BACCALAURÉAT BLANC
SESSION DE FÉVRIER 2023

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES
Partie écrite

Classe terminale de la voie générale

Épreuve 1

Í4 février 2023

DURÉE DE L'ÉPREUVE : 3 heures 30 minutes

Le sujet comporte 6 pages numérotées de 1 à 6, 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 Monnaie locale (4 points)

Cet exercice est un exercice d'algorithmique et de programmation en langage Python. Il aborde la programmation orientée objet.

Les élus d'une agglomération française ont décidé de créer une monnaie locale complémentaire, appelée le "bob", pour inciter les habitants à acheter auprès des commerçants et producteurs locaux.

Le bob est égal à l'euro (1 bob = 1 euro) et le plus petit billet est de 1 bob, il ne peut donc pas être fractionné. Lors de l'achat d'un article, le paiement peut être effectué en bobs ou en euros, si le prix n'est pas entier. Par exemple, un article coûtant 6,20 EUR peut être payé avec 6 bobs et 20 centimes d'euros.

Une association a été créée pour gérer les comptes des adhérents via une application en Python. La classe Compte a été développée avec des propriétés telles que le numéro de compte, le numéro d'adhérent composé d'une lettre suivie de 6 chiffres, le numéro de téléphone et le solde du compte. Lors de la création d'un compte, toutes les propriétés doivent être renseignées sauf le solde qui est mis à 0. Les méthodes de cette classe sont les suivantes :

Méthodes de la classe Compte	Description
`__init__(self, numero, adherent, email)`	Crée un nouveau compte et met le solde à 0.
`ajouter(self, somme)`	Ajoute la somme au solde.
`enlever(self, somme)`	Enlève la somme au solde.
`est_debiteur(self)`	Renvoie Vrai si le solde du compte est strictement négatif.

Partie 1 : Création de la classe compte

On a commencé à écrire la classe Compte.

```
class Compte:
    def __init__(self, numero, adherent, email):
        self.numero = numero
        self.adherent = adherent
        self.email = email
        self.solde = 0
```

1. Écrire sur la copie la méthode `enlever`.
2. Écrire sur la copie la méthode `ajouter`.
3. Écrire sur la copie la méthode `est_debiteur`.

Partie 2 : Utilisation de la classe compte

Monsieur Mebarek souhaite rejoindre la communauté des utilisateurs du bob, déposer 400 EUR sur son compte en bob pour acheter une magnifique statuette à 43 bobs.

1. Écrire la ligne de code en langage Python permettant de créer le compte cpt_0123456A dont le numéro est "0123456A", le titulaire, Monsieur Mebarek, a le numéro d'adhérent suivant "B128672" et son adresse email est "r.mebarek@bleu.fr"

2. Écrire la ligne de code en langage Python permettant d'enlever 43 bobs du compte de Monsieur Mebarek.

3. Monsieur Mebarek souhaite envoyer 50 bobs de son compte "cpt_0123456A" vers un autre compte "cpt_2468794E". On a besoin pour cela d'ajouter une méthode à la classe Compte.
   Écrire la méthode `envoyer(self, autre_compte, montant)` qui transfère le montant passé en paramètre vers un autre compte. On suppose que le compte courant est suffisamment approvisionné. On utilisera les autres méthodes de la classe Compte sans en modifier directement les attributs.

Partie 3 : Gestion des comptes

Pour faciliter la gestion des comptes, on a créé une liste Python nommée `liste_comptes` qui contient tous les comptes des adhérents. Cette liste est composée d'objets de type Compte.

Le gestionnaire de l'association souhaite envoyer des relances aux adhérents dont le compte est débiteur, c'est-à-dire dont le solde est négatif. Pour cela, il faut ajouter une fonction supplémentaire au programme en langage Python qui vérifie les soldes des comptes.

1. Écrire la fonction `rechercher_debiteurs` qui prend en argument `liste_comptes` et renvoie une liste de dictionnaires dont la première clé est l'adresse email et la seconde clé le solde de son compte. Il contiendra uniquement les informations des comptes débiteurs.

   Exemple de résultat :
   `liste_debiteurs = [{'email': 't.dupont@rouge.com', 'solde': -60}, {'email': 's.carneiro@jaune.org', 'solde': -75}]`

2. Écrire la fonction `urgent_debiteur` qui prend en argument la liste de dictionnaires et renvoie l'email de l'adhérent le plus endetté. On suppose qu'aucun adhérent n'a la même dette.

2 Récursivité (4 points)

`Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.`

1. **a.** Définir brièvement ce qu'est une fonction récursive.

   b. On considère la fonction Python suivante :

```
1 def mystere(n):
2     if n < 10:
3         print(n)
4         mystere(n + 1)
```

   L'appel mystere(5) affiche successivement les nombres 5, 6, 7, 8 et 9. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 9.

2. On veut multiplier toutes les valeurs d'une liste python. Exemples:

- Pour `[5, 3]` la sortie est $5 \times 3 = 15$
- Pour `[1, 3, 4, 5]` la sortie est $1 \times 3 \times 4 \times 5 = 60$
- Pour `[9, 3, 0, 5]` la sortie est $9 \times 3 \times 0 \times 5 = 0$

Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive `mult` renvoie le produit de toutes les valeurs de la liste (supposée non vide et remplies de nombres entiers) passée en paramètre.
Exemple : `mult([5, 3])` renvoie 15.

```
1 def mult(lst, i = None):
2     # Ce premier if permet d'initialiser i au dernier indice de la liste
3     if i is None:
4         i = len(lst) - 1
5     if i == 0:
6         return ...
7     else:
8         return ...
```

3. La fonction `joindre_rec` ci-dessous permet de concaténer les chaines de caractères stockées dans une liste (non vide) passée en paramètre en les séparant par un espace.
Par exemple :

- `joindre_rec(["Hello", "world"])` renvoie `"Hello world"`.
- `joindre_rec(["Bonjour", "les", "terminales", "nsi"])` renvoie `"Bonjour les terminales nsi"`.
- `joindre_rec([""])` renvoie `""`.

```
1 def joindre_rec(lst, i = None):
2     # Ce premier if permet d'initialiser i au dernier indice de la liste
3     if i is None:
4         i = len(lst) - 1
5     if i == 0:
6         return lst[i]
7     else:
8         print(lst[i])
9         return joindre_rec(lst, i - 1) + " " + lst[i]
```

L'instruction `print(lst[i])` de la ligne 8 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en oeuvre au niveau des appels récursifs.

a. Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante :
`res = joindre_rec(["Alors,", "il", "est", "dur", "ce sujet ?"])`

b. Quelle valeur sera alors affectée à la variable res?

4. Écrire en Python une fonction `joindre` non récursive : cette fonction devra prendre en argument une liste de chaines de caractères et renvoyer les éléments de la liste séparée par des espaces. Elle devra donc renvoyer le même résultat que la fonction `joindre_rec` définie à la question 3.
Exemple : `joindre(["Bonjour", "les", "terminales", "nsi"])` renvoie "Bonjour les terminales nsi".

3 Camping (4 points)

Cet exercice porte sur les bases de données.

Dans cet exercice, on pourra utiliser les mots clés suivants du langage SQL : SELECT, FROM, WHERE, JOIN, ON, INSERT INTO, UPDATE, VALUES, OR, AND.
Leur utilisation est rappelée en annexe 1 en fin de sujet.

Le schéma relationnel sur lequel repose la base de données de la gestion d'un camping est le suivant:
Emplacement (NumEmplacement, Prix, Surface)
Vacancier (NumVavancier, Nom, Prenom, Email, Telephone)
Reservation (NumRes, #NumVavancier, #NumEmplacement, DateArrivee, DateDepart)

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole #.

1. Le couple (NumVacancier, NumEmplacement) peut-il servir de clé primaire pour la relation Reservation ? Justifier sa réponse.

2. **a.** Écrire une requête SQL donnant la liste des numéros d'emplacement et leur prix dans ce camping.

b. Écrire une requête SQL donnant l'adresse email du vacancier s'appelant "Ada Lovelace".

3. Dans cette base de données, les dates DateArrivee, DateDepart sont au format chaîne de caractères 'aaaa-mm-jj'. Par exemple, la chaîne '2021-09-01' représente le 1er septembre 2021.
On peut alors les manipuler en utilisant la fonction date(). Par exemple :

```
SELECT *
FROM Reservation
WHERE date(DateArrivee) < date('2021-09-01')
```

permet de donner toutes les réservations dont la date d'arrivée est avant le 1 septembre 2021.
Un client nommé "JOHN DOE" veut réserver un emplacement pour la nuit du 23 février 2023.
Écrire une requête donnant la liste des numéros d'emplacements occupées à cette date.

4. **a.** Le prix de l'emplacement 302 a changé et s'élève désormais à 25 euros. Quelle requête faut-il faire pour modifier ce prix dans la base de données ?

b. Ecrire une requête SQL affichant la surface des emplacements où a séjourné "Hedy Lamarr".

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
```