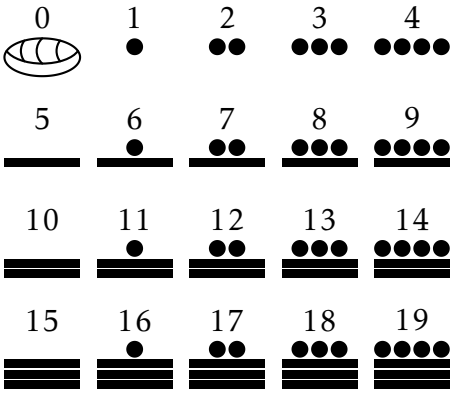
**Lycée Les 3 Sources** — **Bourg-Lès-Valence**
**NSI T^le** — Année **2025-26**

## Autotest n°3 – Correction

**Exercice 1 :** *Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.*

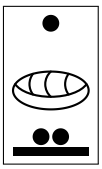
La civilisation *Maya* est une ancienne civilisation de Mésoamérique principalement connue pour ses avancées dans les domaines de l'écriture, de l'art, de l'architecture, de l'agriculture, des mathématiques et de l'astronomie.




La numération *Maya* est une numération positionnelle de base 20 (dite vigésimale) utilisant trois symboles pour former les "chiffres" :

- une coquille pour le zéro : 
- un point pour l'unité : ●
- un trait pour la valeur 5 : ▬

Les chiffres sont les suivants. Ils utilisent une numérotation additive.
Dans une version simplifiée de ce système, l'écriture d'un nombre se fait par empilement de "chiffres". Chaque étage correspond à un chiffre de poids 20 fois supérieur au poids du chiffre de l'étage inférieur.



Ainsi la valeur du chiffre de l'étage le plus bas est multipliée par $20^0$ soit 1, du second étage par $20^1$ , du troisième étage par $20^2$ , et ainsi de suite.
Exemple : la représentation *Maya* de l'entier 407 est la suivante.

1) Compléter le tableau suivant :

| Étage | Écriture *Maya* | Valeur du "chiffre" à l'étage | Valeur dans la conversion |
|---|---|---|---|
| 3 | ●●● | $1\times5+3\times1=8$ | $8\times20^2=3200$ |
| 2 | ● sur ▬ | $2\times5+1=11$ | $11\times20=220$ |
| 1 | ▬▬▬ | $3\times5=15$ | $15\times20^0=15$ |

2) Justifier que l'écriture *Maya* de l'entier 3435 est :

**Solution :** D'après la question précédente, on peut remarquer que :

$$3200+220+15=3435$$

Ce nombre correspond donc à 3435.

On **modélise l'écriture d'un entier** dans sa représentation *Maya* par une pile formée de listes. Chacune de ces listes est composée de trois entiers et modélise le "chiffre" d'un étage :

- le premier entier vaut 0 ou 1 suivant s'il y a ou non une coquille ;
- le deuxième représente le nombre de points ;
- le troisième représente le nombre de traits.

Ainsi, la modélisation *Maya* de l'entier 3435 est `[[0, 0, 3], [0, 1, 2], [0, 3, 1]]` et celle de l'entier 407 est `[[0, 2, 1], [1, 0, 0], [0, 1, 0]]`.
Le sommet de la pile se situe en fin de liste. La classe `Maya` est donnée à la page suivante.

3) Écrire une suite d'instructions permettant de créer une instance, nommée `M`, de la classe `Maya` qui modélise le nombre entier 3435. On utilisera la méthode `ajouter`.

```
M = Maya()
M.ajouter([0, 0, 3])
M.ajouter([0, 1, 2])
M.ajouter([0, 3, 1])
```

```
class Maya:
    def __init__(self):
        self.nombre = []

    def ajouter(self, chiffre):
        """ chiffre est une liste de longueur 3.
        La méthode empile le chiffre au sommet de la pile """
        self.nombre.append(chiffre)

    def retirer(self):
        """ depile et renvoie le chiffre qui etait au sommet de la pile """
        if not self.estVide():
            return self.nombre.pop()

    def estVide(self):
        return self.nombre == []
```

4) Compléter la méthode `nbEtages` de la classe `Maya`. Celle-ci renvoie le nombre de "chiffres" utilisés pour écrire le nombre correspondant en écriture Maya. On pourra utiliser l'attribut `nombre`.

```
    def nbEtages(self) :
        """ renvoie le nombre de chiffres de la pile """
        return len(self.nombre)
```

**De l'écriture *Maya* à l'écriture décimale**

5) Écrire une fonction `valeurChiffre` ayant pour paramètre une liste `L`. Celle-ci renvoie la valeur de l'entier associé à la liste `L = [c, p, t]` où `c` (de valeur 0 ou 1) indique la présence d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant un "chiffre" *Maya*.

Exemple :

```
>>> valeurChiffre([0, 2, 3])
>>> 17
```

```
>>> valeurChiffre([1, 0, 0])
>>> 0
```

```
def valeurChiffre(L):
    c, p, t = L
    return p + 5*t
```

6) Compléter la méthode `MayaToDec` de la classe `Maya`. Cette méthode renvoie la valeur de l'entier associé à l'objet `Maya`. On pourra utiliser les méthodes `estVide`, `nbEtages` et `retirer`.

```
    def MayaToDec(self):
        # renvoie le nombre entier correspondant a la
        # modelisation Maya de l'instance courante
        coeff = 20**(self.nbEtages()-1)
        ch_Dec = 0
        while not self.estVide():
            ch_Maya = self.retirer()
            ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
            coeff = coeff // 20
        return ch_Dec
```

**De l'écriture décimale vers sa modélisation *Maya***

On considère que la fonction DecToVige est déjà écrite. Celle-ci prend en paramètre un entier n et renvoie la décomposition en base 20 de celui-ci sous la forme d'une liste $[a_0, a_1, \ldots, a_p]$ telle que :

$$n = a_0 \times 20^0 + a_1 \times 20^1 + a_2 \times 20^2 + \ldots + a_p \times 20^p$$

```
>>> DecToVige(3435)
[15, 11, 8]
```

```
>>> DecToVige(407)
[7, 0, 1]
```

7) Écrire la fonction `decompChiffre` qui prend en paramètre un entier `n` compris entre 0 et 19 et renvoie la liste `[c, p, t]` où `c` vaut 0 ou 1 et indique la présence ou non d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant le "chiffre" *Maya* correspondant.

```
>>> decompChiffre(17)
[0, 2, 3]
```

```
>>> decompChiffre(0)
[1, 0, 0]
```

```
def decompChiffre(n):
    if n == 0:
        return [1, 0, 0]
    else:
        return [0, n%5, n//5]
```

8) Écrire la fonction `DecToMaya` qui prend en paramètre un entier `n` et renvoie la modélisation *Maya* d'un objet `M` de la classe `Maya` correspondant. On pourra utiliser les fonctions `DecToVige`, `decompChiffre` et les méthodes de la classe `Maya`. Par contre, on n'utilisera pas directement l'attribut `nombre` de cette classe.

```
>>> DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
```

```
>>> DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

```
def DecToMaya(n):
    M = Maya()
    for v in DecToVige(n):
        M.ajouter(decompChiffre(v))
    return M
```

**Opérations sur les nombres en modélisation *Maya***

On souhaite additionner des nombres directement à partir de leur modélisation *Maya*.

9) Écrire la méthode `multiplie` de la classe `Maya` qui renvoie un nouveau nombre `Maya` qui correspond au résultat de la multiplication par 20 d'un nombre en modélisation *Maya*.

```
>>> M = Maya()
>>> M.ajouter([0 ,0, 3])
>>> M.ajouter([0, 1, 2])
>>> M.nombre
[[0, 0, 3], [0, 1, 2]]
>>> M.multiplie().nombre
[[1, 0, 0], [0, 0, 3], [0, 1, 2]]
```

```
    def multiplie(self):
        # renvoie un nombre Maya correspondant au resultat
        # de la multiplication par 20 d'un nombre en modelisation Maya.
        M = Maya()
        M.ajouter([1, 0, 0])
        for chiffre un self.nombre:
            M.ajouter(chiffre)
        return M
```

On donne le code de la fonction `mystere` suivante :

```
def mystere(m1, m2, ret):
    c = 0
    p = (m1[1] + m2[1] + ret)%5
    if m1[1] + m2[1] + ret >= 5:
        ret = 1
    else:
        ret = 0
    t = (m1[2] + m2[2] + ret)%4
    if m1[2] + m2[2] + ret < 4:
        ret = 0
    else:
        ret = 1
    if (m1[0] == 1 and m2[0] == 1) or (p + t == 0 and ret == 1):
        c = 1
    return ([c, p, t], ret)
```

10) Donner les résultats renvoyés par les deux appels suivants :

```
>>> mystere([0, 1, 1], [0, 3, 1], 0)
([0, 4, 2], 0)
>>> mystere([0, 1, 1], [0, 4, 2], 0)
([1, 0, 0], 1)
```

11) Écrire une méthode `somme` de la classe `Maya` permettant d'ajouter à l'instance courante un autre nombre `maya2` de même taille en modélisation *Maya*. On pourra utiliser la fonction `mystere`.

```
    def somme(self, maya2):
        # ajoute maya2 à l'instance courante et renvoie le
        # resultat en modelisation Maya
        if self.nbEtages() == maya2.nbEtages():
            ret = 0
            for i in range(self.nbEtages()):
                chiffre, ret = mystere(self.nombres[i], maya2.nombres[i], ret)
                self.nombres[i] = chiffre
            if ret == 1:
                self.ajouter([0, 1, 0])
```

**Exercice 2 :** *Cet exercice porte sur le codage binaire, les bases de données relationnelles et les requêtes SQL.*

Cet exercice est composé de deux parties peu dépendantes entre elles. Lorsqu'il y a un accident, les pompiers essaient d'intervenir sur les lieux le plus rapidement possible avec les équipes et le matériel adéquats. On s'intéresse à l'étude d'un système informatique simplifié permettant de répondre à certaines de leurs problématiques.

Chaque pompier possède des aptitudes opérationnelles qui lui permettent de tenir un rôle. Lors du départ d'un véhicule (on parle d'agrès), il faut a minima un conducteur et un chef d'agrès. Pour simplifier, on considère que :

- un Véhicule Tout Usage (VTU) ne requiert que le duo conducteur et chef d'agrès, donc deux pompiers ;
- un Véhicule de Secours et d'Assistance aux Victimes (VSAV) requiert, en plus du duo conducteur et chef d'agrès, un équipier, donc trois pompiers ;
- un Fourgon Pompe Tonne (FPT) requiert, en plus du duo conducteur et chef d'agrès, un chef d'équipe et un équipier, donc quatre pompiers.

On souhaite entrer dans une base de données l'ensemble de ces informations afin de pouvoir conserver un historique des interventions.

**Partie A – Encodage binaire**

Afin de gagner de la place mémoire, on décide de coder l'ensemble des aptitudes sur un entier de 8 bits plutôt que d'écrire en toutes lettres "équipier", "chef d'équipe", etc. Cet ensemble d'aptitudes formera la qualification du pompier considéré.

- Un personnel non formé est codé par 0 ;
- l'aptitude "équipier" est codée par le bit de rang 0 (celui de poids le plus faible), soit $2^0$ ;
- l'aptitude "chef d'équipe" est codée par le bit de rang 1, soit $2^1$ ;
- l'aptitude "chef d'agrès" est codée par le bit de rang 2, soit $2^2$ ;
- enfin, l'aptitude "conducteur" est codée par le bit de rang 3, soit $2^3$.

Remarque : un chef d'équipe étant nécessairement équipier, on lui ajoute cette aptitude dans son codage. De même, un chef d'agrès est nécessairement un chef d'équipe et un équipier : on lui ajoute ces aptitudes dans son codage.

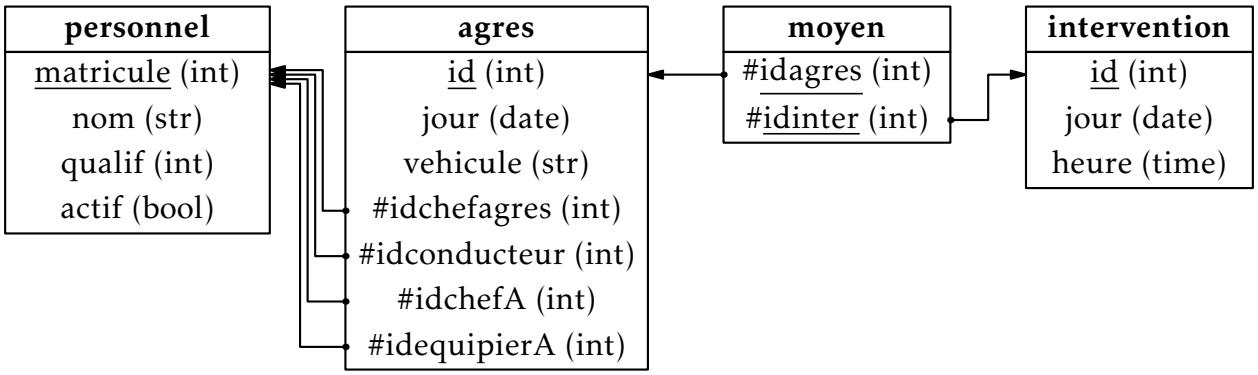
1) Justifier que la qualification décimale 11 correspond à un chef d'équipe conducteur.
   **Solution :** On a $11 = 1011_2$. Il y a donc des 1 dans les rangs 3, 1 et 0. C'est donc un conducteur chef d'équipe (et donc aussi équipier).
2) Déterminer le codage décimal de la qualification chef d'agrès conducteur.
   **Solution :** L'écriture binaire est $1111_2$. Cela correspond à 15 en décimal.
3) Expliquer pourquoi dans la situation décrite, un pompier ne peut pas avoir de qualification dont le codage décimale est 4.
   **Solution :** L'écriture binaire serait $100_2$, ce qui veut dire uniquement chef d'agrès. S'auf qu'un chef d'agrès est aussi chef d'équipe et équipier. Ce n'est donc pas possible.
4) Avec ce codage sur un octet, indiquer combien de nouvelles aptitudes peuvent être définies.
   **Solution :** Il reste 4 bits pour rajouter 4 nouvelles aptitudes.

On considère que chacune des quatre aptitudes aurait pu être codée par une chaîne de 10 caractères dont chaque caractère utilise 1 octet en mémoire.

5) Choisir, avec justification, l'économie mémoire que le codage sur un entier de 8 bits permet de faire comparée au codage basé sur les chaînes de caractères : environ 10 %, 30 %, 50 %, 98 %.
   **Solution :** On a 1 octet contre 40, ce qui fait environ 98 % d'économie.

**Partie B**

On considère maintenant la base de données relationnelle suivante :




La table `personnel` est composée :

- du matricule unique du pompier ;
- de son nom ;
- de sa qualification (selon le codage vu avant) ;
- d'un attribut qui précise s'il est actif (1) ou inactif (0).

| personnel | | | |
|---|---|---|---|
| matricule | nom | qualif | actif |
| 10 | 'Sam' | 3 | 1 |
| 16 | 'Charlot' | 1 | 0 |
| 31 | 'Red' | 23 | 0 |
| 83 | 'Vaillante' | 7 | 1 |
| 2501 | 'Marco' | 1 | 1 |
| 2674 | 'Aicha' | 23 | 1 |
| 3004 | 'Fatou' | 7 | 1 |
| 4044 | 'Abdel' | 19 | 1 |
| 4671 | 'Mamadou' | 17 | 1 |
| 5301 | 'Zoe' | 17 | 1 |
| 7450 | 'Medhi' | 3 | 1 |
| 8641 | 'Gaia' | 1 | 1 |
| 8678 | 'Kevin' | 17 | 0 |
| 8682 | 'Marie' | 1 | 1 |
| 9153 | 'Fred' | 23 | 1 |

La table `agres` est composée :

- de son identifiant ;
- du jour où l'agrès se tient prêt à intervenir ;
- du type de véhicule ;
- de l'identifiant du chef d'agrès ;
- de l'identifiant du conducteur ;
- de l'identifiant du chef d'équipe si le véhicule le nécessite, sinon le champ est à `NULL` ;
- de l'identifiant de l'équipier si le véhicule le nécessite, sinon le champ est à `NULL`.

La table `moyen` est composée :

- de l'identifiant de l'agrès appelé sur intervention ;
- de l'identifiant de l'intervention.

La table `intervention` est composée :

- de son identifiant ;
- du jour de début d'intervention (sauf pour les longues interventions où il correspond au jour de l'agrès) ;
- de l'heure de début d'intervention.

On rappelle que `COUNT(*)` permet de compter le nombre de lignes extraites lors d'une requête. Par exemple, pour afficher le nombre de personnes dans la table personnel, on exécute la requête :

```
SELECT COUNT(*) FROM personnel;
```

`DISTINCT` permet de retirer les doublons des réponses. Par exemple, pour afficher tous les noms distincts de la table personnel, on exécute la requête :

```
SELECT DISTINCT(nom)
FROM personnel;
```

On considère l'extrait de la base de données sur la page précédente et ci-dessous :

| moyen | |
|---|---|
| idagres | idinter |
| 0 | 0 |
| 2 | 3 |
| 2 | 4 |
| 3 | 4 |
| 4 | 4 |
| 9 | 5 |
| 17 | 6 |
| 22 | 7 |
| 23 | 8 |
| 24 | 8 |
| 24 | 9 |

| intervention | | |
|---|---|---|
| id | jour | heure |
| 0 | '2023-11-21' | '12:32:21' |
| 1 | '2023-11-22' | '22:20:00' |
| 2 | '2023-12-17' | '23:17:30' |
| 3 | '2024-02-15' | '01:44:06' |
| 4 | '2024-02-15' | '12:15:00' |
| 5 | '2024-03-02' | '04:58:12' |
| 6 | '2024-03-27' | '13:07:18' |
| 7 | '2024-05-31' | '05:17:12' |
| 8 | '2024-06-11' | '05:38:17' |
| 9 | '2024-06-11' | '15:08:56' |
| 10 | '2024-06-18'' | '07:42:33' |

| agres | | | | | | |
|---|---|---|---|---|---|---|
| id | jour | vehicule | idchefagres | idconducteur | idchefA | idequipierA |
| 0 | '2023-11-21' | 'VSAV' | 83 | 9153 | NULL | 10 |
| 2 | '2024-02-15' | 'VSAV' | 2674 | 4044 | NULL | 8641 |
| 3 | '2024-02-15' | 'FPT' | 9153 | 5301 | 8682 | 2501 |
| 4 | '2024-02-15' | 'VSAV' | 83 | 4671 | NULL | 7450 |
| 7 | '2024-02-29' | 'VSAV' | 9153 | 3004 | NULL | 2501 |
| 9 | '2024-03-02' | 'FPT' | 2674 | 5301 | 8682 | 8641 |
| 12 | '2024-03-21' | 'VTU' | 3004 | 5301 | NULL | NULL |
| 17 | '2024-03-27' | 'VSAV' | 3004 | 8682 | NULL | 10 |
| 18 | '2024-03-27' | 'VSAV' | 9153 | 5301 | NULL | 10 |
| 22 | '2024-05-31' | 'FPT' | 9153 | 4044 | 7450 | 8641 |
| 23 | '2024-06-11' | 'VTU' | 83 | 2674 | NULL | NULL |
| 24 | '2024-06-11' | 'VSAV' | 3004 | 4044 | NULL | 7450 |

6) Expliquer la différence entre une clé primaire et une clé étrangère.

**Solution :** Une clé primaire est un attribut dont les valeurs sont uniques dans une table. Une clé étrangère est un attribut d'une table qui correspond à une clé primaire d'une autre table.

7) Expliquer pourquoi la requête suivante génère une erreur pour l'extrait de données.

```
INSERT INTO moyen (idagres, idinter) VALUES (1, 5);
```

**Solution :** Il n'y a pas d'entrée dans `agres` qui a une `id` de 1. On ne peut donc pas insérer le couple `(1, 5)` dans `moyen`.

8) Proposer une requête SQL qui met à jour l'heure de l'intervention du « 15 février 2024 de 01 heure 44 minutes et 06 secondes » à « 10 heures 44 minutes et 06 secondes ».

```
UPDATE intervention SET heure = '10:44:06'
WHERE jour = '2024-02-15' AND heure = '01:44:06';
```

9) Préciser le résultat de la requête suivante pour l'extrait de données.

```
SELECT nom FROM personnel WHERE actif = 0;
```

**Solution :** On obtiendra les noms Charlot, Red et Kevin.

10) Proposer une requête SQL qui permet d'afficher les noms des personnels conducteurs actifs. On notera qu'un conducteur possède un attribut `qualif` supérieur ou égal à 16.

```
SELECT nom FROM personnel
WHERE qualif >= 16 AND actif = 1;
```

11) Écrire l'affichage obtenu après exécution des deux requêtes ci-dessous sur l'extrait de la base de données. Expliquer ce que chacune des requêtes affiche en général.

```
SELECT COUNT(*) FROM agres
WHERE jour = '2024-03-27';
```

**Solution :** On va obtenir 2, qui correspond au nombre d'agrès prêts à intervenir le 27 mars 2024.

```
SELECT COUNT(*) FROM moyen AS m
INNER JOIN agres AS a ON a.id = m.idagres
WHERE a.jour = '2024-03-27';
```

**Solution :** On va obtenir 1, qui correspond au nombre d'agrès disponibles le 27 mars 2024 qui sont sortis en intervention.

12) Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès assignés à un véhicule le 15 février 2024.

```
SELECT DISTINCT nom FROM personnel
JOIN agres ON matricule = idchefagres
WHERE jour = '2024-02-15';
```

13) Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès engagés en intervention le 11 juin 2024.

```
SELECT DISTINCT nom FROM personnel
JOIN agres ON matricule = idchefagres
JOIN moyen ON id = idagres
WHERE jour = '2024-06-11';
```