All aboard the Infinity Train !

Dans ce projet, nous allons jouer au chef de gare en composant des trains. On sait ce qu'est un train, mais comment le définir, l'envisager virtuellement par programmation ?

1. Pour démarrer, il n'y a qu'une locomotive qui a un nom (le nom du train), et qui n'est pas un wagon.
2. Puis on accroche un premier wagon ayant un numéro (1) un nom ("wagon1"). Il faut un moyen de dire comment ce wagon est accroché, donc il faut aussi que le wagon sache qui est le wagon précédent, ici la locomotive (qui est donc un None wagon). Donc connaissant ce wagon, puisqu'il sait qu'il est attaché à la locomotive, il représente aussi le train.

   Train : Wagon(None,1,"wagon1") $< \boxed{None(Locomotive)} - - \boxed{Wag_1}$
3. Puis on accroche un second wagon (2, "wagon2") au premier wagon en lui signifiant donc que le wagon précédent est "wagon1" qui sait lui-même que le wagon précédent est la locomotive, donc None. Si on connait le dernier wagon, on connait donc toute la composition du train :

   Train: Wagon(Wagon(None,1,"wagon1"),2,"wagon2) $< \boxed{None(Locomotive)} - - \boxed{Wagon_1} - - \boxed{Wagon_2}$
4. Idem, un troisième wagon (3, "wagon3"), connaissant ce wagon, on connaît tout le train

   Train : Wagon(Wagon(Wagon(None,1,"wagon1"),2,"wagon2),3,"wagon3")

   $< \boxed{None(Locomotive)} - - \boxed{Wagon_1} - - \boxed{Wagon_2} - - \boxed{Wagon_3}$
5. *etcaetera, etcaetera, …*

   $< \boxed{None(Locomotive)} - - \boxed{Wagon_1} - - \boxed{Wagon_2} - - \boxed{Wagon_3} - - \boxed{Wagon_4} - - \ldots\ldots - - \boxed{Wagon_n}$

Si on résume :

- un **Train** est défini par un **nom** et par la connaissance du **dernier wagon**
- un **Wagon** a un **nom** et un **numéro** (ou autre chose!) et sait à quel wagon **précédent** il est attaché
- en partant du dernier wagon, on remonte les wagons précédents et on arrive à la locomotive : on a alors parcouru tout le train, comme si on remontait une **chaîne** de wagons.

Construisons donc d'abord les wagons, puis le train.

Travail 1 – Définissons les wagons par une classe Wagon

Construire les 2 méthodes suivantes de la classe Wagon :

- Le constructeur de la classe **Wagon** dont les attributs sont **precedent**, **numero** et **nom**
- Surcharge de la méthode spéciale __str__ qui permettra d'utiliser print pour afficher les attributs du wagon

```
class Wagon:
    '''
    Creation d'un wagon que l'on attache à une suite de wagons, et ayant
    un nom, et un numero  (non forcement sa position dans le convoi)
    (0 = locomotive, 1 = premier wagon, ...)
    On pourra donner un numero et un nom par défaut !
    attributs : precedent, numero, nom
    '''

    def __init__( … ):
        # Votre code

    def __str__( … ):
        # Votre code

w = Wagon(1, "beauWagon")
print (w)
```

Travail 2 – Définissons un début de train par une classe Train

Construire les 2 méthodes suivantes de la classe Train :

- le constructeur de la classe **Train** dont les attributs sont **nom**, **numero** et **dernier wagon**,
- la méthode **estVide()** qui permettra de savoir si le train est vide

```
class Train:

    '''
    Creation d'un train : on le caractérise par son nom et dernier_wagon puisque
    ce wagon est reli'e aux autres (le connaissant, on a acces a l'ensemble du train)
    attributs : nom, dernier_wagon
    '''

    def __init__( ... ):
        # Votre code

    def estVide( ... ):
        # Votre code is None  # "is" est à préférer à "==" pour comparer à None

train = Train()
print (train.estVide())
```

Travail 3 – Améliorons maintenant le train en ajoutant un wagon à la suite

Pour l'instant, nous il n'y a aucun Wagon dans notre train, on va donc devoir définir une méthode **ajouter(..)** qui ajoute automatiquement un wagon à la queue du train. Question à se poser :

- Quel sera alors le wagon précédent au wagon ajouté ?
- Quel sera alors le dernier wagon du train ?

Créez une méthode qui ajoute un wagon (aidez-vous d'un schéma si nécessaire)

```
class Train:
    ……
    def ajouter ( ):
      # Votre code
```

Travail 4 – Quelle est la longueur du train ?

Si le train est vide, on sait que sa longueur est nulle (on ne compte pas la locomotive), sinon, on compte le nombre de wagon, sachant que l'on n'a accès qu'au dernier wagon au début.

1. On pourrait créer une variable intermédiaire c qui désigne le dernier wagon : c = ... (dernier wagon). Que faut-il écrire pour accéder alors au wagon précédent ?
2. Vous êtes au dernier wagon, vous ne savez pas combien il y a de wagons avant vous. Que feriez-vous en situation réelle pour compter le nombre ?
3. Toujours dans la classe Train, compléter le code suivant qui surcharge la méthode spéciale __len__

```
def __len__( .... ):
    if ..... :
        return 0
    else:
        # Votre code
        Return ....
```

4. De même que pour les listes, on peut alors obtenir la longueur du train en tapant la commande :

print(len(train))

Travail 5 – Redéfinissons un affichage propre de la constitution du train par un print ?

Maintenant que vous savez compter le nombre de wagon, vous devriez être capable d'afficher la constitution du train (en chaînes de caractères). N'oubliez pas que l'on démarre toujours du dernier wagon !

Toujours dans la classe Train, complétez le code suivant qui surcharge la méthode spéciale __str__.

```
def __str__( ... ):
    c = ...     # dernier wagon
    affichage=""

    while  ...  :
        affichage +=  ...
        c =  ...
    affichage +=  ...
     return affichage
```

Travail 6 – Accéder aux attributs du n-ième wagon

Le fait important que l'on doit prendre en compte est que l'on part du dernier wagon à chaque fois ! Donc position du wagon (par rapport à la locomotive de rang 0) et position dans le programme (le dernier wagon est de rang 0) ne correspondent pas.

1. Pourtant, comment pourrait-on faire mathématiquement pour les relier ?
   Aide : Dans un train de longueur 5, le 5ème wagon est le wagon ... par rapport au dernier wagon, le 4ème wagon est le wagon ... , etc, etc
2. Après avoir fait cette petite gymnastique pour ne pas vous tromper dans la position du wagon dont on veut connaitre les attributs, n'y a-t-il pas des cas particuliers où on ne peut pas avoir les attributs ?
3. Comment se prémunir des cas indésirables ?

Toujours dans la classe Train, complétez le code de la méthode suivante :

```
def attributsNiemeWagon(self,n):
        '''
        le premier wagon est la tête du train, et le dernier est le dernier wagon ajouté
        '''

        if  ...  :
            return f"la locomotive s'appelle ......."
        elif  ... :
            return f"il n'y a pas autant de wagons donc pas de numéro {n}"
        elif  ... :
            return f"il n'y a pas de train avec un rang négatif, mais ça pourrait être le cas"
        else:
            ......
            return  ......
```

Travail 7 – Un wagon en trop, on l'enlève ?

Toujours dans la classe Train, complétez le code de la méthode suivante :

```
def enlever_wagon(self,n):
    '''
    Soit on utilise un train temporaire (une copie) que l'on crée à partir de l'autre
     sans prendre en compte le wagon que l'on veut enlever
    '''
```

Travail 8 – Il manque un wagon à un endroit, on l'insère !

De même que précédemment, il existe plusieurs façons de faire, on peut le faire soit comme si on le faisait physiquement, soit en copiant le train avec un nouveau wagon, ...

```
def inserer_wagon( .... ):
    '''
    On veut inserer un wagon en n-ieme position, il faut donc
    casser le lien entre le n-1 ieme wagon et le nieme existant
    '''
```

Travail 8 – D'autres idées de méthodes de la classe Train à développer

```
def changerNomNiemeWagon ( ):
    '''
    Pour changer le nom du n- ieme wagon
    '''
def trouverWagonAvecNom ( ):
    '''
    Pour trouver si un wagon particulier ( numero et ou nom ) se trouve dans le train
    renvoyer un booleen : True ou False
    '''
```

Travail 9 – Le train n'a pas ses numéros de Wagon dans l'ordre croissant ? Réorganisons-le !

```
def triNumWagonsOrdreCroissant ( ):
    '''
    Un simple tri sélection suffit ici mais on peut utiliser d'autres façons
    '''
```