Exercice 1 : Conversion de nombres entre bases

Vous allez créer une classe Python qui permet de détecter la base d'un nombre (binaire, octal, décimal ou hexadécimal) et de convertir ce nombre dans une autre base. Suivez les étapes ci-dessous pour compléter l'exercice.

1. **Question 1 : Création de la classe et de l'attribut mot**
   - Créez une classe nommée Converter.
   - Ajoutez un constructeur __init__ qui prend un seul paramètre mot et le stocke comme un attribut de l'objet.
2. **Question 2 : Détection de la base avec getBase()**

   Ajoutez une méthode getBase() dans la classe. Cette méthode doit :
   - Détecter si le nombre appartient à l'une des bases suivantes :
     - Base 2 : commence par 0b ou 0B.
     - Base 8 : commence par 0o ou 0O.
     - Base 16 : commence par 0x ou 0X.
     - Base 10 : aucune des conditions ci-dessus. Vérifiez si le mot peut être converti avec int().
   - Retournez la base (2, 8, 10 ou 16).

   Outils à utiliser :
   - Méthode de chaîne de caractères : startswith() (pour vérifier les préfixes comme 0b, 0x, etc.).
   - La fonction int() pour vérifier si le mot est un entier décimal valide.
   - Gestion des erreurs avec try et except pour capturer les exceptions.
3. **Question 3 : Conversion de la base d'origine vers une base cible**

   Ajoutez une méthode convert(to_base) qui effectue les opérations suivantes :
   a. Appelez la méthode getBase() pour détecter la base du mot.
   b. Convertissez le mot en entier décimal en utilisant la fonction int() et la base d'origine.

c. Convertissez cet entier vers la base cible :

    1. Base 2 : utilisez bin().
    2. Base 8 : utilisez oct().
    3. Base 10 : utilisez int().
    4. Base 16 : utilisez hex().

d. Retournez le résultat sous forme de chaîne.

Exercice 2 :

Partie 1 : Classe abstraite NetworkComponent

1. Créez une classe abstraite nommée **NetworkComponent** pour modéliser les composants réseau.
2. Cette classe doit contenir :
    - Une méthode abstraite **info()** qui retourne les informations sur le composant réseau (doit être implémentée par chaque classe fille).
    - Une méthode abstraite **fonctionner()** qui décrit le rôle principal du composant réseau.

Partie 2 : Classe fille IPv4

1. Créez une classe IPv4 qui hérite de NetworkComponent.
2. Cette classe doit permettre la gestion des adresses IPv4. Elle doit contenir :
    - **Attributs** :
        - adresse (str) : L'adresse IPv4 à gérer.
    - **Méthodes** :
        - **isValidIPV4()** : Vérifie si une adresse IPv4 donnée est valide (utilisez le module re).
        - **getClasse()** : Retourne la classe de l'adresse (A, B, C, etc.).
        - **getMasque()** : Retourne le masque de sous-réseau sous forme d'adresse IP.

- **getIdReseau()** : Calcule et retourne l'identifiant réseau.
- **getIdHost()** : Calcule et retourne l'identifiant hôte.
- **getNbrOfHosts()** : Retourne le nombre maximal d'hôtes possibles dans le sous-réseau.

Partie 3 : Classe fille DNS

1. Créez une classe DNS qui hérite de NetworkComponent.
2. Cette classe doit permettre la gestion des noms de domaine. Elle doit contenir :
    - **Attributs** :
        - registre (dict) : Contient les noms de domaine enregistrés et leurs adresses IPv4 (clé : nom de domaine, valeur : IPv4).
        - cache (dict) : Contient les noms de domaine récemment résolus et leurs adresses IPv4.
    - **Méthodes** :
        - **ajouterAuRegistre(domain, ip)** : Ajoute un domaine et une adresse IPv4 au registre.
        - **supprimerDuRegistre(domain)** : Supprime un domaine du registre.
        - **resoudre(domain)** : Résout un domaine (recherche dans le cache, puis dans le registre).
        - **viderCache()** : Vide le cache des noms de domaine.
        - **listeRegistre()** : Retourne une liste des domaines dans le registre.
        - **listeCache()** : Retourne une liste des domaines dans le cache.

Partie 4 : Classe fille DHCP

1. Créez une classe DHCP qui hérite de NetworkComponent.
2. Cette classe doit permettre la gestion dynamique des adresses IP. Elle doit contenir :
    - **Attributs** :
        - poolAdresses (list) : Une liste des adresses IP disponibles pour l'attribution.

        - adressesAttribuees (dict) : Un dictionnaire contenant les adresses attribuées (clé : nom de l'appareil, valeur : adresse IP).

- **Méthodes** :
    - **ajouterAdresse(pool)** : Ajoute une ou plusieurs adresses au pool d'adresses disponibles.
    - **attribuerAdresse(appareil)** : Attribue une adresse IP disponible à un appareil donné (et la retire du pool).
    - **libererAdresse(appareil)** : Libère l'adresse IP d'un appareil et la réintègre au pool.
    - **listeAdressesDisponibles()** : Retourne la liste des adresses disponibles.
    - **listeAdressesAttribuees()** : Retourne la liste des adresses attribuées.