Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI T^le Année **2024-25**

Devoir surveillé n°3

Nom et prénom :

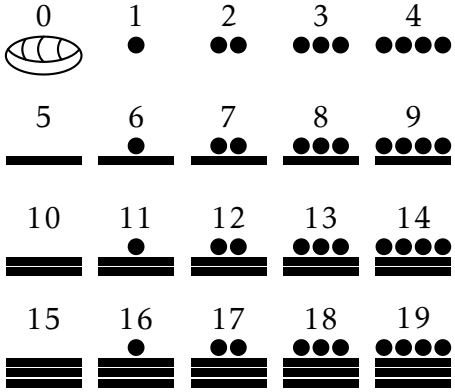
Exercice 1 : (14,5pt) *Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.*

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 : ▬

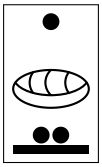
Les chiffres sont les suivants. Ils utilisent une numérotation additive :



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.

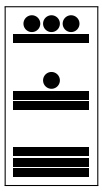
Exemple : la représentation *Maya* de l'entier 407 est la suivante.



1) Compléter le tableau suivant :

Étage	Écriture *Maya*	Valeur du "chiffre" à l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3200$
2			
1			

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



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. On dispose de la classe suivante :

```
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 == []
```

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`.

```

```

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 """
        ...
```

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
```

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**...
        ch_Dec = 0
        while ......................:
            ch_Maya = ...
            ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
            coeff = ...
        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 :

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

Exemple :

```
>>> 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.

Exemple :

```
>>> decompChiffre(17)
[0, 2, 3]
>>> decompChiffre(0)
[1, 0, 0]
```

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.

Exemple :

```
>>> DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
>>> DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

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*.

Exemple :

```
>>> 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]]
```

Nom et prénom :

```
    def multiplie(self):
        # renvoie un nombre Maya correspondant au resultat
        # de la multiplication par 20 d'un nombre en modelisation Maya.
        ...
```

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)


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

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():
            ...
```

Exercice 2 : (8,5pt) *Cet exercice porte sur les structures de Files*

Simon est un jeu de société électronique de forme circulaire comportant quatre grosses touches de couleurs différentes : rouge, vert, bleu et jaune. Le jeu joue une séquence de couleurs que le joueur doit mémoriser et répéter ensuite. S'il réussit, une couleur parmi les 4 est ajoutée à la fin de la séquence. La nouvelle séquence est jouée depuis le début et le jeu continue. Dès que le joueur se trompe, la séquence est vidée et réinitialisée avec une couleur et une nouvelle partie commence.

Exemple de séquence jouée : rouge→bleu→rouge→jaune→bleu

Dans cet exercice nous essaierons de reproduire ce jeu. Les quatre couleurs sont stockées dans un tuple nommé couleurs :

```
couleurs = ("bleu", "rouge", "jaune", "vert")
```

Pour stocker la séquence à afficher nous utiliserons une structure de file que l'on nommera `sequence` tout au long de l'exercice.

La file est une structure linéaire de type FIFO (First In First Out). Nous utiliserons durant cet exercice les fonctions suivantes :

Structure de données abstraite : File	
`creer_file_vide()`	: renvoie une file vide
`est_vide(f)`	: renvoie True si f est vide, False sinon
`enfiler(f, element)`	: ajoute `element` en queue de `f`
`defiler(f)`	: retire l'élément en tête de `f` et le renvoie
`taille(f)`	: renvoie le nombre d'éléments de `f`

En fin de chaque séquence, le Simon tire au hasard une couleur parmi les 4 proposées. On utilisera la fonction `randint(a, b)` de la bibliothèque `random` qui permet d'obtenir un nombre entier compris entre `a` inclus et `b` inclus pour le tirage aléatoire.

Exemple : `randint(1, 5)` peut renvoyer 1, 2, 3, 4 ou 5.

1) Compléter les `...` des lignes 3 et 4 de la fonction `ajout(f)` qui permet de tirer au hasard une couleur et de l'ajouter à une séquence. La fonction `ajout` prend en paramètre la séquence `f` et la modifie en rajoutant une couleur au format chaîne de caractères.

```
def ajout(f):
    couleurs = ("bleu", "rouge", "jaune", "vert")
    indice = randint(..., ...)
    enfiler(.................., ..................)
```

En cas d'erreur du joueur durant sa réponse, la partie reprend au début ; il faut donc vider la file `sequence` pour recommencer à zéro en appelant `vider(sequence)` qui permet de rendre la file `sequence` vide sans la renvoyer.

2) Ecrire la fonction `vider` qui prend en paramètre une séquence `f` et la vide sans la renvoyer.

```
def vider(f):
```

Le Simon doit afficher successivement les différentes couleurs de la séquence. Ce rôle est confié à la fonction `affich_seq(sequence)`, qui prend en paramètre la file de couleurs `sequence`, définie par l'algorithme suivant :

- on ajoute une nouvelle couleur à `sequence` ;
- on affiche les couleurs de la séquence, une par une, avec une pause de 0,5 s entre chaque affichage.

Une fonction `affichage(couleur)` (dont la rédaction n'est pas demandée dans cet exercice) permettra l'affichage de la couleur souhaitée avec `couleur` de type chaîne de caractères correspondant à une des 4 couleurs.
La temporisation de 0,5 s sera effectuée avec la commande `time.sleep(0.5)`. Après l'exécution de la fonction `affich_seq`, la file `sequence` ne devra pas être vidée de ses éléments.

3) Compléter, la fonction `affich_seq(sequence)` ci-dessous :

```
def affich_seq(sequence):
    stock = creer_file_vide()
    ajout(sequence)
    while not est_vide(sequence):
        c = ...
        ...
        time.sleep(0.5)
        ...
    while ............................. :
        ...
```

4) *Cette question est indépendante des précédentes : bien qu'elle fasse appel aux fonctions construites précédemment, elle peut être résolue même si le candidat n'a pas réussi toutes les questions précédentes.*

Nous allons ici créer une fonction `tour_de_jeu(sequence)` qui gère le déroulement d'un tour quelconque de jeu côté joueur. La fonction `tour_de_jeu` prend en paramètre la file de couleurs `sequence`, qui contient un certain nombre de couleurs.

- Le jeu électronique Simon commence par ajouter une couleur à la séquence et affiche l'intégralité de la séquence.
- Le joueur doit reproduire la séquence dans le même ordre. Il choisit une couleur via la fonction `saisie_joueur()`.
- On vérifie si cette couleur est conforme à celle de la séquence.
- S'il s'agit de la bonne couleur, on poursuit sinon on vide `sequence`.

- Si le joueur arrive au bout de la séquence, il valide le tour de jeu et `sequence` doit contenir toutes les couleurs de la séquence afin de pouvoir relancer un nouveau tour de jeu. Sinon le joueur a perdu et le jeu s'arrête.

La fonction `tour_de_jeu` s'arrête donc si le joueur a trouvé toutes les bonnes couleurs de `sequence` dans l'ordre, ou bien dès que le joueur se trompe.
Après l'exécution de la fonction `tour_de_jeu`, la file `sequence` ne devra pas être vidée de ses éléments en cas de victoire.

a) Afin d'obtenir la fonction `tour_de_jeu(sequence)` correspondant au comportement décrit ci-dessus, recopier le script ci-dessous et :

- Compléter le `...` à la ligne 7.
- Choisir parmi les propositions de syntaxes suivantes lesquelles correspondent aux ZONES A, B, C, D, E et F figurant dans le script et les y remplacer (il ne faut donc en choisir que six parmi les onze) :

`vider(sequence)`	`defiler(sequence)`
`enfiler(sequence, c_joueur)`	`enfiler(stock, c_seq)`
`enfiler(sequence, defiler(stock))`	`enfiler(stock, defiler(sequence))`
`while not est_vide(sequence):`	`while not est_vide(stock):`
`if not est_vide(sequence):`	`if not est_vide(stock):`
`affich_seq(sequence)`	

```
1  def tour_de_jeu(sequence):
2      ...                                  # ZONE A
3      stock = creer_file_vide()
4      while not est_vide(sequence):
5          c_joueur = saisie_joueur()
6          c_seq = ...                      # ZONE B
7          if c_joueur ... c_seq:           # À compléter
8              ...                          # ZONE C
9          else:
10             ...                          # ZONE D
11     ...                                  # ZONE E
12         ...                              # ZONE F
```

b) Proposer une modification pour que la fonction se rappelle par récursivité si le joueur trouve toutes les couleurs de la séquence (dans ce cas, une nouvelle couleur est ajoutée) ou s'il se trompe (dans ce cas, la séquence est vidée et se voit ajouter une nouvelle couleur). On pourra ajouter des instructions et des variables qui ne sont pas proposées dans la question a.