les meilleurs.

3. Énumérez les conduits pour la pyramide à trois niveaux représentée dans la figure 2.

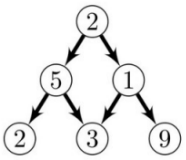

Figure 2.

Afin de compter le nombre de conduits pour une pyramide de `n` niveaux, on remarque qu'un conduit est représenté par une séquence de `n` déplacements à gauche ou à droite.

4. Déterminez le nombre de conduits dans une pyramide de n niveaux. Vous pourrez utiliser un codage binaire pour les conduits, où gauche est représenté par 0 et droite par 1.

5. Justifiez pourquoi la solution consistant à tester tous les conduits possibles pour calculer le score de confiance maximal d'une pyramide n'est pas raisonnable.

On définira par la suite un conduit comme maximal si son score de confiance est le plus élevé. Pour calculer efficacement le score maximal, il est possible d'analyser la structure des conduits maximaux.

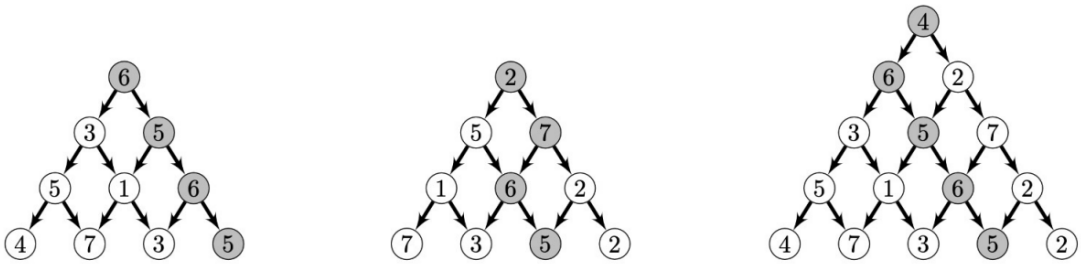

Figure 3.

- Première observation : Si on a des conduits maximaux `cm1` et `cm2` (représentés en gris dans la figure 3) pour les deux pyramides obtenues en enlevant le sommet de `ex1`, on peut obtenir un conduit maximal en ajoutant le sommet 4 devant le conduit ayant le score le plus élevé parmi `cm1` et `cm2`. Ici, le score de `cm1` est 6+5+6+5=22, et le score de `cm2` est 2+7+6+5=20. Ainsi, le conduit maximal dans `ex1` est celui obtenu à partir de `cm1` et dessiné à droite dans la figure 3.
- Deuxième observation : Si la pyramide n'a qu'un seul niveau, il n'y a que le sommet. Dans ce cas, il n'y a pas de choix à faire, le seul conduit possible est celui qui contient le sommet, et le nombre associé à ce sommet est le score maximal que l'on peut obtenir.

Avec ces deux observations, il est possible de calculer le score maximal possible pour un conduit dans une pyramide p par récurrence. Définissons `score_max(i, j, p)` comme le score maximal possible depuis le nombre d'indice `j` du niveau `i`, c'est-à-dire dans la petite pyramide issue de ce nombre. Les relations suivantes peuvent alors être établies :

- `score_max(len(p)-1, j, p) = p[len(p)-1][j]`
- `score_max(i, j, p) = p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))`

Le score maximal possible pour p dans son intégralité sera alors `score_max(0, 0, p)`.

6. Écrivez la fonction récursive `score_max` qui implémente les règles précédentes.

Cependant, une résolution stricte de cette définition pourrait conduire à un coût prohibitif en raison de la redondance des calculs. Par exemple `score_max(3,1,p)` va être calculé pour chaque appel à `score_max(2,0,p)` et `score_max(2,1,p)`. Pour éviter cette redondance, on va construire une pyramide s dont le nombre à l'indice `j` du niveau `i` correspond à `score_max(i,j,p)`, c'est-à-dire au score maximal pour un conduit à partir du nombre correspondant dans p.

7. Écrivez une fonction `pyramide_nulle` qui prend en paramètre un entier `n` et construit une pyramide remplie de 0 à `n` niveaux.

8. Complétez la fonction `score_max2` ci-dessous qui prend en paramètre une pyramide p et renvoie le score maximal pour un conduit dans p. Pour cela, construisez une pyramide s remplie de 0 de la même taille et remplissez-la avec les valeurs de `score_max` en commençant par le dernier niveau et en appliquant petit à petit les relations données ci-dessus.

```
1  def score_max2(p):
2      n = len(p)
3      s = ...
4      # Remplissage du dernier niveau
5      for j in ...:
6          s[n-1][j] = ...
7      # Remplissage des autres niveaux
8      for i in ...:
9          for j in ...:
10             s[i][j] = ...
11     # Renvoie du score maximal
12     return s[0][0]
```

9. Montrez que le coût d'exécution de cette fonction est quadratique en `n` (en $O(n^2)$)pour une pyramide à `n` niveaux.

3 Livres (8 points)

`Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL`

Cet exercice se divise en trois parties distinctes. L'objectif est de développer une application dédiée au stockage et au traitement d'informations relatives à des livres de science-fiction. Les données que l'on souhaite enregistrer comprennent les éléments suivants :

- l'identifiant du livre (id) ;
- le titre (titre) ;
- le nom de l'auteur (nom_auteur) ;
- l'année de première publication (ann_pub) ;
- une note sur 10 (note).

Un extrait des informations à consigner est présenté ci-dessous :

figure 1
`livres`

id	titre	auteur	ann pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K. Dick	1953	9
8	Blade Runner	K. Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
1 dico_livres = {
2  'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
3  'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade␣Runner', 'Les␣
     Robots', 'Ravage', 'Chroniques␣martiennes', 'Dragon␣déchu', '
     Fahrenheit␣451'],
4  'auteur' : ['Orwell', 'Herbert', 'Asimov', 'K. Dick', 'K. Dick', 'Asimov'
     , 'Barjavel', 'Bradbury', 'Hamilton', 'Bradbury'],
5  'ann_pub' : [1949, 1965, 1951, 1953,1968, 1950, 1943, 1950, 2003,
     1953],
6  'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
7  }
8 a = dico_livres['id']
9 b = dico_livres['titre'][1]
```

1. Déterminer les valeurs des variables a et b après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
1 def titre_livre(dico, id_livre):
2   for i in range(len(dico['id'])):
3     if dico['id'][i] == ... :
4       return dico['titre'][...]
5   return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.

3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.

4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).

5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Partie B

Dans cette partie, on utilise le paradigme orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
1  class Livre:
2    def __init__(self, id_livre, titre, auteur, ann_pub, note):
3      self.id = id_livre
4      self.titre = titre
5      self.auteur = auteur
6      self.ann_pub = ann_pub
7      self.note = note
8    def get_id(self):
9      return self.id
10   def get_titre(self):
11     return self.titre
12   def get_auteur(self):
13     return self.auteur
14   def get_ann_pub(self):
15     return self.ann_pub
16
17 class Bibliotheque:
```

```
18   def __init__(self):
19     self.liste_livre = []
20   def ajout_livre(self, livre):
21     self.liste_livre.append(livre)
22   def titre_livre(self, id_livre):
23     for livre in self.liste_livre :
24       if ... == id_livre :
25         return ...
26     return ...
```

6. Citer un attribut et une méthode de la classe `Livre`.

7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.

8. Écrire le programme permettant d'ajouter le livre Blade Runner à la fin de la "bibliothèque" en utilisant la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).

9. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`. Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table livres qui est exactement la table de la figure 1.

L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut auteur ne peut pas être choisi comme clé primaire.

11. Donner le résultat renvoyé par la requête SQL suivante :

```
1 SELECT titre
2 FROM livres
3 WHERE auteur = 'Bradbury';
```

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par K. Dick publiés après 1960.

13. Écrire une requête SQL permettant de modifier la note du livre Dune en la passant de 8/10 à 7/10.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type `INT` ;

- `nom` de type `TEXT` ;
- `prenom` de type `TEXT` ;
- `annee_naissance` de type `INT` (année de naissance).

`auteurs`

ID	Nom	Prénom
1	Orwell	George
2	Herbert	Franck
3	Asimov	Isaac
4	K. Dick	Philip
5	Bradbury	Ray
6	Barjavel	René
7	Hamilton	Peter

La table `livres` est aussi modifiée comme suit :

`livres`

id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14. Expliquer l'intérêt d'utiliser deux tables (livres et auteurs) au lieu de regrouper toutes les informations dans une seule table.

15. Expliquer le rôle de l'attribut id_auteur de la table livres.

16. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1955.

17. Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance > 50;
```

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18. Expliquer en quoi la réalisation de ce projet pourrait être problématique.

4 ANNEXE

Types de données	
CHAR(t)	Texte fixe de t caractères.
VARCHAR(t)	Texte de t caractères variables.
TEXT	Texte de 65 535 caractères max.
INT	Nombre entier
FLOAT	Réel à virgule flottante
DATE	Date format AAAA-MM-JJ
DATETIME	Date et heure format AAAA-MM-JJ HH:MI

Quelques exemples de syntaxe SQL :

- Insérer des enregistrements :

```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

Exemple :

```
INSERT INTO client(id_client,nom,prenom) VALUES (112,'Dehnou','Danièle')
```

- Modifier des enregistrements :

```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

- Supprimer des enregistrements :

```
DELETE FROM Table WHERE Selecteur
```

- Sélectionner des enregistrements :

```
SELECT attributs FROM Table WHERE Selecteur
```

- Effectuer une jointure :

```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE
Selecteur
```