DS04 - Tale spé NSI 2 - POO - Vendredi 11 octobre 2024

Exercice 1

Implémenter une fonction qui prend en paramètre une liste de chaînes de caractères `mots` et un entier `taille` et qui renvoit une liste avec les éléments de `mot` dont la taille est inférieur à `taille`.

```
def limite(mots: list, taille: int):
  mots_l = []
  for mot in mots:
    if len(mot) <= taille:
      mots_l.append(mot)
  return mots_l

def limite2(mots: list, taille: int):
  return [mot for mot in mots if len(mot) <= taille]


assert limite(["a", "ab", "abc"], 3) == ["a", "ab", "abc"]
assert limite(["a", "ab", "abc"], 2) == ["a", "ab"]
assert limite(["a", "ab", "abc"], 1) == ["a"]
assert limite(["a", "ab", "abc"], 0) == []
```

Exercice 2

1. Programmer une classe `Eleve` avec :
    - les atrributs :
        - `nom` (`str`) : initialisé par le constructeur;
        - `points` (`int`) : initialisé à 10, valeur maximale de 20, valeur minimale de 0;
    - les méthodes :
        - `travaille` : fait gagner un point à l'élève (dans une limite de 20 points);
        - `procrastine` : fait perdre un point à l'élève (dans une limite de 0 point).
2. Créer un premier élève. Le faire travailler puis procrastiner. Afficher son nombre de points.
3. Compléter votre classe `Eleve` pour gérer le tutorat. Un élève "tuteur" peut "tutorer" un autre élève "tutoré", le tuteur et le tutoré gagant ainsi un point.
4. Créer un deuxième élève. Le faire tutorer le premier élève. Afficher leur nombre de points.

```
class Eleve:
  def __init__(self, nom):
    self.nom = nom
    self.points = 10

  def travaille(self):
    if self.points + 1 <= 20:
      self.points += 1

  def procrastine(self):
    if self.points - 1 >= 0:
      self.points -= 1

```

```
  def tutore(self, eleve):
    self.travaille()
    eleve.travaille()


eleve_1 = Eleve("eleve_1")
eleve_1.travaille()
eleve_1.procrastine()
print(eleve_1.points)

eleve_2 = Eleve("eleve_2")
eleve_2.tutore(eleve_1)
print(eleve_1.points, eleve_2.points)

```

```
10
11 11
```

Exercice 3

On veut programmer un jeu de puissance 4 en POO.

1. Pogrammer une classe `Joueur`, avec uniquement son constructeur pour le moment, qui initialise le nom du joueur :

   ```
   joueur_1 = Joueur("J1")
   joueur_2 = Joueur("J2")
   ```

2. Pogrammer une classe `Partie`, avec uniquement son constructeur pour le moment, qui initialise une grille vide :

   ```
   partie_1 = Partie(joueur_1, joueur_2)
   print(partie_1.grille)
   ```

   Résultat en console :

   ```
   [[None, None, None, None, None, None, None], \
   [None, None, None, None, None, None, None], \
   [None, None, None, None, None, None, None], \
   [None, None, None, None, None, None, None], \
   [None, None, None, None, None, None, None], \
   [None, None, None, None, None, None, None]]
   ```

3. Programmer une méthode `jouer` de la classe `Joueur` qui lui demande au joueur un numéro de colonne et renvoie cette valeur :

   ```
   joueur_1.jouer()
   joueur_2.jouer()
   ```

   Résultat en console :

   ```
   J1, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 3
   ```

```
J2, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 4
```

4. Programmer une méthode `placerPionJoueur` de la classe `Partie` qui demande au joueur une colonne et :
   - si possible place son pion en idiquant son nom le plus bas possible dans la grille et renvoie `True`;
   - renvoie `False` si ce n'est pas possible car la colonne est toute remplie.

   ```
   partie_1.placePionJoueur(joueur_1)
   partie_1.placePionJoueur(joueur_2)
   partie_1.placePionJoueur(joueur_1)
   partie_1.placePionJoueur(joueur_2)
   print(partie_1.grille)
   ```

   Résultat en console :

   ```
   J1, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 3
   J2, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 4
   J1, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 4
   J2, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 3
   [[None, None, None, None, None, None, None],
   [None, None, None, None, None, None, None],
   [None, None, None, None, None, None, None],
   [None, None, None, None, None, None, None],
   [None, None, None, 'J2', 'J1', None, None],
   [None, None, None, 'J1', 'J2', None, None]]
   ```

5. **question bonus (optionnelle)** Compléter votre code pour pouvoir jouer une partie complète.

```
'''
Jeu de de puissance 4
'''
class Partie:
  def __init__(self, joueur_1, joueur_2):
    self.grille = [[None for j in range(7)] for i in range(6)]
    self.joueur_1 = joueur_1
    self.joueur_2 = joueur_2

  def placePionJoueur(self, joueur) -> bool:
    col = joueur.jouer()
    ligne = 5
    while self.grille[ligne][col] is not None and ligne >= 0:
      ligne = ligne - 1
    if ligne == 0:
      return False
    else:
      self.grille[ligne][col] = joueur.nom
      return True


class Joueur:
  def __init__(self, nom):
    self.nom = nom

  def jouer(self) -> int:
    return int(input(f"{self.nom}, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : "))

```

```
29
30 joueur_1 = Joueur("J1")
31 joueur_2 = Joueur("J2")
32
33 partie_1 = Partie(joueur_1, joueur_2)
34 # print(partie_1.grille)
35
36 # joueur_1.jouer()
37 # joueur_2.jouer()
38
39 partie_1.placePionJoueur(joueur_1)
40 partie_1.placePionJoueur(joueur_2)
41 partie_1.placePionJoueur(joueur_1)
42 partie_1.placePionJoueur(joueur_2)
43 print(partie_1.grille)
```

```
J1, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 4
J2, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 3
J1, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 3
J2, à vous de jouer, indiquer le rang d'une colonne (0 à 6) : 4
[[None, None, None, None, None, None, None], [None, None, None, None, None, None, None], [N
```