EXERCICE 3 (8 points)

Cet exercice porte sur la programmation objet, les structures de données, les réseaux et l'architecture matérielle.

On considère un réseau local constitué des trois machines de Alice, Bob et Charlie dont les adresses IP sont les suivantes :

- la machine d'Alice a pour adresse `192.168.1.1` ;
- la machine de Bob a pour adresse `192.168.1.2`.

On rappelle que l'adresse `192.168.1.255` est l'adresse de diffusion qui sert à communiquer avec toutes les machines du réseau local et le masque de ce réseau local est `255.255.255.0`. Cette adresse de diffusion est réservée et ne peut être attribuée à une machine.

Partie A

1. Donner une adresse IP possible pour la machine de Charlie afin qu'elle puisse communiquer avec celles d'Alice et Bob dans le réseau local. Justifier votre réponse en donnant toutes les conditions à respecter dans le choix de cette adresse IP

Ce réseau est utilisé pour effectuer des transactions financières en monnaie numérique `nsicoin` entre les trois utilisateurs. Pour cela, on crée la classe `Transaction` ci-dessous :

```
1 class Transaction:
2     def __init__(self, expediteur, destinataire, montant):
3         self.expediteur = expediteur
4         self.destinataire = destinataire
5         self.montant = montant
```

2. Toutes les dix minutes, les transactions réalisées pendant cet intervalle de temps sont regroupées par ordre d'apparition dans une liste Python. Dans un intervalle de dix minutes, Alice envoie dix `nsicoin` à Charlie puis Bob envoie cinq `nsicoin` à Alice. Écrire la liste Python correspondante à ces transactions.

Pour garder une trace de toutes les transactions effectuées, on utilise une liste chaînée de blocs (ou *blockchain*) dont le code Python est fourni ci-dessous. Toutes les dix minutes un nouveau bloc contenant les nouvelles transactions est créé et ajouté à la *blockchain*.

```
1 class Bloc:
2     def __init__(self, liste_transactions, bloc_precedent):
3         self.liste_transactions = liste_transactions
4         self.bloc_precedent = bloc_precedent # de type Bloc
5
6 class Blockchain:
7     def __init__(self):
8         self.tete = self.creer_bloc_0()
9
10    def self.creer_bloc_0(self):
11        """
12        Crée le premier bloc qui distribue 100 nsicoin à tous les
utilisateurs
13        (un pseudo-utilisateur Genesis est utilisé comme
expéditeur)
14        """
15        liste_transactions = [
16             Transaction("Genesis", "Alice", 100),
17             Transaction("Genesis", "Bob", 100),
18             Transaction("Genesis", "Charlie", 100)
19        ]
20        return Bloc(liste_transactions, None)
```

3. La figure 1 représente les trois premiers blocs d'une Blockchain. Expliquer pourquoi la valeur de l'attribut `bloc_precedent` du bloc0 est `None`.

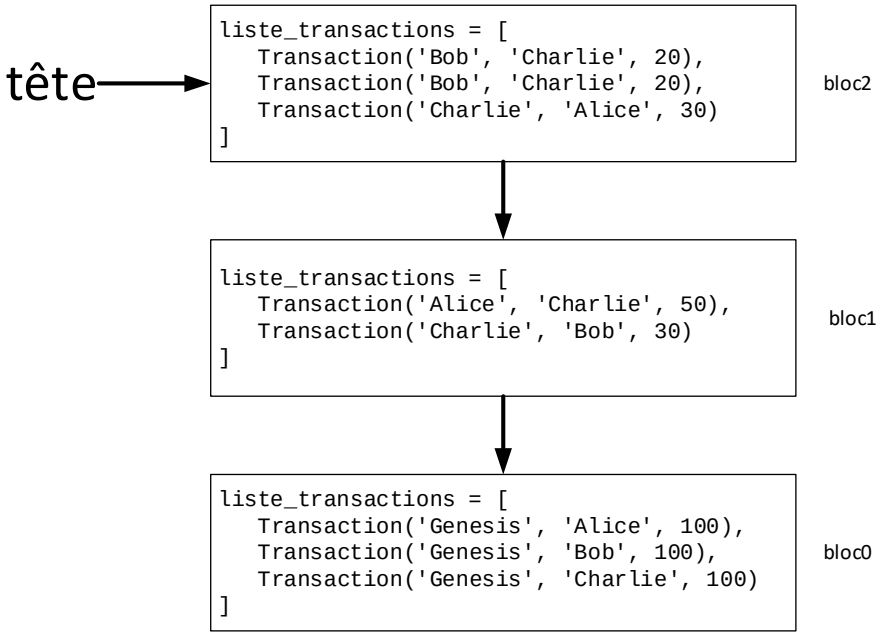

Figure 1. Blockchain

4. Donner la valeur de l'attribut `bloc_precedent` du bloc1 afin que celui-ci soit lié au bloc0.

5. À l'aide des classes `Bloc` et `Blockchain`, écrire le code Python permettant de créer un objet `ma_blockchain` de type `Blockchain` représenté par la figure 1.

6. Donner le solde en `nsicoin` de Bob à l'issue du bloc2.

7. On souhaite doter la classe `Blockchain` d'une méthode `ajouter_bloc` qui prend en paramètre la liste des transactions des dix dernières minutes et l'ajoute dans un nouveau bloc. Écrire le code Python de cette méthode ci-dessous.

```
def ajouter_bloc(self, liste_transactions):
    # A compléter
```

8. Lorsqu'un utilisateur ajoute un nouveau bloc à la Blockchain, il l'envoie aux autres membres. Ainsi chaque utilisateur dispose sur sa propre machine d'une copie identique de la Blockchain. Donner le nom et la valeur de l'adresse IP à utiliser pour effectuer cet envoi.

9. On souhaite doter la classe `Bloc` d'une nouvelle méthode `calculer_solde` permettant de renvoyer le solde à l'issue de ce bloc. Recopier et compléter sur votre copie le code Python de cette méthode :

```
def calculer_solde(self, utilisateur):
    if self.precedent is None: # cas de base
        solde = 0
    else:
        solde = ... # appel récursif : calcul du solde au bloc précédent
        for transaction in bloc.liste_transactions
            if ... == utilisateur:
                solde = solde - ....
            elif ... :
                ...
        return solde
```

10. Écrire l'appel à la fonction `calculer_solde` permettant de calculer le solde actuel de Alice

Partie B

Dans cette partie, on va améliorer la sécurité de la blockchain. Pour cela on enrichit la classe `Bloc` comme indiqué ci-dessous :

```
1 class Bloc:
2     def __init__(self, liste_transactions, bloc_precedent):
3         self.liste_transactions = liste_transactions
4         self.bloc_precedent = bloc_precedent
5         # Définition de trois nouveaux attributs
6         self.hash_bloc_precedent = self.donner_hash_precedent()
7         self.nonce = 0 # Fixé arbitrairement et temporairement à 0
avant le minage du bloc
8         self.hash = self.calculer_hash()
9
10    # Définition de trois nouvelles méthodes
11    def donner_hash_precedent(self):
12        if self.bloc_precedent is not None:
13            return self.bloc_precedent.hash
14        else :
15            return "0"
16
17    def calculer_hash(self):
18        """calcule et renvoie le hash du bloc courant"""
19        # Le code python n'est pas étudié dans cet exercice
20
21    def minage_bloc(self):
22        """modifie le nonce d'un bloc pour que son hash commence
par '00'"""
23        # A compléter
```

La fonction `calculer_hash` produit une chaîne de caractères appelée `hash` qui possède les propriétés suivantes :

- Le hash d'un bloc dépend de toutes les données contenues dans le bloc et uniquement de ces données ;
- Le calcul du hash d'un bloc est rapide et facile à calculer par une machine ;
- La moindre modification dans le bloc produit un hash complètement différent ;
- Il est impossible de déduire le bloc à partir de son hash.
- Si deux blocs ont le même hash, c'est qu'ils sont parfaitement identiques.

11. L'attribut `nonce` est de type entier. Miner un bloc signifie trouver une valeur de `nonce` de telle façon que l'attribut `hash` du bloc commence par les deux caractères "00". Compte tenu des propriétés précédentes, la seule façon de trouver cette valeur est de procéder à une recherche exhaustive. Expliquer en quoi consiste le fait de trouver une valeur par recherche exhaustive.

Dans la suite de l'exercice, on considère que tous les utilisateurs cherchent à miner le nouveau bloc. Le premier qui réussit, l'ajoute à la blockchain et gagne une récompense en `nsicoin`.

12. En justifiant votre réponse, donner la valeur de l'attribut `hash_bloc_precedent` du bloc0.

13. Sachant que le `hash` est écrit sur 256 bits, donner le calcul permettant d'obtenir le nombre de hash possibles.

14. Recopier et compléter sur votre copie le code python de la méthode `minage_bloc`.

```
1 def minage_bloc(self):
2     """modifie le nonce d'un bloc pour que son hash commence par
'00' en énumérant tous les entiers naturels en partant de 0."""
3     self.nonce = 0
4     self.hash = self.calculer_hash()
5     while ... :
6         self.nonce = ...
7         self.hash = ...
```